Global Real-Time Semaphore Protocols: A Survey, Unified Analysis, and Comparison

— extended tech report —

Maolin Yang[†‡] Alexander Wieder[†] Björn B. Brandenburg[†]
[†]*Max Planck Institute for Software Systems* (*MPI-SWS*) [‡]*University of Electronic Science and Technology of China* (*UESTC*)

***Abstract*—All major real-time suspension-based locking protocols (or semaphore protocols) for global fixed-priority scheduling are reviewed and a new, unified response-time analysis framework applicable to all protocols is proposed. The newly proposed analysis, based on linear programming, is shown to be clearly preferable compared to all prior conventional approaches. Based on the new analysis, all protocols are directly compared with each other in a large-scale schedulability study. Interestingly, the *Priority Inheritance Protocol* (PIP) and the *Flexible Multiprocessor Locking Protocol* (FMLP), which are the two oldest and simplest of the considered protocols, are found to perform best.**

I. Introduction

When real-time applications synchronize access to shared resources with *binary semaphores* (*i.e.*, "mutexes" or suspension-based locks), a scheduler-specific real-time locking protocol is required to avoid unbounded *priority inversions* (*i.e.*, uncontrolled delays arising from the preemption of lock-holding tasks, discussed in detail in §III and §IV).

For *global fixed-priority* (G-FP) scheduling — the default multiprocessor real-time scheduling policy of VxWorks, QNX, Linux and many other RTOSs — several such protocols have been proposed, namely the PIP [12, 24], FMLP [4], P-PCP [12], OMLP [8], LB-PCP [19], and the FMLP$^+$ [5–7]. Given that virtually all practical systems have non-trivial synchronization requirements (if not explicitly at the application level, then at least implicitly at the kernel level, *e.g.*, due to shared I/O devices, communication buffers, or scheduler locks), a real-time locking protocol is a crucial component in any RTOS.

Unfortunately, it is largely unclear how to choose among the many available protocols. For one, most of the protocols have been analyzed with various, now-outdated techniques, which renders a direct comparison inconclusive. Further, while some of the more recent proposals (*e.g.*, the P-PCP and the FMLP$^+$) offer *potentially* improved blocking bounds (a mostly untested promise), they also introduce considerable design complexity and require sophisticated runtime mechanisms, which is highly undesirable from a pragmatic implementor's point of view.

Scope. To shed light on the bewildering array of choices, we take a fresh look at the real-time locking problem under G-FP scheduling. We first survey all major real-time semaphore protocols for global scheduling (§III) and then identify, and precisely define, six distinct types of delays (such as "direct" or "indirect" blocking) that arise due to mutual exclusion (§IV).

In the next step, to enable a fair comparison, we re-analyze all major protocols from first principles using a state-of-the-art methodology based on linear optimization (§V). The result is a *unified* response-time analysis framework for lock-using sporadic tasks that is applicable to all considered semaphore protocols. In particular, the proposed approach is sufficiently general to be instantiated even in the presence of *uncontrolled* (*i.e.*, potentially unbounded) priority inversions that arise in the absence of a proper real-time locking protocol.

Finally, we report on a large-scale, apples-to-apples comparison (§VI) of the available protocols based on the newly proposed unified analysis across a wide range of scenarios.

Contributions. This work advances the field in several ways. First, as shown in the evaluation (§VI), the newly proposed unified analysis is substantially more accurate than prior approaches: for all tested workloads subject to significant resource contention, the new analysis performed usually much better, and never worse. Second, our analysis is the first that is sufficiently general to characterize and *quantify* the effect of uncontrolled priority inversions in the absence of a real-time locking protocol. Third, our experiments are the first direct comparison of all major semaphore protocols for G-FP scheduling — for instance, the PIP and the FMLP have not been systematically compared in prior work, nor have the PIP and the P-PCP. And finally, our experiments paint a clear, although surprising picture: the PIP and the FMLP, which are the two oldest and simplest of the considered protocols, *always* performed best in all tested scenarios.

In a nutshell, this paper clears up the confusion surrounding the many available global locking protocol choices and makes a clear recommendation in favor of the PIP and the FMLP. As support for the former protocol is already specified by the POSIX real-time standard, and since the latter protocol is just as simple to support, this is a welcome outcome.

We begin with an overview of major protocols (§III) after briefly establishing essential definitions and notation.

II. Assumptions and Notation

We assume the standard sporadic task model of recurrent real-time processes executing upon a shared-memory multiprocessor.

Tasks. We consider a set of n *constrained-deadline sporadic tasks* $\tau = \{T_1, \ldots, T_n\}$ scheduled upon m identical processors. Each task T_i is characterized by a *worst-case execution time (WCET)* e_i, a *minimum inter-arrival time* (or *period*) p_i, and a *relative deadline* d_i, where $d_i \leq p_i$. Each task generates a potentially infinite sequence of jobs; we let $J_{i,j}$ denote the j^{th} job of T_i. We use τ^i to denote the set of all tasks in τ except T_i.

A job $J_{i,j}$ *arrives* at time $a_{i,j}$ and *finishes* at time $f_{i,j}$; its *response time* is given by $r_{i,j} = f_{i,j} - a_{i,j}$. During the interval $[a_{i,j}, f_{i,j})$, $J_{i,j}$ is *pending*, and while pending, it is either *ready* (and available for execution) or *suspended* (*i.e.*, blocked and not

The conference version of this extended tech report appears in the *Proceedings of the 36th IEEE Real-Time Systems Symposium (RTSS 2015)*, December 2015, IEEE.

available for execution). We assume that tasks suspend only to wait for a lock. Jobs arrive at least p_i time units apart ($a_{i,j+1} \geq a_{i,j} + p_i$) and must finish within d_i time units ($r_{i,j} \leq d_i$).

The *worst-case response time (WCRT)* of T_i, given by $r_i = \max_j\{r_{i,j}\}$, is the maximum response time of any job of T_i. The goal of a response-time analysis is to derive a *safe response-time bound* R_i such that $r_i \leq R_i \leq d_i$ in any possible schedule of τ. For brevity, we let J_i denote an arbitrary job of T_i.

For simplicity, we assume discrete time (*i.e.*, all parameters are multiples of a smallest quantum such as a processor cycle).

Scheduling. We consider G-FP scheduling, where jobs may freely migrate among processors. Each task is assigned a unique, fixed *base priority* shared by all jobs of the task. We assume that tasks are indexed by decreasing base priority (*i.e.*, *lower* indices correspond to *higher* base priorities). For brevity, when discussing a specific task T_i, we let τ^H and τ^L denote the sets of tasks with base priority higher and lower than T_i, respectively.

At runtime, locking protocols may assign temporarily elevated *effective* priorities. We let $\pi_i(t)$ denote the effective priority of task T_i at time t. At any time, a G-FP scheduler dispatches the (up to) m ready jobs with the highest effective priorities.

Resources. In addition to the processors, the tasks share n_r serially-reusable resources $\ell_1, \ldots, \ell_{n_r}$. We use $N_{i,q}$ to denote the maximum number of times that a job J_i accesses ℓ_q, and use $L_{i,q}$ to denote the *maximum critical section length* of T_i, *i.e.*, the maximum processor service required by T_i before it releases ℓ_q. (A task's WCET includes all its critical sections, *i.e.*, each $L_{i,q}$ is included in e_i.) The *priority ceiling* $\Pi(\ell_q) \triangleq \min\{i \mid N_{i,q} > 0\}$ is the base priority (*i.e.*, index) of the highest-base-priority (*i.e.*, lowest-index) task using ℓ_q. Since most of the locking protocols studied in this paper do not support nested critical sections, we require that tasks use at most one resource at any time.

Finally, we let $\eta_x(t) = \lceil (R_x + t)/p_x \rceil$ denote the maximum number of jobs of a task T_x that are pending in any contiguous interval of length t, and let $N^i_{x,q} \triangleq \eta_x(R_i) \cdot N_{x,q}$ denote an upper bound on the maximum number of requests for a resource ℓ_q that jobs of T_x can issue while a single job of T_i is pending.

With the essential definitions in place, we next review the major global real-time locking protocols proposed to date.

III. Survey of Global Real-time Locking Protocols

A real-time locking protocol serves to avoid unpredictable *priority inversions* [8, 24]. Intuitively, a priority inversion exists when a high-priority job J_h that *should* be scheduled (according to its base priority) is prevented from executing by a lower-base-priority job J_l (*e.g.*, if J_l holds a lock that J_h needs). Such a priority inversion is considered *predictable* if its duration can be bounded in terms of the maximum critical section length, and *unpredictable* or *uncontrolled* if it depends on the WCET of any task; we revisit this issue more formally in §IV.

In this paper, we focus on (binary) *semaphore* protocols (*i.e.*, *suspension-based* locks), where blocked jobs suspend to yield their processor to other ready jobs (if any).[1] In recent years, several such protocols have been proposed for global scheduling, which we briefly summarize in order of their appearance.[2]

The *Priority Inheritance Protocol* (**PIP**) [24] is the classic real-time locking protocol. It combines simple priority-ordered wait queues (*i.e.*, under contention, a lock is always granted to the highest-priority waiter) with *priority inheritance* (PI), a progress mechanism that prevents unpredictable priority inversion by raising the effective priority of lock-holding jobs when they block higher-base-priority jobs [24]. More precisely, a task's effective priority $\pi_i(t)$ is the maximum of its own base priority and the effective priority of any job that it blocks at time t [24].

As mentioned in §I, the PIP has considerable practical relevance: for instance, the POSIX real-time standard—supported by Linux, QNX, VxWorks and many other RTOSs that implement G-FP scheduling—specifies support for PI.

The PIP was originally designed for uniprocessors [24]. An analysis of the PIP under G-FP scheduling assuming non-nested critical sections, which we use as a baseline in our experiments (§VI), was later presented by Easwaran and Andersson [12].

The *Flexible Multiprocessor Locking Protocol* (**FMLP**) [4] is a suite of protocols for global and partitioned scheduling that was designed with simplicity as the guiding principle [4]. The FMLP variant relevant to this work is the *global* suspension-based FMLP, which combines PI (like the PIP) with simple FIFO wait queues (unlike the PIP), *i.e.*, the FMLP satisfies conflicting lock requests in *first-come first-served* order.

Only limited analyses of the FMLP under G-FP scheduling exist [4, 5]. Specifically, both prior analyses are intended for *suspension-oblivious* schedulability analysis [8]. In short, suspension-oblivious analysis pessimistically models blocking time as processor demand even if blocked tasks actually suspend, whereas *suspension-aware* analysis [8] accurately reflects that waiting tasks do not occupy processors. We revisit the analysis of suspensions in §IV and present the first suspension-aware analysis of the global FMLP in §V.

The *Parallel Priority Ceiling Protocol* (**P-PCP**) [12] is an extension of the PIP that attempts to avoid certain unfavorable blocking situations, albeit at the expense of additional effort at both design- and runtime. In particular, the P-PCP requires developers to configure a per-task parameter α_i, which is used by the runtime mechanism to restrict the maximum number of critical sections concurrently in progress, as described next.

Let $HPR(i, t)$ be the set of higher-base-priority jobs (relative to T_i) holding locks at time t, and similarly let $LPR(i, t)$ be the set of lower-base-priority jobs holding resources with priority ceilings exceeding the base priority of J_i. The P-PCP allows a job J_i to lock a resource ℓ_q if and only if ℓ_q is available and the following condition holds: $|HPR(i, t)| + |LPR(i, t)| < \alpha_i$. Otherwise, if ℓ_q is available but $|HPR(i, t)| + |LPR(i, t)| \geq \alpha_i$, then J_i suspends and the job in $LPR(i, t)$ with the shortest maximum critical section length inherits J_i's base priority. If ℓ_q is unavailable, the rules of the regular PIP take effect. An analysis sketch for the P-PCP was given by Easwaran and Andersson [12]; we derive a more accurate bound in §V.

Compared to the PIP and the FMLP, the P-PCP adds considerable complexity as it imposes additional PI rules and requires

[1] An alternative is *spin locks*, where blocked jobs execute a delay loop (often with interrupts disabled). However, spinning and non-preemptive execution require substantially different analysis that is beyond the scope of this paper.

[2] Supplementary example schedules that illustrate each protocol's rules are provided in Appendices A–D.

the RTOS to track the sets $HPR(i,t)$ and $LPR(i,t)$ at runtime. Further, the developer must configure appropriate α_i parameters, which together with the $L_{i,q}$ bounds must be known at runtime. Concerning the former, it is not obvious how to best choose α_i. The original P-PCP proposal [12] suggests to set $\alpha_i = n$ if $i \leq m$, and $\alpha_i = m$ otherwise; we follow this suggestion and call the resulting setup an (m,n)-configuration.

The family of $O(m)$ *Locking Protocols* (**OMLP**) [8, 9] is a suite of suspension-based locking protocols that are *asymptotically optimal* under suspension-oblivious analysis [8], in the sense that the worst-case blocking incurred by *any* task is within a (small) constant factor of the lower bound on blocking that is inevitably incurred by some task under *any* protocol [8]. (A detailed discussion of asymptotic blocking optimality is beyond the scope of this paper; see [5, 7–9] for details.)

From a pragmatic point of view, the *global OMLP* [8, 9], which is the OMLP variant relevant to this work, is a hybrid of the PIP and the FMLP: it relies on PI, but uses hybrid FIFO-priority queues. More precisely, for each resource, there is a FIFO queue of bounded length m, and a priority-ordered tail that feeds into the FIFO queue. Contending jobs enter the tail queue only if the FIFO queue is already full [8, 9].

The OMLP is specifically designed for suspension-oblivious analysis. We hence do not derive a new analysis of the OMLP, but consider the existing analysis [5] as a baseline in §VI.

Like the OMLP, the design of the *Generalized FIFO Multiprocessor Locking Protocol* (**FMLP$^+$**) [5, 7] is also driven by optimality concerns. In particular, it was noted that the original FMLP [4] is asymptotically suboptimal under suspension-aware analysis due to its use of PI. (In fact, on multiprocessors, generally *no* PI-based protocol is asymptotically optimal under suspension-aware analysis [5, 7].)

The FMLP$^+$ thus uses a novel progress mechanism called *restricted segment boosting* (RSB). Under RSB, a job's execution is split into an alternating sequence of *independent segments* and *request segments*. An independent segment starts when a job is released or when it releases a lock, and ends when it completes or issues a request; conversely, a request segment starts when a job issues a lock request, and ends when it releases the lock. To avoid unpredictable priority inversion, the lock-holding job with the earliest request segment start time is *priority-boosted*, *i.e.*, it is assigned an effective priority higher than that of any non-boosted job, which ensures that it is scheduled. Further, to ensure asymptotic optimality in the case of certain pathological scenarios [7], up to $m-1$ non-lock-holding jobs (in independent segments) with higher base priorities are *co-boosted*, *i.e.*, also given an effective priority above that of non-boosted jobs.

Like the original FMLP, the FMLP$^+$ uses simple per-lock FIFO queues to order conflicting requests.

Unlike PI-based protocols, the FMLP$^+$ supports *clustered* multiprocessor scheduling, which is a generalization of both global and partitioned scheduling. In prior work, the FMLP$^+$ has been evaluated under partitioned [6] and (non-global) clustered scheduling [7] and was found to perform well compared to alternatives [6, 7], despite asymptotic optimality having been the primary design goal. This, however, does *not* imply that the FMLP$^+$ will necessarily perform well (relative to other choices) under global scheduling, too—global scheduling allows for simpler semaphore protocols than either partitioned or clustered scheduling [5], which means that, from a purely empirical point of view, RSB might not actually be beneficial under global scheduling. We explore this question in our experiments (§VI).

For the sake of completeness, we also consider a new protocol that combines RSB with per-resource priority queues. While the choice of FIFO queues in the FMLP$^+$ is deliberate and essential to asymptotic optimality [7], there is, from a pragmatic point of view, no reason that precludes using *priority queues with RSB* (**PRSB**). We hence consider this variant in our evaluation, too.

Finally, we also study *locks without progress mechanism* as a base case. In the absence of either PI or RSB, all tasks execute at their base priority at all times. As a result, lock-holders that block high-priority tasks may be preempted at any time, which can cause prolonged priority inversion [24]. While long priority inversions are obviously detrimental to a system's ability to meet tight deadlines, prior work has not attempted to quantify the negative effects of locks without associated progress mechanism. Our unified analysis approach, introduced in the following two sections, is sufficiently general to cover this case as well, which we study as a lower bound on reasonable performance that any real-time locking protocol should exceed.

IV. Priority Inversion and Interference

Under G-FP scheduling, at any time, the (up to) m pending jobs with the currently highest (base) priorities are expected to be scheduled. Correspondingly, jobs are expected to be delayed only if all processors are busy executing higher-base-priority jobs. Any deviation from this expectation — that is, any disturbance of the normal G-FP schedule — is called a *priority inversion*, whereas expected delays due to the execution of higher-base-priority jobs are called *regular interference*.

If all tasks are independent, *i.e.*, in the absence of resource conflicts, jobs are delayed only by regular interference [3]. However, when jobs compete for semaphores, priority inversions arise due to two principal sources: task self-suspensions (*i.e.*, resource unavailability) and negative side effects of the employed progress mechanism (if any).

For example, Fig. 1 shows an FMLP$^+$ schedule that exhibits both effects: task T_5 locks resource ℓ_1 at time 1, thus starting to execute a request segment. Since T_5 executes the only (and thus earliest) request segment, it is priority-boosted, and since T_4 executes an earlier-started independent segment, it is co-boosted while T_5 holds ℓ_1. This has no effect as long as there are at most $m = 3$ jobs pending, but at time 4, when T_1 releases a job, the usual G-FP scheduling order is disrupted: T_3 is preempted (it is not co-boosted), whereas the lower-base-priority tasks T_4 and T_5 have an elevated effective priority and continue to execute—an RSB-induced priority inversion. A priority inversion due to a resource conflict occurs at time 5: T_1 attempts to lock ℓ_2, which T_3 already locked at time 3; consequently, T_1 self-suspends and the lower-base-priority jobs T_3–T_5 are scheduled instead.

To derive a safe response-time bound R_i, a precise definition of "priority inversion" is required. The exact definition, however, depends on *how* task self-suspensions are analyzed [8]. Under *suspension-aware (s-aware)* response-time analysis (*e.g.*, [6, 16, 17]), suspended jobs are accurately modeled to not

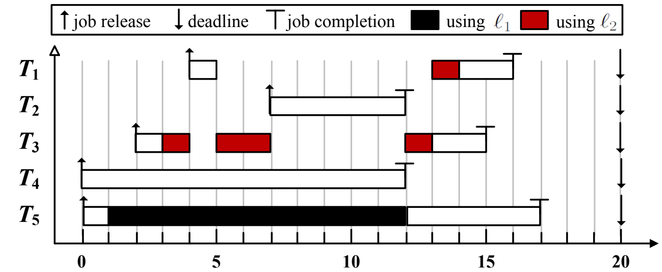


Fig. 1: Example FMLP$^+$ schedule ($m = 3$).

occupy a processor. In contrast, under *suspension-oblivious (s-oblivious)* analysis, each task's execution time is *inflated* to account for suspensions, which is safe but pessimistic, as it over-approximates each task's processor demand. On the flip side, the s-oblivious approach allows reusing existing response-time analyses that do not take self-suspensions or semaphores into account (*e.g.*, [2, 3]). Furthermore, since a part of the locking-related delays is implicitly accounted for as (inflated) interference, asymptotically lower bounds on maximum priority inversion lengths can be found under s-oblivious analysis [5, 8, 9].

In this work, we focus on the more precise s-aware approach, in which case "priority inversion" is defined as follows [8].

Def. 1: A job J_i is subject to *priority inversion* at time t iff J_i is pending but not scheduled, and fewer than m higher-base-priority jobs are scheduled at time t (*i.e.*, at least one processor is idle or occupied by a lower-base-priority job).

To analyze the individual causes of priority inversion, we further split Def. 1 into five different cases.

A. A Precise Categorization of Locking-Related Delays

For brevity, we denote any priority inversion induced by the execution of critical sections as *pi-blocking*. We distinguish among three types of pi-blocking. Let J_x denote a job scheduled at time t that is holding a resource ℓ_q, and suppose job J_i is pending but not scheduled at time t. Then J_x causes J_i to incur

- *direct pi-blocking* iff J_i is waiting for ℓ_q while fewer than m higher-base-priority jobs are scheduled at time t;
- *indirect pi-blocking* iff $x > i > \pi_x(t)$ and J_i is suspended and waiting for another resource ℓ_u ($\ell_u \neq \ell_q$) that is held by a job J_a and J_a is not scheduled at time t; and
- *preemption pi-blocking* iff $x > i > \pi_x(t)$, and J_i is ready but not scheduled at time t.

In addition, we define three types of the interference that J_i may incur due to a job J_x. In the following, J_x is assumed to be scheduled and J_i is assumed to be pending, not scheduled, and to *not* incur direct pi-blocking at time t. Then J_x causes

- *regular interference* at time t iff $x < i$;
- *co-boosting interference* at time t iff $x > i > \pi_x(t)$ and J_x is not holding any resource at time t; and
- *stalling interference* at time t iff $x > i$ and $\pi_x(t) \geq i$.

Despite their names, co-boosting and stalling interference are in fact cases of priority inversion according to Def. 1. We nonetheless use the term "interference" for these two cases since they are accounted for similarly to regular interference.

The FMLP$^+$ example depicted in Fig. 1 exhibits each type of delay. Consider task T_1. During $[5, 7)$, and again during $[12, 13)$, while T_1 waits for T_3 to release ℓ_2 and T_3 is scheduled, T_3 causes T_1 to incur *direct* pi-blocking. During $[7, 12)$, T_5 is scheduled and holds ℓ_1, while T_1 is still waiting for ℓ_2, which is held by the preempted T_3; T_5 thus causes T_1 to incur *indirect* pi-blocking during the interval. At the same time, T_4 is scheduled due to its elevated effective priority (recall that T_4 is co-boosted since it started its current independent segment before T_5 requested ℓ_1); as T_4 does not hold a resource, it causes T_1 to incur *co-boosting* interference during $[7, 12)$. In contrast, T_2 is *not* co-boosted and does not hold a resource, but is still scheduled while T_1 incurs priority inversion during $[7, 12)$; T_2 is thus considered to cause *stalling* interference. Finally, *preemption* pi-blocking is incurred by T_3 and caused by T_5 during $[4, 5)$ and $[7, 12)$. During the same two intervals, T_1 and T_2 are considered to cause *regular* interference since the two tasks have higher base priority.

Protocol	PI-Blocking D	I	P	Interference R	C	S	Prior Analysis
PIP	yes	yes	yes	yes	—	—	[12, 24]
FMLP	yes	yes	yes	yes	—	—	[4, 5]
P-PCP	yes	yes	yes	yes	—	yes	[12]
FMLP$^+$	yes	yes	yes	yes	yes	yes	[5, 7]
PRSB	yes	yes	yes	yes	yes	yes	—
no progress	yes	—	—	yes	—	yes	—

TABLE I: The types of pi-blocking (**D**irect, **I**ndirect, **P**reemption) and interference (**R**egular, **C**o-boosting, **S**talling) caused by each protocol.

Not all protocols cause all types of delay; Table I summarizes the types of pi-blocking and interference that a job may be exposed to under each of the studied protocols.

B. Basic Properties

With the precise definitions in place, we are now ready to begin our analysis of the individual types of delay. We start with three basic lemmas characterizing how each type occurs.

Lemma 1: If a job J_i incurs direct pi-blocking at time t, then it does not incur any other type of delay at time t.

Proof: By definition, when J_i incurs direct pi-blocking, it does not incur any type of interference. Further, J_i is suspended and waiting for some resource ℓ_q at time t while the job J_x holding ℓ_q is scheduled. Since J_i is suspended, it cannot incur preemption pi-blocking at time t. Since J_i requests at most one resource at a time, and since each resource is held by at most one job at any time, J_i is only waiting for ℓ_q at time t. Since J_x is scheduled, J_i cannot incur indirect pi-blocking at time t. ■

We next note that the different types of delays are intentionally defined to be mutually disjoint: while J_i may incur various type of delays due to different jobs at the same time, each delaying job causes J_i to incur at most one type of delay at a time.

Lemma 2: At any point in time, a job J_x causes a job J_i to incur at most one type of delay.

Proof: By exhaustive case analysis. If J_x causes J_i to incur direct pi-blocking at time t, then by Lemma 1 J_i will not incur any other type of delay due to any other job, including J_x, at time t. If J_x causes J_i to incur regular interference at time t, then $x < i$. Thus J_x cannot cause J_i to incur co-boosting interference, stalling interference, indirect pi-blocking, or preemption pi-blocking, which all require $x > i$. If instead J_x causes J_i to incur co-boosting interference at time t, then (i) $\pi_x(t) < i$, and (ii) J_x is not holding any resource at time t. Thus J_x cannot cause J_i to incur stalling interference due to (i), or any type of pi-blocking due to (ii). Further, if J_x causes J_i to incur stalling interference at time t, then $\pi_x(t) \geq i$. Thus J_x cannot cause J_i to incur indirect or preemption pi-blocking. Finally, if J_x causes J_i to incur preemption pi-blocking at time t, then J_i is ready at time t. Thus J_x cannot cause J_i to incur indirect pi-blocking. ■

Lemma 2 is essential to our analysis as it allows us to avoid double-counting the impact of individual critical sections. Lastly, we observe that all m processors are busy whenever a job is delayed without incurring direct pi-blocking.

Lemma 3: Under all considered protocols, and also in the absence of a progress mechanism, there are m jobs scheduled while a job incurs indirect pi-blocking, preemption pi-blocking, or any type of interference.

Proof: Suppose not. Then there exists a time t at which fewer than m jobs are scheduled while a job J_i incurs indirect pi-blocking, preemption pi-blocking, or any type of interference. Since G-FP scheduling is work-conserving, if J_i is not scheduled while there are fewer than m jobs scheduled at time t, then J_i is suspended at time t. There are two cases to consider.

Case 1: J_i is waiting for a resource that is held by another job J_a ($a \neq i$) at time t. Then J_a is ready at time t. Since there are fewer than m jobs scheduled at time t, J_a is scheduled at time t. Then, by definition, J_i incurs direct pi-blocking. By Lemma 1, J_i will not incur any other delay at time t. Contradiction.

Case 2: J_i is waiting for a resource ℓ_q not held by any job at time t. Among the considered protocols, this is possible only under the P-PCP, and only if $|HPR(i,t)| + |LPR(i,t)| \geq \alpha_i$, where $\alpha_i \geq m$ in the considered (m,n)-configuration. Therefore, there are at least m resource-holding, ready jobs that can be dispatched at time t. However, by initial assumption, at least one processor is idle. Contradiction. ■

Next, we introduce our main contribution: a unified G-FP response-time analysis for tasks with shared resources.

V. A Unified Analysis Framework

In contrast to conventional blocking analysis (*e.g.*, [4, 12]), we do not seek to manually identify and analyze an actual worst-case scenario. Rather, we model the problem of finding a safe response-time bound as a *linear optimization problem* that can be solved using *linear programming* (LP). As part of this LP formulation, we do not aim to identify a worst case, but rather identify (and rule out) *impossible* scenarios. The resulting optimal solution to our LP formulation is a safe upper bound on the maximum response time *across all schedules not shown to be impossible*, which includes any actually possible worst case.

To rule out impossible scenarios, we impose a number of constraints that encode invariants that hold in any schedule under G-FP scheduling and the respective analyzed locking protocols. Although most of the constraints are simple, together they are effective at eliminating pessimism. We begin by deriving a linear, locking-protocol-agnostic response-time model for resource-sharing tasks under G-FP scheduling, and then introduce the constraints that form the core of our analysis in §V-B

A. Response Time in a Fixed Schedule

Consider an arbitrary, but fixed job J_i in an arbitrary, but fixed G-FP schedule. To account for the delay that J_i incurs due to different types of interference, we let I^R_x, I^C_x, and I^S_x denote the *total cumulative* regular, co-boosting, and stalling interference, respectively, that J_i incurs due to jobs of task T_x. To express delays due to pi-blocking, we adopt the recently introduced notion of "blocking fractions" [6].

Def. 2: Let $\Re_{x,q,v}$ be the v^{th} request for resource ℓ_q by jobs of T_x while J_i is pending, and let $b_i^{x,q,v}$ be the actual amount of pi-blocking incurred by J_i due to $\Re_{x,q,v}$. The corresponding *blocking fraction* is defined as $X_{x,q,v} \triangleq b_i^{x,q,v}/L_{x,q}$.

In other words, a blocking fraction $X_{x,q,v}$ relates the actual delay incurred by J_i to the maximum critical section length $L_{x,q}$, where $X_{x,q,v} \leq 1$. For a given schedule, all blocking fractions can be easily calculated, but they are generally unknown *a priori*.

As defined in §IV-A, there are different ways in which a request causes pi-blocking. We hence introduce blocking variables specific to each type of pi-blocking: we let $X^D_{x,q,v}$, $X^I_{x,q,v}$, and $X^P_{x,q,v}$ denote the blocking fractions corresponding to only direct, indirect, and preemption pi-blocking, respectively. We further let $B^D_x \triangleq \sum_{\ell_q} \sum_{v=1}^{N^i_{x,q}} L_{x,q} \cdot X^D_{x,q,v}$ denote the *total direct pi-blocking* that J_i incurs due to jobs of T_x, where $N^i_{x,q}$ denotes the maximum number of requests for resource ℓ_q that tasks of T_x can issue while J_i is pending. Analogously, we define B^I_x and B^P_x to denote the total indirect and preemption pi-blocking, respectively, incurred by J_i due to jobs of T_x.

Based on these definitions, J_i's response time can be expressed as a simple linear function of the total pi-blocking and interference parameters of all tasks.

Lemma 4: The cumulative length of all intervals during which J_i is pending, not scheduled, and not incurring direct pi-blocking is given by

$$OD_i = \frac{1}{m} \cdot \left(\sum_{T_h \in \tau^H} I^R_h + \sum_{T_l \in \tau^L} \left(I^C_l + I^S_l + B^I_l + B^P_l \right) \right).$$

Proof: Consider any job J_x that is scheduled at a time t at which J_i is pending, not scheduled, and not subject to direct pi-blocking. The following cases exist: either **(a)** $x < i$, or $x > i$. If $x > i$, then either **(b)** $\pi_x(t) \geq i$, or $\pi_x(t) < i$. And if $\pi_x(t) < i$, then either **(c)** J_x is *not* holding any resource at time t, or **(d)** J_x *is* holding a resource at time t.

By definition, J_x causes J_i to incur regular interference in case (a), stalling interference in case (b), co-boosting interference in case (c), preemption pi-blocking in case (d) if J_i is ready, and indirect pi-blocking in case (d) if J_i is suspended.

Therefore, while J_i is pending, not scheduled, and not subject to direct pi-blocking at time t, other jobs are scheduled for a total of $\sum_{T_h \in \tau^H} I^R_h + \sum_{T_l \in \tau^L} (I^C_l + I^S_l + B^I_l + B^P_l)$ time units. By Lemma 3, there are m tasks scheduled while J_i incurs indirect or preemption pi-blocking, or any type of interference. Thus $\sum_{T_h \in \tau^H} I^R_h + \sum_{T_l \in \tau^L} (I^C_l + I^S_l + B^I_l + B^P_l) = m \cdot OD_i$. ■

Lemma 4 allows us to characterize J_i's response time.

Lemma 5: J_i's response time is bounded by

$$R_i = e_i + OD_i + \sum_{T_x \in \tau^i} B^D_x. \tag{1}$$

Proof: At any point in time while J_i is pending, J_i is either **(i)** scheduled, **(ii)** not scheduled and not incurring direct pi-blocking, or **(iii)** not scheduled and incurring direct pi-blocking. By definition, e_i bounds the duration of (i). By Lemma 4, OD_i bounds the duration of (ii). Further $\sum_{T_x \in \tau^i} B_x^D$ bounds the duration of (iii). The claim follows. ■

B. An LP-based Response-Time Bound

Eq. (1) does not immediately yield a practical schedulability test since the interference bounds and blocking fractions are unknown *a priori*. We close this gap by formulating the problem of finding a response-time bound as a linear optimization problem. In particular, we use Eq. (1) as the *maximization objective* and interpret all blocking fractions $X^D_{x,q,v}$, $X^I_{x,q,v}$, and $X^P_{x,q,v}$ as variables with domain $[0, 1]$, and all interference bounds I^R_x, I^C_x, and I^S_x as variables with domain $[0, \infty)$.

In the following, we establish constraints that encode invariants valid in any *possible* schedule, thereby guiding the LP solver to disregard variable assignments that reflect *impossible* schedules. We first present generic constraints valid under any protocol, followed by PI-, queue-, and protocol-specific constraints required for the analysis of the PIP and the FMLP, the two best-performing protocols in our evaluation (§VI). We further present the analysis of uncontrolled priority inversion, which has not been studied in prior work. The full analysis of the remaining protocols is provided in Appendices E–H.

Generic constraints. A task's *workload*, *i.e.*, the gross amount of processor cycles used by its jobs while J_i is pending, implies an upper bound on the total delay that it causes. In prior work [3], Bertogna and Cirinei derived the following upper bound on the workload of any sporadic task, which we use in Constraint 1.

Def. 3 (Bertogna et al.'s slack-aware workload bound [3]): During an interval of length R_i, task T_x's workload bound is

$$W_x(R_i) = N_x(R_i) \cdot e_x + \min(e_x,\ R_i + d_x - e_x - s_x - N_x(R_i) \cdot p_x),$$

where $s_x = d_x - R_x$ and $N_x(R_i) = \left\lfloor \frac{R_i + d_x - e_x - s_x}{p_x} \right\rfloor$.

Since a task causes pi-blocking or interference only when scheduled, Def. 3 implies the following constraint.

Constraint 1: In any G-FP schedule of τ:

$$\forall T_x \in \tau^i : I^R_x + I^C_x + I^S_x + B^D_x + B^I_x + B^P_x \leq W_x(R_i).$$

Proof: By Lemma 2, T_x causes J_i to incur at most one type of delay at any point in time. By definition, any T_x causes pi-blocking or interference only while executing. By Def. 3, T_x executes for at most $W_x(R_i)$ time units while J_i is pending. ■

Next, we bound the contribution of any single task to OD_i.

Constraint 2: In any G-FP schedule of τ:

$$\forall T_x \in \tau^i : I^R_x + I^C_x + I^S_x + B^I_x + B^P_x \leq OD_i.$$

Proof: By Lemma 4, OD_i characterizes the cumulative duration while J_i is pending, not scheduled, and not subject to direct pi-blocking. By Lemma 2, any job causes at most one type of delay at a time. Therefore, for the inequality to be invalid, there must exist a time t at which a job J_x causes J_i to incur some type of interference, indirect pi-blocking, or preemption pi-blocking while J_i is either scheduled, not pending, or incurring direct pi-blocking at time t. Clearly, J_i does not incur any delay if it is scheduled or not pending at time t. By Lemma 1, a job that incurs direct pi-blocking cannot incur any other type of delay at the same time. The claim follows. ■

Recall from §IV-B that the different types of pi-blocking are mutually exclusive. We express this property as follows.

Constraint 3: In any G-FP schedule of τ:

$$\forall T_x \in \tau^i,\ \forall \ell_q,\ \forall v : X^D_{x,q,v} + X^I_{x,q,v} + X^P_{x,q,v} \leq 1.$$

Proof: Suppose not: then there exists a schedule in which a request causes multiple types of pi-blocking at the same time. By Lemma 2, this is impossible. ■

Constraint 3 is instrumental in limiting pessimism because it prevents counting any request more than once. Next, we rule out stalling interference for tasks that do not share resources.

Constraint 4: In any G-FP schedule of τ:

$$\sum_{\forall \ell_q} N_{i,q} = 0 \implies \sum_{T_x \in \tau^L} I^S_x = 0.$$

Proof: Recall that J_i incurs stalling interference due to a job J_x if both J_x has an equal or lower effective priority and J_x is scheduled while J_i is pending but not scheduled (and not directly blocked), which under G-FP scheduling is possible only if J_i is suspended. Since J_i does not request any resources, it does not suspend, and hence cannot incur stalling interference. ■

Finally, we impose a trivial constraint to encode that no task is directly pi-blocked due to a resource that it does not request.

Constraint 5: In any G-FP schedule of τ:

$$\forall \ell_q \text{ s.t. } N_{i,q} = 0 : \sum_{T_x \in \tau^i} \sum_{v=1}^{N^i_{x,q}} X^D_{x,q,v} = 0.$$

Constraints 1–5 apply to all considered protocols. Next, we establish constraints that apply only to PI-based protocols.

PI-specific constraints. First, we note that co-boosting interference, which is specific to RSB, does not arise under PI.

Constraint 6: In any G-FP schedule of τ under PI:

$$\forall T_x \in \tau^L : I^C_x = 0.$$

Proof: Under PI, the effective priority of a job is elevated only if it is holding a (contested) resource. By definition, a job causes co-boosting interference only if it has an elevated effective priority while *not* holding a resource (recall §IV-A). ■

In preparation of the next constraint, we first make two simple observations about the effective priorities of scheduled jobs.

Lemma 6: The effective priorities of ready jobs are unique under all considered PI-based protocols.

Proof: Follows from the fact that base priorities are unique: for two ready jobs to have the same effective priority under PI, they both must hold a resource requested by the same job. Since jobs request at most one resource at a time, this is impossible. ■

Lemma 6 implies that only jobs with higher effective priority are scheduled when J_i incurs indirect or preemption pi-blocking.

Lemma 7: Under PI, if J_i incurs indirect or preemption pi-blocking at time t, then each of the m jobs scheduled at time t has an effective priority exceeding the base priority of J_i.

Proof: If J_i incurs indirect pi-blocking at time t, then it is directly delayed by another job J_a that is ready, but not scheduled. Thus no job with effective priority lower than $\pi_a(t)$ is scheduled. Due to PI, $\pi_a(t) \leq i$. By Lemma 6, the effective priorities of ready jobs are unique. There are thus m jobs with effective priority exceeding $\pi_a(t) \leq i$ scheduled at time t.

If J_i incurs preemption pi-blocking at time t, then it is ready and not scheduled at time t. Thus no job with effective priority lower than i can be scheduled at time t. Since by Lemma 6 effective priorities of ready jobs are unique, there are m higher-effective-priority jobs scheduled at time t. ∎

From Lemmas 6 and 7, we can infer that the m tasks with highest base priorities do not incur indirect pi-blocking, preemption pi-blocking, or any type of interference under PI.

Constraint 7: In any G-FP schedule of τ under PI:

$$i \leq m \implies \forall T_x \in \tau^i : I_x^R + I_x^C + I_x^S + B_x^I + B_x^P = 0.$$

Proof: Suppose not. Then there exists a time t at which J_i is pending, not scheduled, and not subject to direct blocking.

Case 1: If J_i is ready at time t, then under G-FP scheduling there must exist m ready jobs with effective priorities exceeding J_i's base priority; however, this is impossible because effective priorities are unique (Lemma 6) and since $i \leq m$.

Case 2: If J_i is suspended and waiting for a resource held by a job J_a, then J_a is ready, but not scheduled (otherwise J_i would incur direct pi-blocking), and hence J_i incurs indirect pi-blocking at time t. By Lemma 7, this requires the presence of m ready jobs with effective priorities exceeding J_i's base priority; as in Case 1, this is impossible.

Case 3: Finally, if J_i is suspended and waiting for a resource not held by any job, which among the considered protocols is possible only under the (m, n)-configured P-PCP, then $|HPR(i,t)| + |LPR(i,t)| \geq n$ (recall that $\alpha_i = n$ if $i \leq m$). Then all tasks, including T_i, are holding resources and J_i is thus ready at time t. Contradiction. ∎

A constraint for FIFO queues. We first consider FIFO queues, which are simpler to analyze as they provide starvation freedom.

Constraint 8: When using FIFO queues:

$$\forall \ell_q,\ \forall T_x \in \tau^i : \sum_{v=1}^{N_{x,q}^i} X_{x,q,v}^D \leq N_{i,q}.$$

Proof: In a FIFO queue, a request is directly delayed only by earlier-issued requests. Consequently, since jobs issue at most one request at a time, each time that J_i requests a resource, each other task can directly block J_i at most once. ∎

Next, we consider direct blocking in priority queues.

Constraints for priority queues. To begin with, we constrain direct blocking due to lower-priority tasks, which is trivially bounded by the number of requests issued by J_i.

Constraint 9: When using priority queues:

$$\forall \ell_q : \sum_{T_x \in \tau^L} \sum_{v=1}^{N_{x,q}^i} X_{x,q,v}^D \leq N_{i,q}.$$

Proof: When conflicting requests are satisfied in priority order, each time J_i requests a resource ℓ_q, at most one request from lower-base-priority tasks directly delays J_i. Hence, for each resource ℓ_q, at most $N_{i,q}$ requests for ℓ_q of tasks with lower base priority cause J_i to incur direct pi-blocking. ∎

Constraining direct pi-blocking by higher-priority tasks is considerably more involved since priority queues permit starvation of lower-priority requests. As a result, the analysis of higher-priority blocking resembles uniprocessor response-time analysis: a starving low-priority lock request will be satisfied only when there is no more higher-priority contention. To this end, we require a bound on the maximum resource-holding time.

Def. 4: We let $H_{x,q}$ denote a bound on the *maximum contested resource-holding time* of T_x, which is the maximum duration that any job J_x holds a resource ℓ_q while J_i is waiting to acquire ℓ_q. If $N_{x,q} = 0$, then trivially $H_{x,q} = 0$; otherwise, $H_{x,q}$ depends on the employed progress mechanism.

We begin by bounding $H_{x,q}$ under PI. Let sr_x^i denote the set of resources used by task T_x that have priority ceilings higher than the base priority of T_i, *i.e.*, $sr_x^i = \{\ell_q | N_{x,q} \neq 0 \wedge \Pi(\ell_q) < i\}$.

Lemma 8: Under PI, the maximum contested resource-holding time is bounded by $H_{x,q} = L_{x,q}$ if $x \leq m$, and by the least positive solution (if any) of the equation

$$H_{x,q} = L_{x,q} + \frac{1}{m}\left(\sum_{h<y} W_h(H_{x,q}) + \sum_{l>y \wedge l \neq z} \sum_{\ell_u \in sr_l^y} L_{l,u}^{x,q}\right)$$

if $x > m$, where $y = \min(x, i)$, $z = \max(x, i)$, and $L_{l,u}^{x,q} = \eta_l(H_{x,q}) \cdot N_{l,u} \cdot L_{l,u}$.

Proof: While holding ℓ_q, J_x is ready. If $x \leq m$, J_x is scheduled as it has one of the m highest effective priorities and since effective priorities of ready jobs are unique (Lemma 6). J_x thus holds ℓ_q for at most $L_{x,q}$ time units.

If $x > m$, then J_x can be preempted while holding ℓ_q, either due to regular interference or due to preemption pi-blocking. Since J_i is waiting for ℓ_q, $\pi_x(t) \leq \min(x, i) = y$ due to PI.

Thus, while J_i is waiting for J_x to release ℓ_q, **(i)** only tasks with base priority higher than y cause regular interference for J_x, and **(ii)** only tasks with base priority lower than y and effective priority higher than y cause preemption pi-blocking.

Regarding (i), by Def. 3, jobs with base priorities higher than y execute for at most $\sum_{h<y} W_h(H_{x,q})$ time units during an interval of length $H_{x,q}$.

Regarding (ii), jobs other than J_i and J_x with base priorities lower than y execute — while holding resources with priority ceilings higher than y — for at most $\sum_{l>y \wedge l \neq z} \sum_{\ell_u \in sr_l^y} L_{l,u}^{x,q}$ time units during an interval of length $H_{x,q}$.

By Lemma 3, there are m jobs scheduled whenever J_x incurs regular interference or preemption pi-blocking. Thus, $\frac{1}{m}\left(\sum_{h<y} E_h^{x,q} + \sum_{l>y \wedge l \neq z} \sum_{\ell_u \in sr_l^y} L_{l,u}^{x,q}\right)$ bounds the time in which J_x is not scheduled while J_i is waiting for J_x to release ℓ_q. In addition, J_x uses ℓ_q for at most $L_{x,q}$ time units. ∎

Next, we establish a bound $H_{x,q}$ under RSB, where resource-holding jobs are priority-boosted in FIFO order.

Lemma 9: Under RSB, $H_{x,q}$ is bounded by

$$H_{x,q} = L_{x,q} + \sum_{T_a \in \tau \setminus \{T_x, T_i\}} \max_{\ell_u \neq \ell_q} \{L_{a,u}\}.$$

Proof: Under RSB, resource-holding jobs are priority-boosted in order of request-segment start time. A job J_x holding a resource ℓ_q that J_i is waiting for is thus priority-boosted after each task in $\tau \setminus \{T_x, T_i\}$ has completed a critical section (not pertaining to ℓ_q, which is held by J_x). Since under RSB the priority-boosted resource holder is always scheduled, J_x is delayed for at most $\sum_{T_a \in \tau \setminus \{T_x, T_i\}} \max_{\ell_u \neq \ell_q} \{L_{a,u}\}$ time units before using ℓ_q for at most $L_{x,q}$ time units itself. ■

Finally, we bound $H_{x,q}$ in the absence of a progress mechanism. In the absence of a progress mechanism, lock-holders may be preempted at any time by newly released higher-priority jobs regardless of the priority of any waiters, which can cause prolonged contested resource-holding times.

Lemma 10: In the absence of a progress mechanism, the maximum contested resource-holding time is bounded by $H_{x,q} = L_{x,q}$ if $x \leq m$, and, if $x > m$, by the least positive solution (if any) of the equation

$$H_{x,q} = L_{x,q} + \frac{1}{m} \cdot \sum_{h < x \wedge h \neq i} W_h(H_{x,q}).$$

Proof: In the absence of a progress mechanism, $\pi_x(t) = x$ at all times that J_x is pending. If $x \leq m$, J_x is scheduled as it has one of the m highest base priorities. J_x thus holds ℓ_q for at most $L_{x,q}$ time units. If $x > m$, then J_x can be preempted while holding ℓ_q, but only due to regular interference by higher-priority jobs (other than J_i, which is suspended). By Def. 3, jobs (other than J_i) with base priorities higher than x execute for at most $\sum_{h<x \wedge h \neq i} W_h(H_{x,q})$ time units during an interval of length $H_{x,q}$. The claim follows analogously to Lemma 8. ■

Under RSB, each $H_{x,q}$ can be computed directly. Under PI and in the absence of a progress mechanism, fixed-point searches are required to determine $H_{x,q}$ for all but the m highest-priority tasks. If all bounds can be determined, the maximum time that J_i waits due to a single request can be bounded as follows.

Lemma 11: Suppose J_i issues a request for resource ℓ_q at time t_0, and acquires ℓ_q at time t_1. If access to ℓ_q is serialized with a priority queue, then $t_1 - t_0 < W_{i,q}$, where $W_{i,q}$ is the least positive solution less than d_i (if any) of the equation:

$$W_{i,q} = 1 + \max_{T_l \in \tau^L} \{H_{l,q}\} + \sum_{T_h \in \tau^H} \eta_h(W_{i,q}) \cdot N_{h,q} \cdot H_{h,q}.$$

Proof: When using priority queues, each time J_i issues a request for ℓ_q, it is delayed by at most one request for ℓ_q issued by a job with lower base priority. Thus, $\max_{T_l \in T^L} \{H_{l,q}\}$ bounds the delay J_i incurs when requesting ℓ_q due to requests from lower-priority jobs. For each higher-base-priority task T_h, there are further at most $\eta_h(W_{i,q})$ jobs while J_i is waiting to access ℓ_q, each of which issues at most $N_{h,q}$ requests for ℓ_q. Such jobs thus hold ℓ_q for at most $\sum_{T_h \in \tau^H} \eta_h(W_{i,q}) \cdot N_{h,q} \cdot H_{h,q}$ time units in an interval of length $W_{i,q}$. The claim follows. ■

If a wait-time bound $W_{i,q} < d_i$ can be found via a fixed-point search, it is possible to constrain direct pi-blocking.

Constraint 10: When using priority queues:

$$\forall T_x \in \tau^H,\ \forall \ell_q : \sum_{v=1}^{N_{x,q}^i} X_{x,q,v}^D \leq N_{i,q} \cdot \eta_x(W_{i,q}) \cdot N_{x,q}.$$

Proof: By Lemma 11, the maximum per-request delay of J_i is bounded by $W_{i,q}$. Thus, at most $\eta_x(W_{i,q})$ jobs of task T_x compete for ℓ_q while J_i is waiting to acquire ℓ_q once. Hence, across all of J_i's $N_{i,q}$ requests for ℓ_q, J_i is directly blocked by at most $N_{i,q} \cdot \eta_x(W_{i,q}) \cdot N_{x,q}$ requests by jobs of T_x. ■

Since $W_{i,q}$ is typically short (relatively to typical period lengths), quite often $\eta_x(W_{i,q}) = 1$, which limits the pessimism arising from the potential for starvation in priority queues.

Protocol-specific constraints. We first observe that stalling interference arises neither under the PIP nor the FMLP.

Constraint 11: In any PIP or FMLP schedule of τ:

$$\forall T_x \in \tau^L : I_x^S = 0.$$

Proof: Suppose not. Then there exists a schedule and a time t such that a job J_x, where $\pi_x(t) \geq i$, is scheduled while J_i is pending, not scheduled, and not subject to direct pi-blocking.

If J_i is ready and not scheduled, then no job with effective priority lower than J_i's effective priority is scheduled under G-FP scheduling. Further, according to Lemma 6, while J_i is ready, no other job has the same effective priority. Thus J_x will not cause J_i to incur any delay while J_i is ready and not scheduled.

If J_i is suspended and not subject to direct pi-blocking, then, under the PIP or FMLP, J_i incurs indirect pi-blocking. Thus, by Lemma 7, there are m jobs with effective priorities higher than J_i scheduled at time t. Contradiction. ■

The next two constraints limit indirect and preemption pi-blocking under the PIP and the FMLP. We begin with the PIP.

Constraint 12: In any G-FP schedule of τ under the PIP:

$$\forall \ell_q : \sum_{T_x \in \tau^L} \sum_{v=1}^{N_{x,q}^i} X_{x,q,v}^I + X_{x,q,v}^P \leq \sum_{T_h \in \tau^H} N_{h,q}^i.$$

Proof: Under the PIP, to cause indirect or preemption pi-blocking, a lower-base-priority job must inherit the priority of (and thus directly block) a job of a task with base priority higher than J_i. Since priority queues are used, each request of a higher-priority task is directly blocked by at most one lower-priority request. Thus, at most $\sum_{T_h \in \tau^H} N_{h,q}^i$ lower-priority requests for ℓ_q cause J_i to incur indirect or preemption pi-blocking. ■

Analogous to Constraint 12, we impose a constraint for the FMLP based on the combination of PI and FIFO queuing.

Constraint 13: In any G-FP schedule of τ under the FMLP:

$$\forall T_x \in \tau^L,\ \forall \ell_q : \sum_{v=1}^{N_{x,q}^i} X_{x,q,v}^I + X_{x,q,v}^P \leq \sum_{T_h \in \tau^H} N_{h,q}^i.$$

Proof: Under the FMLP, in order to cause J_i to incur indirect or preemption pi-blocking, a lower-base-priority job must inherit the priority of a job of a task with base priority higher than J_i. Since FIFO queues are used, each request of a higher-priority task is directly blocked at most once by *each* lower-priority task. As there are at most $\sum_{T_h \in \tau^H} N_{h,q}^i$ higher-base-priority requests, each lower-priority task causes J_i to incur indirect or preemption pi-blocking at most $\sum_{T_h \in \tau^H} N_{h,q}^i$ times. ■

Finally, we consider the case of priority- and FIFO-ordered locks without any progress mechanism.

Constraints for uncontrolled priority inversion. By design, the unified analysis framework introduced in §IV and §V-A does not make any assumptions regarding protocol-specific rules. In particular, it does not assume that a progress mechanism such as PI or RSB is employed. Our analysis can hence be instantiated even in the absence of a progress mechanism simply by leaving out all PI-, RSB-, and protocol-specific constraints.

Furthermore, any delays that arise under PI and RSB as a side effect of raised effective priorities can be ruled out.

Constraint 14: In any G-FP schedule of τ in the absence of a progress mechanism: $\forall T_x \in \tau^L : B_x^I + B_x^P + I_x^C = 0$.

Proof: Recall from §IV-A that, to cause indirect pi-blocking, preemption pi-blocking, or co-boosting interference, a lower-priority task $T_x \in \tau^L$ must have an elevated effective priority. However, in the absence of a progress mechanism, the effective priority of a job is always equal to its base priority. ■

In our analysis, uncontrolled priority inversion manifests as stalling interference: while J_i is waiting to acquire a resource that is held by a lower-priority, preempted job J_l, jobs of tasks of priority higher than l, but lower than i, may be scheduled for an extended duration. Conversely, uncontrolled priority inversion (and hence stalling interference) cannot be caused by tasks that cannot preempt jobs that share resources with J_i.

Constraint 15: In any schedule of τ in the absence of a progress mechanism:

$$\forall T_x \in \tau^L \text{ s.t. } \sum_{l>x} \sum_{N_{i,q}>0} N_{l,q} = 0 : \quad I_x^S = 0.$$

Proof: By contradiction. Suppose a lower-priority job J_x causes stalling interference although $\sum_{l>x} \sum_{N_{i,q}>0} N_{l,q} = 0$.

To cause stalling interference at time t, J_x must be scheduled while J_i is pending but not scheduled. Since all jobs execute at their base priority in the absence of a progress mechanism, this is possible only if J_i is suspended while it waits to acquire a resource ℓ_q held by another job J_a.

Since $\sum_{l>x} \sum_{N_{i,q}>0} N_{l,q} = 0$, it follows that $a < x$, and since J_a holds ℓ_q, it is ready at time t. By initial assumption, J_x causes stalling interference and hence is scheduled at time t. Under G-FP scheduling, the higher-priority, ready job J_a is thus scheduled at time t, too. However, this implies that J_i incurs direct pi-blocking at time t, and hence, by Lemma 1, J_i does not incur stalling interference at time t. Contradiction. ■

This concludes our analysis of the PIP, the FMLP, and locks without a progress mechanism. Due to space constraints, analogous constraints for the P-PCP, the FMLP$^+$, and the PRSB are provided in Appendices E–H.

C. Instantiating the Response-Time Analysis

A response-time bound R_i for a task $T_i \in \tau$ can be obtained with an LP solver by maximizing Eq. (1) subject to all applicable constraints, as listed in Table II. However, a circular dependency exists as the maximum number of possible requests $N_{x,q}^i$, which is required for each task $T_x \in \tau^i$, depends on both R_i and R_x.

This dependency can be resolved with an iterative fixed-point search. Starting from initial values $R_j^{(0)} = e_j$ for each $T_j \in \tau$, an updated response-time bound $R_j^{(k+1)}$ is repeatedly determined for each task by solving the LP based on the estimates

Protocol	Applicable Constraints			
	Generic	Progress	Queue	Protocol
PIP	1–5	6, 7	9, 10	11, 12
FMLP	1–5	6, 7	8	11, 13
Priority (no progress)	1–5	14–15	9, 10	—
FIFO (no progress)	1–5	14–15	8	—
FMLP$^+$	1–5	16–22	8	23–25
PRSB	1–5	16–22	9, 10	26
P-PCP	1–5	6, 7	9, 10	27–31

TABLE II: Overview of constraint applicability.
Constraints 16–31 are provided in Appendices E–H.

$R_1^{(k)}, \ldots, R_n^{(k)}$ determined in the previous iteration. The fixed-point search proceeds until either **(i)** a consistent fixed-point for all response-time bounds is found, *i.e.*, $R_j^{(k+1)} = R_j^{(k)} \leq d_j$ for each $T_j \in \tau$, or **(ii)** the preliminary response-time bound for some task exceeds its deadline, *i.e.*, after some iteration, $R_j > d_j$ for some $T_j \in \tau$. In case (i), the task set is deemed schedulable; in case (ii), failure is reported.

The fixed-point search is guaranteed to terminate because the analysis is monotonic with regard to response times: if the response-time estimate R_x of any task $T_x \in \tau^i$ increases, the resulting bound R_i may grow as a result, but it cannot decrease since Def. 3 and the bound $\eta_x(R_i) = \lceil (R_x + R_i)/p_x \rceil$ in the definition of $N_{x,q}^i$ are monotonic in R_x.

D. Accuracy of the Analysis

It is worth to consider two notions of accuracy. First, the LP variables (*i.e.*, blocking fractions and interference bounds) have continuous domains, whereas we assume discrete time in our system model (*e.g.*, typically processor cycles). Using real-valued variables is a valid relaxation that always results in safe bounds, but it can theoretically also result in an over-approximation of a few time units — since the objective is to *maximize* Eq. (1), adding integer constraints can result only in lowered response-time bounds. Any negative impact from this relaxation, however, is negligible compared to the pessimism inherent in current global schedulability analysis techniques.

A second potential concern pertains to the "completeness" of our analysis — we cannot preclude the possibility that the discovery of additional constraints could result in further improved blocking bounds. (In fact, the extensibility of the LP-based analysis approach is one of its key features [6].) However, the presented constraints encode invariants derived from the mechanisms employed by the respective protocols, and we strongly believe to have captured all major invariants.

Instead, we expect future improvements to come from refined task models (*e.g.*, the integration of control-flow information such as critical sections on conditional branches or minimum separation bounds between requests), from the integration of workload-specific information (*e.g.*, if only every second job of a task accesses a certain resource), and from the analysis of known arrival times of periodic tasks (*e.g.*, where offsets could be chosen such that jobs of certain tasks cannot conflict).

Finally, we note that the proposed unified analysis — with the presented constraints, and despite the use of continuous variables — is already more accurate than *any* of the prior approaches, as demonstrated by our experiments, which we discuss next.

VI. Empirical Comparison

We implemented the presented LP-based analysis in *SchedCAT* [23], using the *GNU Linear Programming Kit* (GLPK) as the underlying LP solver. Based on SchedCAT, we conducted a schedulability study to **(i)** assess to which extent our new LP-based analysis improves upon prior analyses and to **(ii)** identify the protocols that perform best across a wide range of scenarios.

A. Experimental Setup

Our experimental setup resembles in large parts the design of prior locking-related schedulability studies [5, 6, 16].

We considered two platforms with $m \in \{4, 8\}$ processors. For a given $n \geq m$, we randomly generated n implicit-deadline tasks according to the following procedure. A task's period p_i was randomly chosen from log-uniform distributions ranging over $[10ms, 100ms]$ (*homogeneous periods*) or $[1ms, 1000ms]$ (*heterogeneous periods*). For each task, a utilization $u_i \in (0, 1]$ was chosen according to an exponential distribution with a mean value of either 0.1 (*light*) or 0.25 (*medium*), and the task's WCET e_i was set to $e_i = p_i \cdot u_i$ (rounded to the next microsecond). Task priorities were assigned using the DkC heuristic [10] and three earlier heuristics recommended by Easwaran and Andersson for use with the P-PCP [12]; a task set was deemed schedulable if it was shown to be schedulable using any of the four heuristics.

The number of shared resources was varied across $n_r \in \{m/4, m/2, m, 2m\}$. Critical sections were generated as follows: each task T_i requires each resource ℓ_q with probability $p^{acc} \in \{0.1, 0.25, 0.5\}$, and if T_i was chosen to access ℓ_q, then $N_{i,q}$ was chosen uniformly at random from $\{1, \ldots, N^{max}\}$, where the maximum number of critical sections per job was varied across $N^{max} \in \{1, 3, 5, 7, 10\}$. The maximum critical section length $L_{i,q}$ was chosen uniformly from $[1\mu s, 25\mu s]$ (*short*), $[25\mu s, 100\mu s]$ (*medium*), or $[100\mu s, 500\mu s]$ (*long*). To ensure that each task's WCET is plausible, we ensured that $e_i \geq \sum_{\ell_q} N_{i,q} \cdot L_{i,q}$ by increasing e_i if necessary.

While these parameter choices admittedly do not represent any specific workload, they were chosen to cover a wide configuration space, including both high- and low-contention scenarios, that is likely to encompass many real-world workloads.

For each of the 1,440 possible combinations of the considered configuration parameters, we varied $n \in \{m, 12m\}$ and, for each n, generated at least 1,000 task sets. The *schedulability metric* of each protocol and analysis combination is simply the fraction of task sets that could be claimed schedulable under it.

We evaluated 12 different analyses: the seven LP-based analyses proposed in this work, respectively for the PIP, P-PCP, FMLP, FMLP$^+$, PRSB, and two cases without progress mechanism using either FIFO or priority queues (labeled "NP-FIFO" and "NP-priority," respectively); two prior analyses [12] of the PIP and P-PCP (labeled "PIP-prior" and "P-PCP-prior," respectively); and two s-oblivious analyses of the FMLP [4, 5] and OMLP [5, 8] (labeled "s-ob FMLP" and "s-ob OMLP," respectively). Finally, as an upper bound on achievable performance, we included a hypothetical case in which no blocking occurs (labeled "no blocking") based on Guan et al.'s response-time analysis for independent tasks [15]. Guan et al.'s test was also used as the underlying test in the two s-oblivious analyses.

B. Results

Due to the large number of tested configurations, we focus here on major trends. Figs. 2–5 show representative graphs that exemplify our findings; the full data set is available online.[3] To reduce clutter, RSB results have been omitted in Figs. 2 and 3 and only LP-based analysis results are shown in Figs. 4 and 5. All four graphs show results for the light utilization distribution.

Real-time locking protocols matter. Our results clearly show that real-time locking protocols are essential under non-trivial load, as evident in Figs. 2–5 by the wide gap between the no-blocking upper bound and the two NP-FIFO and NP-priority curves, which reflect uncontrolled priority inversion. In total, the use of a progress mechanism shifts schedulability closer to the no-blocking upper bound in 1,427 of the 1,440 tested configurations, whereas uncontrolled priority inversion often endangers temporal correctness even at low task counts (*i.e.*, low system load and light contention). (In the other 13 cases, schedulability is low anyway due to excessive contention.) While this result is perhaps not surprising, to the best of our knowledge, our work is the first to quantify the effect and to provide response-time bounds despite uncontrolled priority inversion.

S-aware, LP-based analysis dominates. In 1,437 out of all 1,440 tested configurations, our LP-based analyses of the PIP and the P-PCP outperform their respective conventional counterparts. (In the remaining 3 cases, the workload is infeasible under all analyses due to excessive contention.) For example, in Fig. 3, more than 60% of the task sets with $n = 20$ tasks are schedulable under the new PIP analysis, whereas less than 40% are schedulable under the prior PIP analysis. Similarly, the new P-PCP analysis outperforms the baseline in Figs. 2 and 3.

The improvement is equally apparent with regard to the OMLP and the FMLP: in 1,323 out of 1,440 tested configurations, our s-aware analysis of the FMLP claimed (often substantially) more task sets schedulable than either of the two prior s-oblivious analyses. For example, in Figs. 2 and 3, the new analysis for FMLP increases the supported task count by more than 50%, compared to the prior s-oblivious FMLP analysis.

Interestingly, in 63 scenarios with very low contention, the prior s-oblivious analyses still perform marginally better because blocking is negligible in these cases, *i.e.*, schedulability is dominated by the underlying schedulability test. The s-oblivious approach thus benefits from Guan et al.'s improved s-oblivious analysis [15], whereas our LP-based analysis builds on Bertogna and Cirinei's earlier, less-accurate response-time analysis [3].

Finally, there even exist extreme cases (not shown here) in which our LP-based analysis of *uncontrolled* priority inversion performs (slightly) better than the prior analyses of the PIP (in 329 scenarios), the P-PCP (in 385 scenarios), and the FMLP and OMLP (in 132 scenarios), *which all benefit from PI.*

PIP and FMLP perform best. Given that the proposed unified analysis provides the most accurate results for *all* considered protocols (if there is non-negligible contention), it enables a fair comparison reflecting the best available analysis. And the results are clear: in 1,427 out of 1,440 scenarios, either the PIP or the FMLP is the best-performing protocol, often by a wide margin,

[3] All results are available at http://www.mpi-sws.org/~bbb/papers/data/rtss15.

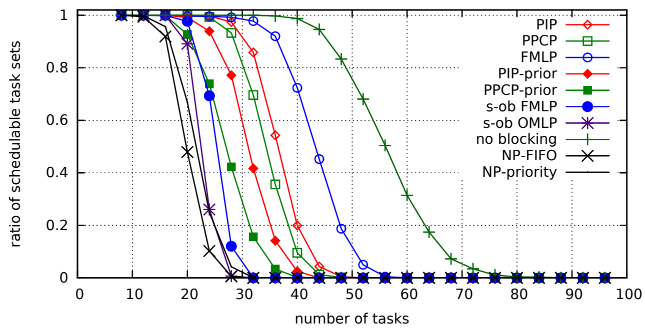


Fig. 2: Classic vs. LP-based analysis for $m = 8$, homogeneous periods, short CS lengths, $n_r = 16$, $p^{acc} = 0.5$, and $N^{max} = 5$.

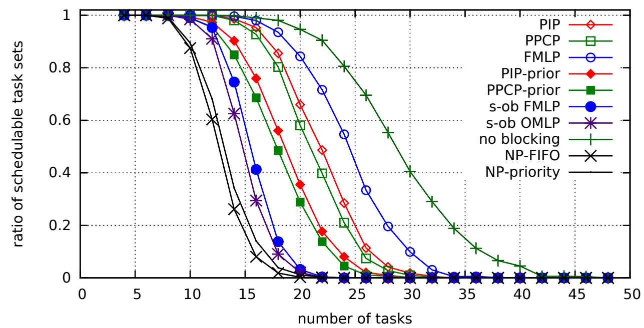


Fig. 3: Classic vs. LP-based analysis for $m = 4$, homogeneous periods, medium CS lengths, $n_r = 4$, $p^{acc} = 0.5$, and $N^{max} = 5$.

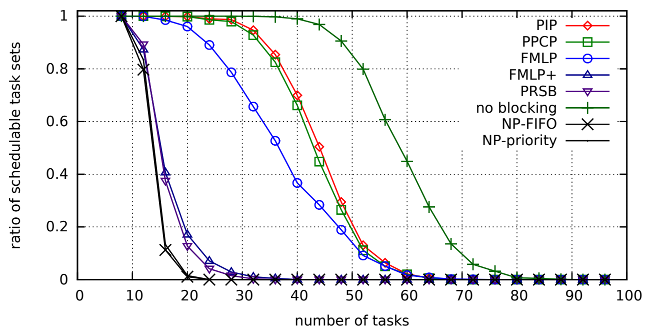


Fig. 4: LP-based protocol comparison for $m = 8$, heterogeneous periods, short CS lengths, $n_r = 8$, $p^{acc} = 0.25$, and $N^{max} = 5$.

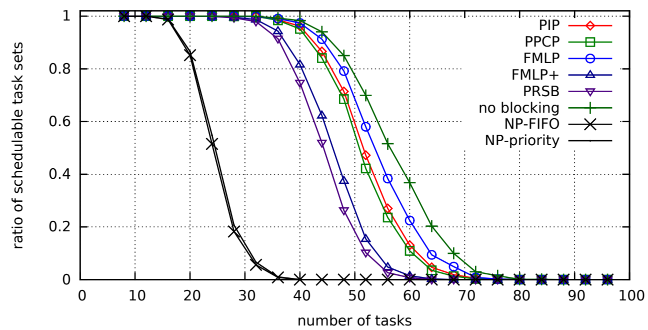


Fig. 5: LP-based protocol comparison for $m = 8$, homogeneous periods, short CS lengths, $n_r = 8$, $p^{acc} = 0.25$, and $N^{max} = 5$.

as evident in Figs. 2–5. Conversely, we found *no* configuration in which any protocol exceeds the better of the two.

Whether the PIP or the FMLP performs best depends on the type of task sets. Generally speaking, FIFO-based protocols (*e.g.*, the FMLP), which due to their starvation-free nature enable tighter bounds for lower-priority tasks, perform better than priority-based protocols (*e.g.*, the PIP) if timing constraints are mostly homogeneous, *i.e.*, if the ratio of the maximum to the minimum period is relatively small. Conversely, priority-based protocols, which ensure lower bounds for higher-priority tasks, are preferable if the ratio is large. For example, the FMLP performs best in Figs. 2, 3, and 5, where $p_i \in [10ms, 100ms]$, while the PIP exceeds in Fig. 4, where $p_i \in [1ms, 1000ms]$.

In our experiments, the FMLP performed better than the PIP more often than not: in the case of homogeneous (respectively, heterogeneous) periods, the FMLP outperformed the PIP in 642 (respectively, 245) scenarios (out of 720 each), whereas the PIP outperformed the FMLP in only 68 (respectively, 471) scenarios. However, this may be an artifact of our task set generation method and should not be understood as an absolute ranking. Overall, our data shows both protocols to be competitive in a wide range of scenarios, and to complement each other well.

P-PCP results do not justify the added complexity. Interestingly, the PIP and the FMLP, which are the oldest and the simplest considered protocols, dominate later proposals designed to improve upon them. In particular, whereas the P-PCP sometimes performs marginally better than the PIP under the prior analysis (in 157 scenarios), under the new LP-based analysis, the P-PCP *never* improves upon the PIP. In fact, the PIP actually outperforms the P-PCP in 1,195 of the 1,440 tested cases (*e.g.*, in Figs. 2–5). The P-PCP's added complexity, in terms of analysis challenges and the required runtime mechanism, are thus not justified. (To the best of our knowledge, the P-PCP has not been empirically compared to the PIP in prior work.)

PI performs better than RSB. Finally, our data also clearly shows that the FMLP$^+$'s good performance under partitioned scheduling [5–7] does not extend to G-FP scheduling: the FMLP$^+$ never outperformed the FMLP in our experiments. The primary design goal of the generalized FMLP$^+$ [7] for global and clustered scheduling was asymptotic optimality [7]; however, given that it is empirically one of the best-performing protocols under partitioned scheduling [5–7], we expected the FMLP$^+$ to perform better in our experiments. We attribute the FMLP$^+$'s relative weakness under G-FP scheduling to stalling and co-boosting interference, which arise neither under the simpler PI-based protocols PIP and FMLP (recall Table I), nor under partitioned scheduling (as each core is scheduled and analyzed individually, there is no parallelism). Intuitively speaking, the partitioned locking problem requires more "heavy-weight" solutions (the PI-based PIP and FMLP are ineffective under partitioned scheduling), which, however, do not pay off under G-FP scheduling. Given the challenges faced by the FMLP$^+$, it is no surprise that the other RSB-based protocol, the PRSB, also failed to perform well (*e.g.*, see Figs. 4 and 5).

In summary, from a pragmatic point of view, the PIP and the FMLP warrant primary consideration under G-FP scheduling.

VII. Related Work

Our basic analysis framework builds directly on Bertogna and Cirinei's classic multiprocessor response-time analysis [3] for independent tasks and adds support for self-suspensions, which is a straightforward extension since Bertogna and Cirinei's workload bound (Def. 3) trivially holds even if tasks self-suspend (as previously realized by Easwaran and Andersson [12]). While

Guan et al. [15] later significantly improved upon Bertogna and Cirinei's analysis, using a technique developed by Baruah [2], we unfortunately cannot incorporate Guan et al.'s refined bounds in our framework as their analysis is inherently s-oblivious (*i.e.*, Guan et al.'s analysis holds only if tasks never self-suspend [15]).

Liu and Anderson [17] recently proposed an s-aware analysis for global scheduling that realizes improved interference bounds for *computation tasks* (that do not suspend) while still permitting other tasks to suspend. However, their analysis is computationally involved [17] and cannot be easily incorporated into an LP; further, prior experiments [7, 17] have shown Liu and Anderson's analysis to perform well only for long self-suspension times [17], but actually worse than s-oblivious analysis in the context of semaphore protocols [7]. The problem of integrating substantially better interference bounds [2, 15] into our unified analysis framework remains an interesting challenge.

The first real-time locking protocols for uniprocessors, including the PIP, are due to Sha et al. [24]. Subsequently, much related work has targeted *partitioned* multiprocessor scheduling, including the first multiprocessor real-time locking protocols, namely the MPCP [21] and the DPCP [22] by Rajkumar et al. A recent survey and comparison of semaphore protocols for partitioned scheduling may be found in [6].

A review of all major real-time suspension-based locking protocols for G-FP scheduling is provided in §III. One recent protocol that we did not consider in detail is the LB-PCP [19], an extension of the P-PCP. Upon closer analysis, we observed that the LB-PCP does not rule out uncontrolled priority inversion in certain corner cases (*i.e.*, in the worst case, tasks are exposed to extended stalling interference as in the absence of a progress mechanism), which has since been confirmed [18].

Spin locks, a popular alternative to semaphores, have been studied under partitioned [14, 28], global [4, 11], and clustered scheduling [5]. Since tasks do not yield their processors while waiting to acquire a spin lock, the resulting loss of processor service and transitive delays must be accounted for, either implicitly by WCET inflation [4, 5, 11, 14], which resembles s-oblivious analysis, or explicitly [28], which resembles s-aware analysis. In future work, it would be beneficial to explicitly integrate busy-waiting (as in [28]) into our analysis (§V).

Several protocols for reservation-based scheduling [13] and hybrid approaches such as clustered [20] and semi-partitioned [1] scheduling have been proposed in recent years; developing an LP-based analysis similar to our framework (§V) for such hybrid approaches would be a useful extension.

Prior analyses of the PIP [12], FMLP [4, 5], and P-PCP [12] have not considered nested critical sections; we have adopted the same limitation in this work. Ward and Anderson [25, 26] recently introduced the *real-time nested locking protocol* (RNLP), a meta-protocol that adds support for fine-grained locking on top of certain underlying non-nested protocols [25, 26]. To our knowledge, of the analyzed protocols, only the FMLP$^+$ is compatible with the RNLP, *i.e.*, only the FMLP$^+$ can be integrated with the RNLP to support nested critical sections [7].

Most closely related to this paper are two prior LP-based blocking analyses [6, 28]. While our unified analysis (§V) follows roughly the same approach, it extends the technique in several novel ways. In particular, our analysis is the first to explicitly consider interference, the first to address global scheduling, which requires a much more careful definition of pi-blocking and interference (§IV-A), and the first of sufficient generality to analyze uncontrolled priority inversion (§VI-B).

VIII. Conclusion

We have summarized the current state of the art in the area of real-time semaphore protocols for G-FP scheduling (§III), identified and carefully defined six distinct types of delay (§IV), and proposed a novel, unified, LP-based, s-aware response-time analysis applicable to all protocols (§V), which we have used to conduct a large-scale, apples-to-apples comparison of all major protocols (§VI). Interestingly, we found the two oldest and simplest protocols, namely the PIP [24] and the FMLP [4], to consistently perform best across a wide range of scenarios.

Numerous interesting avenues for future work exist, as already mentioned in §V-D and §VII. Most pressingly, we seek to support nested critical sections under the PIP and the FMLP, which, however, is a fundamentally more challenging problem [27] that will require new analysis approaches.

References

[1] S. Afshar, F. Nemati, and T. Nolte, "Resource sharing under multiprocessor semi-partitioned scheduling," in *RTCSA*, 2012.

[2] S. Baruah, "Techniques for multiprocessor global schedulability analysis," in *RTSS*, 2007.

[3] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *RTSS*, 2007.

[4] A. Block, H. Leontyev, B. Brandenburg, and J. Anderson, "A flexible real-time locking protocol for multiprocessors," in *RTCSA*, 2007.

[5] B. Brandenburg, "Scheduling and locking in multiprocessor real-time operating systems," Ph.D. dissertation, The University of North Carolina at Chapel Hill, 2011.

[6] ——, "Improved analysis and evaluation of real-time semaphore protocols for P-FP scheduling," in *RTAS*, 2013.

[7] ——, "The FMLP$^+$: An asymptotically optimal real-time locking protocol for suspension-aware analysis," in *ECRTS*, 2014.

[8] B. Brandenburg and J. Anderson, "Optimality results for multiprocessor real-time locking," in *RTSS*, 2010.

[9] ——, "The OMLP family of optimal multiprocessor real-time locking protocols," *Design Automation for Embedded Systems*, vol. 17, no. 2, pp. 277–342, 2013.

[10] R. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," *Real-Time Systems*, vol. 47, no. 1, pp. 1–40, 2011.

[11] U. Devi, H. Leontyev, and J. Anderson, "Efficient synchronization under global EDF scheduling on multiprocessors," in *ECRTS*, 2006.

[12] A. Easwaran and B. Andersson, "Resource sharing in global fixed-priority preemptive multiprocessor scheduling," in *RTSS*, 2009.

[13] D. Faggioli, G. Lipari, and T. Cucinotta, "The multiprocessor bandwidth inheritance protocol," in *ECRTS*, 2010.

[14] P. Gai, G. Lipari, and M. Di Natale, "Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip," in *RTSS*, 2001.

[15] N. Guan, M. Stigge, W. Yi, and G. Yu, "New response time bounds for fixed priority multiprocessor scheduling," in *RTSS*, 2009.

[16] K. Lakshmanan, D. De Niz, and R. Rajkumar, "Coordinated task scheduling, allocation and synchronization on multiprocessors," in *RTSS*, 2009.

[17] C. Liu and J. Anderson, "Suspension-aware analysis for hard real-time multiprocessor scheduling," in *ECRTS*, 2013.

[18] G. Macariu, personal communication, April, 2015.

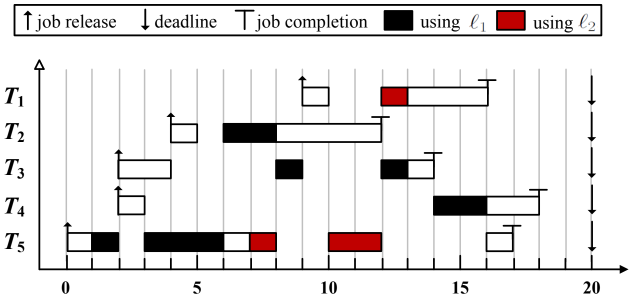
[19] G. Macariu and V. Cretu, "Limited blocking resource sharing for global multiprocessor scheduling," in *ECRTS*, 2011.

[20] F. Nemati, M. Behnam, and T. Nolte, "Independently-developed real-time systems on multi-cores with shared resources," in *ECRTS*, 2011.

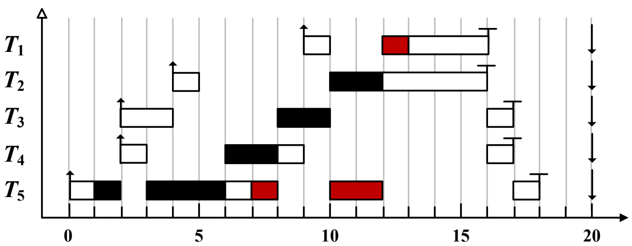
[21] R. Rajkumar, "Real-time synchronization protocols for shared memory multiprocessors," in *ICDCS*, 1990.

[22] R. Rajkumar, L. Sha, and J. Lehoczky, "Real-time synchronization protocols for multiprocessors," in *RTSS*, 1988.

[23] "SchedCAT: Schedulability test collection and toolkit," http://www.mpi-sws.org/∼bbb/projects/schedcat.



(a) Example PIP schedule ($m = 2$)



(b) Example FMLP schedule ($m = 2$)

Fig. 6: Example PIP and FMLP schedules.

[24] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority inheritance protocols: an approach to real-time synchronization," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1175–1185, 1990.

[25] B. Ward and J. Anderson, "Supporting nested locking in multiprocessor real-time systems," in *ECRTS*, 2012.

[26] ——, "Fine-grained multiprocessor real-time locking with improved blocking," in *RTNS*, 2013.

[27] A. Wieder and B. Brandenburg, "On the complexity of worst-case blocking analysis of nested critical sections," in *RTSS*, 2014.

[28] ——, "On spin locks in AUTOSAR: blocking analysis of FIFO, unordered, and priority-ordered spin locks," in *RTSS*, 2013.

APPENDIX

A. PIP Example Schedule

Fig. 6(a) depicts an example schedule under the PIP. T_3 and T_4 are released at time $t = 2$ and prevent T_5 from executing, thus both T_3 and T_4 cause T_5 to incur regular interference at time $t = 2$. Then T_4, T_3, and T_2 request ℓ_1 and are directly delayed at times $t = 3$, $t = 4$, and $t = 5$, respectively.

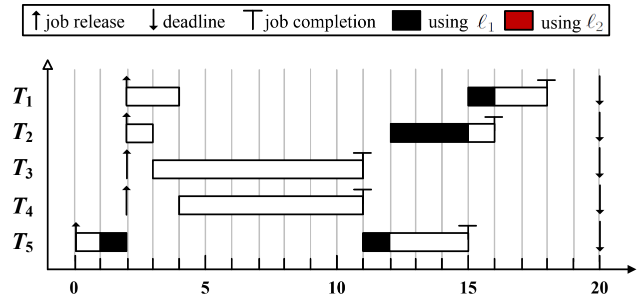
During $[3, 6)$, while T_4 waits for T_5 to release ℓ_1 and T_5 is scheduled, T_5 causes T_4 to incur direct pi-blocking. Meanwhile, two higher-base-priority tasks T_3 and T_2 are scheduled during $[3, 4)$ and $[4, 5)$, respectively; hence, T_4 is also considered to incur regular interference due to T_3 and T_2 during $[3, 4)$ and $[4, 5)$, respectively.

Since priority queues are used under the PIP, T_2, which is the highest-base-priority waiting for ℓ_1, is granted ℓ_1 at time $t = 6$ and scheduled. During $[6, 8)$, while T_3 and T_4 wait for T_2 to release ℓ_1 and while there are fewer than $m = 2$ higher-base-priority tasks scheduled, T_2 causes T_3 and T_4 to incur direct pi-blocking.

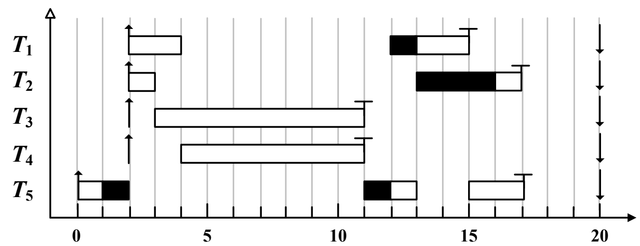
During $[8, 10)$, T_4 is suspended and waits for T_3 to release ℓ_1. However, it does not incur pi-blocking during this interval since all scheduled tasks (*i.e.*, T_2 and T_3) have higher base priorities. Hence, T_4 incurs regular interference during $[8, 10)$.

At time $t = 10$, T_1 requests ℓ_2, which is held by T_5, and thus T_1 suspends. Consequently, T_5 inherits an effective priority of T_1 (*i.e.*, $\pi_5(10) = 1$) and is scheduled until it releases ℓ_1 at time $t = 12$.

Since T_5 is holding ℓ_2 and scheduled during $[10, 12)$, T_5 causes T_1 to incur direct pi-blocking. During the same interval, while T_3 is ready and not scheduled, T_5 causes T_3 to incur preemption pi-blocking. Meanwhile, T_4 waits for T_3 to release ℓ_1. Since T_3 is not scheduled while T_5 is scheduled and holding a resource, T_5 causes T_4 to incur indirect pi-blocking.



(a) Example NP-FIFO schedule ($m = 2$)



(b) Example NP-priority schedule ($m = 2$)

Fig. 7: Example schedules of demonstrating uncontrolled priority inversions in the absence of a real-time locking protocol.

B. FMLP Example Schedule

Fig. 6(b) shows an example FMLP schedule for the same job-arrival sequence as discussed in the preceding appendix. Since FIFO queues are used in the FMLP and T_4 requests ℓ_1 earlier than T_2 and T_3, at time $t = 6$, T_4 is granted ℓ_1 (instead of T_2 in the PIP example). Then T_3 and T_2 are granted ℓ_1 at times $t = 8$ and $t = 10$, respectively. Consequently, T_4 incurs direct pi-blocking due to T_5 during $[3, 6)$, T_3 incurs direct pi-blocking due to T_5 during $[4, 6)$, and due to T_4 during $[6, 8)$, and T_2 incurs direct pi-blocking due to T_5, T_4, and T_3 during $[5, 6)$, $[6, 8)$, $[8, 10)$, respectively. During $[10, 12)$, while T_1 waits for T_5 to release ℓ_2 and T_5 is scheduled, T_5 causes T_1 to incur direct pi-blocking. During the same interval, as T_3 and T_4 are ready and not scheduled, T_5 causes both T_3 and T_4 to incur preemption pi-blocking.

C. Example Schedules with Uncontrolled Priority Inversion

Figs. 7(a) and 7(b) depict examples of uncontrolled priority inversion in the absence of a real-time locking protocol. At time $t = 2$, $T_1, \ldots, T_4$ are released and the two highest-priority tasks T_1 and T_2 are scheduled, thus preventing T_5 from being scheduled. Then T_2 and T_1 are suspended while waiting for T_5 to release ℓ_1 at times $t = 3$ and $t = 4$, respectively, thereby allowing T_3 and T_4 to be scheduled. Since no real-time locking protocol is used, the effective priorities of $T_1, \ldots, T_5$ are equal to their respective base priorities. Consequently, T_5 cannot be

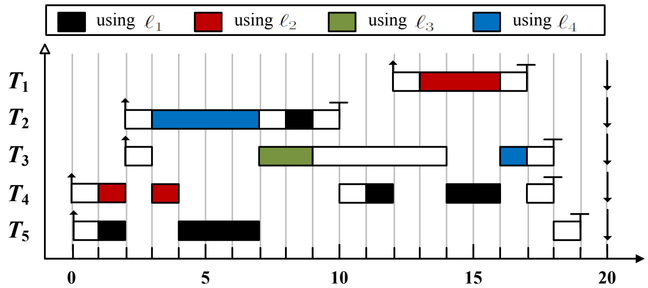


Fig. 8: Example P-PCP schedule ($m = 2$).

scheduled until T_3 and T_4 finish at time $t = 11$. Thus, T_2 incurs stalling interference due to T_3 during $[3, 11)$, and due to T_4 during $[4, 11)$. T_1 incurs stalling interference due to both T_3 and T_4 during $[4, 11)$. During $[11, 12)$, while T_5 is scheduled and holding ℓ_1, which T_1 and T_2 are waiting for, T_5 causes T_1 and T_2 to incur direct pi-blocking. When FIFO queues are used (*i.e.*, in Fig. 7(a)), T_2 is granted ℓ_1 at time $t = 12$ and is scheduled during $[12, 15)$. Since T_1 is waiting for ℓ_1 during $[12, 15)$, T_2 causes T_1 to incur direct pi-blocking. In contrast, when priority queues are used (*i.e.*, in Fig. 7(b)), T_1 is granted ℓ_1 at time $t = 12$ and releases ℓ_1 at time $t = 13$, at which point T_2 is granted ℓ_1.

D. P-PCP Example Schedule

Finally, we show a P-PCP example schedule in Fig. 8, where $m = 2$ and $n = 5$. At time $t = 2$, T_2 and T_3 are released and prevent T_4 and T_5 from executing. At time $t = 3$, T_2 requests ℓ_4 and T_3 requests ℓ_3. Since T_4 and T_5 have locked resources at time $t = 3$, $|LPR(2,3)| + |HPR(2,3)| = |LPR(3,3)| + |HPR(2,3)| = 2$. Under the (m, n)-configured P-PCP, $\alpha_2 = n = 5$, and since $|LPR(2,3)| + |HPR(2,3)| < \alpha_2 = n$ and ℓ_4 is not held by any task at time $t = 3$, T_2 is granted ℓ_4. However, because $\alpha_3 = m = 2$ and $|LPR(3,3)| + |HPR(3,3)| = \alpha_3 = m$, T_3 is denied access to all resources at time $t = 3$. Since T_3 is suspended at time $t = 3$, T_4, which is the task with the shortest maximum critical section length in $LPR(3, 3)$, inherits the effective priority of T_3 and is consequently scheduled. At time $t = 4$, when T_4 releases ℓ_2, since $|LPR(3,3)| + |HPR(3,3)| = 1 + 1 = 2 = m$, T_3 is still denied access to all resources and suspended, and instead T_5 is scheduled. Therefore, T_3 incurs stalling interference due to T_4 during $[3, 4)$, and due to T_5 during $[4, 7)$, respectively. Similarly, T_3 also incurs stalling interference due to T_4 during $[14, 16)$.

E. RSB-specific Constraints

The key property of the RSB is that a job has an effective priority higher than its base priority only **(i)** if it is priority-boosted while holding a resource, or **(ii)** if it is co-boosted and not holding any resource. We derive several constraints from this property. In preparation, we establish the following lemma.

Lemma 12: Under RSB, J_i incurs co-boosting or stalling interference at time t only if **(i)** a higher-base-priority job J_h ($T_h \in \tau^H$) is priority-boosted and J_h does not cause J_i to incur direct pi-blocking at time t, or **(ii)** J_i incurs indirect or preemption pi-blocking at time t.

Proof: By definition, when J_i incurs co-boosting or stalling interference, it is either *(a)* ready and not scheduled at time t, or *(b)* suspended and not incurring direct pi-blocking at time t.

If J_i incurs co-boosting interference at time t, then there exists a job J_x with $\pi_x(t) < i$ (and $x > i$) that is not scheduled and not holding a resource. Under RSB, this is possible only if J_x is co-boosted at time t. Then there exists another priority-boosted job J_a (with $a > x$) that is scheduled and holding a resource at time t. By definition, J_a causes J_i to incur preemption pi-blocking (in case (a)) or indirect pi-blocking (in case (b)).

If J_i incurs stalling interference at time t, then there exists a job J_x with $\pi_x(t) \geq i$ (where $x > i$) that is scheduled at time t. Under RSB, the effective priority of any job is either boosted (*i.e.*, $\pi_x(t) = 0$) or equal to its base priority. Since $\pi_x(t) \geq i$, J_x is not priority-boosted, and thus $\pi_x(t) > i$ at time t. Clearly, this is possible only if J_i is suspended (*i.e.*, in case (b)).

Thus J_i is suspended and waiting for some job to release a resource at time t. Under RSB, if there exists a resource-holding job, then there exists a priority-boosted job J_c ($T_c \in \tau^i$) that is scheduled at time t. Thus, if $T_c \in \tau^H$, then case (i) holds; otherwise, if $T_c \in \tau^L$, then J_i incurs preemption pi-blocking (in case (a)) or indirect pi-blocking (in case (b)) at time t and case (ii) holds. ■

Based on Lemma 12, we impose the following constraint to limit the total cumulative co-boosting and stalling interference that J_i incurs due to each other task.

Constraint 16: In any G-FP schedule of τ under RSB:

$$\forall T_x \in \tau^L : I_x^C + I_x^S \leq \left(\sum_{T_h \in \tau^H} \left(\sum_{\forall \ell_q} N_{h,q}^i \cdot L_{h,q} \right) - B_h^D \right) + \sum_{T_l \in \tau^L \setminus T_x} B_l^I + B_l^P.$$

Proof: By Lemma 12, whenever J_i incurs co-boosting or stalling interference due to a job of T_x, either a higher-base-priority job is priority-boosted (and not causing direct pi-blocking) or a lower-priority job is causing preemption or indirect pi-blocking. We consider the two cases separately.

Higher-base-priority jobs: While J_i is pending, higher-base-priority tasks hold resources for at most $\sum_{T_h \in \tau^H} \sum_{\forall \ell_q} N_{h,q}^i \cdot L_{h,q}$ time units, and cause J_i to incur direct pi-blocking for $\sum_{T_h \in \tau^H} B_h^D$ time units. Thus higher-base-priority tasks are priority-boosted while not causing J_i to incur direct pi-blocking for at most $\sum_{T_h \in \tau^H} (\sum_{\forall \ell_q} N_{h,q}^i \cdot L_{h,q}) - B_h^D$ time units.

Lower-base-priority jobs: by definition of the variables B_l^I and B_l^P, J_i incurs indirect or preemption pi-blocking due to lower-base-priority jobs (except J_x) for at most $\sum_{T_l \in \tau^L \setminus T_x} B_l^I + B_l^P$ time units. T_x is excluded because, by Lemma 2, while J_x causes J_i to incur co-boosting or stalling interference, it will not cause J_i to incur any other delay.

The right-hand side of the constraint is the sum of the two individual bounds. ■

Furthermore, when J_i incurs co-boosting or stalling interference, there are at most $m - 1$ jobs that cause J_i to incur co-boosting and stalling interference. Based on this observation, in addition to Constraint 16, we constrain the total co-boosting and stalling interference that J_i incurs with the following constraint.

Constraint 17: In any G-FP schedule of τ under RSB:

$$\sum_{T_x \in \tau^L} I_x^C + I_x^S \leq (m - 1) \times$$

$$\left(\left(\sum_{T_h\in\tau^H}\left(\sum_{\forall\ell_q} N^i_{h,q}\cdot L_{h,q}\right) - B^D_h\right) + \sum_{T_l\in\tau^L} B^I_l + B^P_l\right).$$

Proof: By Lemma 12, if J_i incurs co-boosting or stalling interference at some time t, then there exists a job J_h ($h < i$) that is priority-boosted but does not cause J_i to incur direct pi-blocking at time t, or there exists a job J_l ($l > x$) that causes J_i to incur indirect or preemption pi-blocking at time t. By Lemma 2, at any point in time a job can cause another job to incur at most one type of delay. Thus, neither J_h nor J_l will cause J_i to incur co-boosting or stalling interference at time t. Hence at most $m-1$ jobs cause J_i to incur co-boosting or stalling interference at time t. By definition, tasks in τ^H are priority boosted while not causing J_i to incur direct pi-blocking for at most $\sum_{T_h\in\tau^H}(\sum_{\forall\ell_q} N^i_{h,q}\cdot L_{h,q}) - B^D_h$ time units; whereas tasks in τ^L cause J_i to incur indirect and preemption pi-blocking for $\sum_{T_l\in\tau^L} B^I_l + B^P_l$ time units. Thus, J_i incurs at most

$$(m-1)\times\left(\left(\sum_{T_h\in\tau^H}\left(\sum_{\forall\ell_q} N^i_{h,q}\cdot L_{h,q}\right) - B^D_h\right) + \sum_{T_l\in\tau^L} B^I_l + B^P_l\right)$$

total co-boosting and stalling interference. ■

Under RSB, a job J_x causes J_i to incur co-boosting interference at some time t only if J_x is co-boosted at time t, which requires that some job J_a with base priority lower than J_x ($a > x$) is priority-boosted at time t. Consequently, J_a by definition causes J_i to incur indirect or preemption pi-blocking at time t. In other words, a job causes J_i to incur co-boosting interference only if some other job causes J_i to incur indirect or preemption pi-blocking. Based on this observation, we impose the following constraint to limit co-boosting interference.

Constraint 18: In any schedule of τ under RSB:

$$\forall T_x\in\tau^L : I^C_x \le \sum_{a>x} B^I_a + B^P_a.$$

Proof: Suppose not. Then there is a time t at which J_x causes J_i to incur co-boosting interference while no job causes J_i to incur indirect or preemption pi-blocking at time t. Since J_x causes J_i to incur co-boosting interference at time t, J_x is co-boosted, and J_i is pending, not scheduled and does not incur direct pi-blocking at time t. Since J_x is co-boosted at time t, there is a priority-boosted job J_a ($a > x$) that is holding a resource at time t. By definition, J_a causes J_i to incur preemption pi-blocking if J_i is ready and not scheduled at time t, or J_a causes J_i to incur indirect pi-blocking if J_i is suspended and not incurring direct pi-blocking at time t. Contradiction. ■

In addition to Constraint 18, we impose the following constraint to limit the total cumulative co-boosting that J_i incurred due to all lower-base-priority tasks.

Constraint 19: In any schedule of τ under RSB:

$$\sum_{T_x\in\tau^L} I^C_x \le (m-1)\times\sum_{T_a\in\tau^L} B^I_a + B^P_a.$$

Proof: Analogous to the proof for Constraint 18, J_i incurs co-boosting interference only if it incurs indirect or preemption pi-blocking. By Lemma 2, each other task causes J_i to incur at most one type of delay at any time. Thus at most $m-1$ tasks can cause J_i to incur co-boosting interference at any time. Since J_i incurs $\sum_{T_a\in\tau^L} B^I_a + B^P_a$ indirect and preemption pi-blocking, it incurs at most $(m-1)\times\sum_{T_a\in\tau^L} B^P_a + B^I_a$ co-boosting interference. ■

Analogous to Constraint 15, we rule out stalling interference that J_i incurred due to another job J_x in case none of the jobs with base priority lower than J_x accesses a resource that will be requested by J_i.

Constraint 20: In any schedule of τ under RSB:

$$\forall T_x\in\tau^L \text{ s.th.} \sum_{l>x}\sum_{\substack{\forall\ell_q \text{ s.th.}\\ N_{i,q}>0}} N_{l,q} = 0 : \quad I^S_x = 0.$$

Proof: $\sum_{l>x}\sum_{\forall\ell_q \text{ s.th.} N_{i,q}>0} N_{l,q} = 0$ implies that no job with base priority lower than J_x accesses a resource that will be requested by J_i. Then, under RSB, whenever J_i is suspended, it is waiting for a resource either held by J_x or some job J_a ($a < x$).

If J_i is waiting for a resource held by J_x at some time t, then J_x causes J_i to incur direct pi-blocking (*i.e.*, J_x is scheduled at time t) or does not cause J_i to incur any delay (*i.e.*, J_x is not scheduled at time t). By Lemma 1, J_x will not cause J_i to incur stalling interference if J_x causes J_i to incur direct pi-blocking. Thus J_x will not cause stalling interference when J_i is suspended and waiting for a resource held by J_x.

If J_i is waiting for a resource held by a job J_a (where $a < x$), then it follows that J_a is ready. Hence, if J_a is scheduled at time t, then J_a causes J_i to incur direct pi-blocking. In that case, J_i does not incur stalling interference according to Lemma 1.

If J_a is not scheduled at time t, then it is impossible for J_x to be scheduled at time t according to G-FP scheduling. Thus, J_x cannot cause J_i to incur stalling interference when J_i is suspended and waiting for a resource held by J_a.

Finally, if J_i is ready and not scheduled, then J_x will not be scheduled at time t under G-FP scheduling. Thus J_x cannot cause J_i to incur stalling interference when J_i is ready and not scheduled. The claim follows. ■

In addition, we impose a constraint to rule out preemption pi-blocking due to the m highest-base-priority tasks.

Constraint 21: In any G-FP schedule of τ under RSB:

$$\forall T_x \text{ s.t. } i < x \le m : \quad \sum_{\forall\ell_q}\sum_{v=1}^{N^i_{x,q}} X^P_{x,q,v} = 0.$$

Proof: Consider a job J_x that is priority-boosted. To cause J_i to incur preemption pi-blocking at some time t, by definition, J_x must be scheduled and holding a resource while J_i is ready and not scheduled and $i < x$.

Under RSB, only jobs with higher base priority are included in a job's co-boosting set. If $x \le m$, then there are at most $m-1$ jobs in J_x's co-boosting set at any time.

If J_i is included in J_x's co-boosting set at time t, then J_i is obviously scheduled and J_x does not cause J_i to incur preemption pi-blocking.

If J_i is not included in J_x's co-boosting set at time t, then there are at most $m-2$ jobs with effective higher priorities than J_i because $x \le m$ and $i < x$ (and since each task has at most one ready job at any time). Thus J_i is scheduled if ready at time t even if not included in J_x's co-boosting set, and thus J_x does not cause J_i to incur preemption pi-blocking at any time t. ■

Next, we impose a constraint to limit indirect pi-blocking from the fact that J_i incurs indirect pi-blocking only if it is directly delayed by some other job (*i.e.*, J_i incurs indirect pi-blocking due to J_x only if there exists a job other than J_x that is holding a resource requested by J_i).

Constraint 22: In any G-FP schedule of τ under RSB:

$$\forall T_x \in \tau^L : \sum_{\forall \ell_q} \sum_{v=1}^{N^i_{x,q}} X^I_{x,q,v} \leq \sum_{T_y \in \tau^i \setminus T_x} \sum_{\forall \ell_u : N_{i,u} > 0} N^i_{y,u}.$$

Proof: By the definition of indirect pi-blocking, if J_i incurs indirect pi-blocking due to a job J_x at time t, then $x > i > \pi_x(t)$ and J_i is waiting for some resource held by another job J_a that is not scheduled at time t while J_x is scheduled and holding a resource at time t (due to being priority-boosted).

Under RSB, resource-holding jobs are priority-boosted in order of increasing request-segment start times. While a job J_a is holding a resource that J_i is waiting for, since tasks are sequential, no T_x can be priority-boosted due to a second request segment before J_a has been priority-boosted (and hence before J_a has released the resource that J_i is waiting for).

Therefore, the number of times that any task (other than T_x) holds a resource that J_i may be waiting for upper-bounds the number of requests executed by jobs of T_x that cause J_i to incur indirect pi-blocking. ■

In the following, we impose protocol-specific constraints for the FMLP$^+$, the PRSB, and the P-PCP.

F. Constraints for the FMLP$^+$

Constraint 23: In any G-FP schedule of τ under the FMLP$^+$:

$$\forall T_x \in \tau^i : \sum_{\forall \ell_q} \sum_{v=1}^{N^i_{x,q}} X^D_{x,q,v} + X^I_{x,q,v} + X^P_{x,q,v} \leq 1 + 2 \sum_{\forall \ell_q} N_{i,q}.$$

Proof: The proof follows directly from the key property of the FMLP$^+$ that ensures asymptotic optimality under s-aware analysis: each other task can cause J_i to incur pi-blocking at most once per segment under the FMLP$^+$ [7]. ■

Since indirect pi-blocking can arise only if there is direct contention for shared resources, we establish the following bound on overall direct and indirect pi-blocking.

Constraint 24: In any G-FP schedule of τ under the FMLP$^+$:

$$\forall T_x \in \tau^i : \sum_{\forall \ell_q} \sum_{v=1}^{N^i_{x,q}} X^D_{x,q,v} + X^I_{x,q,v} \leq \sum_{\forall \ell_u} \min \left(N_{i,u}, \sum_{T_y \in \tau^i} N^i_{y,u} \right).$$

Proof: Under the FMLP$^+$, J_i suspends due to a request for a resource ℓ_u only if both J_i and some other job have both issued conflicting requests for ℓ_u. J_i issues at most $N_{u,i}$ requests for ℓ_u, and jobs of other tasks issue at most $\sum_{T_y \in \tau^i} N^i_{y,u}$ requests for ℓ_u while J_i is pending. Hence, J_i suspends due to requests for ℓ_u at most $\min\left(N_{i,u}, \sum_{T_y \in \tau^i} N^i_{y,u}\right)$ times.

Each time that J_i suspends due to a resource conflict, each other task can cause direct or indirect pi-blocking with at most one request under the FMLP$^+$ because all contention is resolved in FIFO order [7]. Hence, no task causes J_i to incur direct or indirect pi-blocking with more than $\sum_{\forall \ell_u} \min\left(N_{i,u}, \sum_{T_y \in \tau^i} N^i_{y,u}\right)$ requests in total. ■

In addition to Constraint 24, the following constraint is imposed to constrain only indirect pi-blocking, which allows excluding T_x from the sum on the right-hand side of the constraint.

Constraint 25: In any G-FP schedule of τ under the FMLP$^+$:

$$\forall T_x \in \tau^L : \sum_{\forall \ell_q} \sum_{v=1}^{N^i_{x,q}} X^I_{x,q,v} \leq \sum_{\forall \ell_u} \min \left(N_{i,u}, \sum_{T_y \in \tau^i \setminus T_x} N^i_{y,u} \right).$$

Proof: Analogous to the proof for Constraint 24, J_i requests ℓ_u at most $N_{i,u}$ times, and tasks other than T_x can issue at most $\sum_{T_y \in \tau^i \setminus T_x} N^i_{y,u}$ requests to ℓ_u. Thus T_x can cause J_i to incur indirect pi-blocking while J_i is waiting to acquire ℓ_u at most $\sum_{T_y \in \tau^i \setminus T_x} N^i_{y,u}$ times. Hence, for each resource ℓ_u, T_x causes J_i to incur indirect pi-blocking at most $\min\left(N_{i,u}, \sum_{T_y \in \tau^i \setminus T_x} N^i_{y,u}\right)$ times. Then across all resources, T_x causes J_i to incur indirect pi-blocking at most $\sum_{\forall \ell_u} \min\left(N_{i,u}, \sum_{T_y \in \tau^i \setminus T_x} N^i_{y,u}\right)$ times. ■

Constraint 25 can reduce pessimism if a single task T_x is responsible for most of the direct pi-blocking.

Next, we impose a protocol-specific constraint for the PRSB.

G. A Constraint for the PRSB

Next, we derive a constraint to limit indirect pi-blocking under the PRSB based on the combination of RSB and priority queues. In preparation, we establish the following lemma.

Lemma 13: Under the PRSB, each time a job J_i requests a resource ℓ_q, it is delayed by at most $ND_{i,q}$ requests.

$$ND_{i,q} = \min \left(1, \sum_{T_l \in \tau^L} N_{l,q} \right) + \sum_{T_h \in \tau^H} \left\lceil \frac{R_h + W_{i,q}}{p_h} \right\rceil \cdot N_{h,q}.$$

Proof: Since priority queues are used under the PRSB, each time J_i issues a request to ℓ_q, it is delayed by at most one lower-base-priority request (if any), *i.e.*, by at most $\min(1, \sum_{T_l \in \tau^L} N_{l,q})$ lower-base-priority requests. Further, by Lemma 11, J_i waits for ℓ_q for at most $W_{i,q}$ time units, thus at most $\sum_{T_h \in \tau^H} \left\lceil \frac{R_h + W_{i,q}}{p_h} \right\rceil$ higher-base-priority jobs exist while J_i is waiting for ℓ_q. Since each higher-base-priority job J_h ($T_h \in \tau^H$) issues at most $N_{h,q}$ requests to ℓ_q, each time J_i requests ℓ_q it is delayed by at most $\sum_{T_h \in \tau^H} \left\lceil \frac{R_h + W_{i,q}}{p_h} \right\rceil \cdot N_{h,q}$ higher-base-priority requests. The proof follows. ■

Based on Lemma 13, we derive the following constraint.

Constraint 26: In any G-FP schedule of τ under the PRSB:

$$\forall T_x \in \tau^L : \sum_{\forall \ell_q} \sum_{v=1}^{N^i_{x,q}} X^I_{x,q,v} \leq \sum_{\forall \ell_u} ND_{i,u} \cdot N_{i,u}.$$

Proof: By contradiction. Suppose there is a schedule such that $\sum_{\forall \ell_q} \sum_{v=1}^{N^i_{x,q}} X^I_{x,q,v} > \sum_{\forall \ell_u} ND_{i,u} \cdot N_{i,u}$ for some $T_x \in \tau^L$.

It follows from Lemma 13 that J_i is delayed by at most $\sum_{\forall \ell_u} ND_{i,u} \cdot N_{i,u}$ directly conflicting requests. From the definition of indirect pi-blocking, we observe that J_i is waiting to acquire a resource when it incurs indirect pi-blocking. Thus, by the pigeon-hole principle, there exists a task T_x that causes J_i to incur indirect pi-blocking due to at least two requests while J_i is waiting to acquire a resource.

Let ℓ_u denote the resource that J_i is waiting for when it is delayed twice by J_x, and let J_a (where $T_a \in \tau^i \setminus T_x$) denote the job holding ℓ_u.

By definition, at any point in time t at which a job J_x causes J_i to incur indirect pi-blocking, J_x is scheduled and holding a resource, and $\pi_x(t) < i$.

Under RSB, $\pi_x(t) < i$ holds only if J_x is priority boosted at time t (recall that $T_x \in \tau^L$). Thus, while J_a is holding the resource ℓ_u (which J_i seeks to acquire), one or more jobs of T_x are priority-boosted due to more than one request of T_x. However, this is impossible because, under RSB, only one resource-holding job is priority-boosted at a time, and resource-holding jobs are priority-boosted in increasing order of segment start times. Contradiction. ■

H. Constraints for the P-PCP

Next, we impose constraints for the P-PCP. To reiterate, we consider the (m, n)-configuration that was suggested in the original paper [12]. In preparation, we establish the following lemmas. We begin by establishing a necessary condition for stalling interference under the (m, n)-configured P-PCP.

Lemma 14: Under the (m, n)-configured P-PCP, a job J_i incurs stalling interference at time t only if **(i)** J_i is suspended and waiting for a resource that is not held by any job at time t, and **(ii)** at least one job in $LPR(i, t)$ is scheduled at time t.

Proof: By contradiction. Suppose J_i incurs stalling interference due to some job J_x ($T_x \in \tau^L$) at a time t while either condition is violated. By the definition of stalling interference, at time t, J_i is pending, not scheduled, and does not incur direct pi-blocking. Furthermore, it is implied that J_x is scheduled at time t, where $x > i$ and $\pi_x(t) \geq i$. Two cases are considered.

Case 1: Suppose condition (i) does not hold, *i.e.*, at time t, J_i is either ready or suspended and waiting for a resource that is held by another job J_a ($J_a \neq J_x$).

First, suppose that J_i is ready. By Lemma 6, effective priorities of ready jobs are unique. Thus, under G-FP scheduling, if J_i is ready while a lower-priority job J_x ($\pi_x(t) \geq i$) is scheduled at time t, then J_i is scheduled at time t, too. However, this is impossible because, by definition, J_i is not scheduled when it incurs stalling interference. Hence J_i is not ready.

If J_i is suspended and waiting for a resource that is held by another job J_a at time t, then J_a is ready at time t and $\pi_a(t) \leq i$ due to PI. Since both $\pi_x(t) \geq i$ and $i \geq \pi_a(t)$, and since effective priorities of ready jobs are unique (Lemma 6), we observe that $\pi_x(t) > \pi_a(t)$. Since J_i does not incur direct pi-blocking at time t, J_a is not scheduled at time t. Then, according to G-FP scheduling, J_x will also not be scheduled at time t (since $\pi_x(t) > \pi_a(t)$). However, this is impossible because, by definition, J_x must be scheduled to cause stalling interference.

Therefore, it follows that Condition (i) holds. Contradiction.

Case 2: Condition (ii) does not hold, *i.e.*, no job in $LPR(i, t)$ is scheduled at time t. Since Condition (i) holds, we have $|HPR(i, t)| + |LPR(i, t)| \geq m$ according to the rules of the P-PCP.

Let J_b (with $b > i$) denote the job in $LPR(i, t)$ with the shortest maximum critical section length. According to the rules of the P-PCP, J_b inherits J_i's base priority at time t: $\pi_b(t) \leq i$.

Since the effective priorities of ready jobs are unique (Lemma 6), and since $\pi_x(t) \geq i$, we have $\pi_b(t) < \pi_x(t)$. Thus, under G-FP scheduling, if J_b is not scheduled at time t, then J_x is also not scheduled at time t. However, by definition, J_x must be scheduled to cause stalling interference.

It follows that Condition (ii) holds. Contradiction. ■

Next, we establish that no job is added to $LPR(i, t)$ while J_i incurs stalling interference.

Lemma 15: Under the (m, n)-configured P-PCP, if a job J_i (with $i > m$) is ineligible to lock a resource throughout an interval $[t_a, t_b)$, then no job is added to $LPR(i, t)$ during $t \in [t_a, t_b)$.

Proof: Suppose not. Then there exists a job J_x ($x > i$) that is added to $LPR(i, t)$ at some time $t \in (t_a, t_b)$, *i.e.*, some lower-base-priority job J_x acquires a lock while J_i is still ineligible to lock a resource.

Since an (m, n)-configuration is assumed and $x > i > m$, $\alpha_x = \alpha_i = m$. Thus, as J_x locks a resource at time t, we observe that $|HPR(x, t)| + |LPR(x, t)| < m$.

By definition, $HPR(x, t) \bigcup LPR(x, t)$ contains all jobs that are holding resources with priority ceilings higher than x at time t. Similarly, $HPR(i, t) \bigcup LPR(i, t)$ contains all jobs that are holding resources with priority ceilings higher than i at time t. Since, $i < x$, it follows that $HPR(i, t) \bigcup LPR(i, t) \subseteq HPR(x, t) \bigcup LPR(x, t)$.

Thus $|HPR(i, t)| + |LPR(i, t)| \leq |HPR(x, t)| + |LPR(x, t)| < m$. Therefore, if J_x is allowed to lock a resource at time t, then J_i is also eligible to acquire a lock at time t.

However, J_i continuously incurs stalling interference during $[t_a, t_b)$ and hence is ineligible to acquire a lock at time t. Contradiction. ■

Based on Lemma 15, it follows that $LPR(i, t)$ contains at most m distinct jobs while a job J_i incurs stalling interference.

Lemma 16: Under the (m, n)-configured P-PCP, if a job J_i (with $i > m$) is ineligible to lock a resource throughout an interval $[t_a, t_b)$, then $\left|\bigcup_{t \in [t_a, t_b]} LPR(i, t)\right| \leq m$.

Proof: Easwaran and Andersson showed that $|LPR(x, t)| \leq \alpha_x$ at any time t and for each task $T_x \in \tau$ [12]. Thus, under the (m, n)-configured P-PCP, for $x > m$, we have $|LPR(i, t)| \leq m$ at all times. Hence, $|LPR(i, t_a)| \leq m$.

By Lemma 15, no job is added to $LPR(i, t)$ at any point in time in $[t_a, t_b)$. Thus there are a total of at most m distinct jobs in $LPR(i, t)$ during $t \in [t_a, t_b)$, *i.e.*, $\left|\bigcup_{t \in [t_a, t_b]} LPR(i, t)\right| \leq m$. ■

Based on Lemmas 14 and 16, we are now ready to state a constraint that limits stalling interference due to lower-base-priority tasks. Some additional notation is required.

Recall that sr^i_x denotes the set of resources used by task T_x that have priority ceilings higher than the base priority of T_i. In the following, we let

$$LL_x^q = \max\left\{L_{x,u} \;\middle|\; \ell_u \in sr_x^i \wedge \ell_u \neq \ell_q\right\}$$

denote the maximum request length with respect to resources in sr_x^i (other than ℓ_q), *i.e.*, with respect to resources in $\{\ell_u | \ell_u \neq \ell_q \wedge \ell_u \in sr_x^i\}$. We further let $\Phi_{i,q} = \left\{LL_x^q \;\middle|\; T_x \in \tau^L\right\}$ denote a set of such bounds for all lower-base-priority tasks. Finally, we let $\varphi_{i,q}^c$ denote the c^{th} largest value in $\Phi_{i,q}$, and define $\varphi_{i,q}^c = 0$ if $c > |\Phi_{i,q}|$.

Constraint 27: In any G-FP schedule of τ under the (m,n)-configured P-PCP:

$$i > m \implies \forall T_x \in \tau^L : I_x^S \leq \sum_{\forall \ell_q} N_{i,q} \cdot \sum_{c=1}^{m-1} \varphi_{i,q}^c.$$

Proof: By contradiction. Suppose $i > m$ and that there exists a $T_x \in \tau^L$ such that, in some G-FP schedule and for some resource ℓ_q, $I_x^S > \sum_{\forall \ell_q} N_{i,q} \cdot \sum_{c=1}^{m-1} \varphi_{i,q}^c$.

By Lemma 14, if J_i incurs stalling interference at some point in time t, then it has issued a request and is suspended at time t. Thus, if $I_x^S > \sum_{\ell_q} N_{i,q} \cdot \sum_{c=1}^{m-1} \varphi_{i,q}^c$, then there exists an interval $[t_a, t_b)$ during which J_i is waiting to acquire some resource ℓ_q that is not held by any task, and one or more jobs of T_x cause J_i to incur stalling interference for more than $\sum_{c=1}^{m-1} \varphi_{i,q}^c$ time units during $[t_a, t_b)$.

Let $t_m \in [t_a, t_b)$ denote a point in time at which both **(i)** J_x has caused more than $\sum_{c=1}^{m-1} \varphi_{i,q}^c$ time units of stalling interference during $[t_a, t_m]$, and **(ii)** a job J_x causes J_i to incur stalling interference at time t_m. Such a point in time exists since, by initial assumption, jobs of T_x cause more than $\sum_{c=1}^{m-1} \varphi_{i,q}^c$ time units of stalling interference during $[t_a, t_b)$.

We make the following observations.

1) By Lemma 14, at least one job in $LPR(i,t)$ is scheduled at any point in time at which J_i incurs stalling interference.
2) By Lemma 16, $\left|\bigcup_{t \in [t_a, t_b]} LPR(i,t)\right| \leq m$.
3) According to the definition of $\varphi_{i,q}^c$, the maximum total length of the (up to) $m-1$ longest requests of jobs in $LPR(i,t)$ is bounded by $\sum_{c=1}^{m-1} \varphi_{i,q}^c$.
4) Since J_i is waiting to acquire a resource that is not being held by any task throughout $[t_a, t_b)$, and since $i > m$, we have $|HPR(i,t)| + |LPR(i,t)| \geq m$ for all $t \in [t_a, t_b)$.

From observations (1), (2), and (3), we conclude that at time t_m at most one job remains in $LPR(i, t_m)$.

From observation (4), we conclude that $HPR(i, t_m) \geq m-1$ since $LPR(i, t_m) \leq 1$.

Since jobs in $HPR(i, t_m)$ are ready, we know that at least $m-1$ higher-base-priority jobs are scheduled at time t_m. Since J_i incurs stalling interference at time t_m, at most $m-1$ higher-base-priority jobs are scheduled at time t_m. This implies that exactly one lower-base-priority job is scheduled at time t_m, and that $LPR(i, t_m) = 1$.

To cause stalling interference, J_x must be scheduled. J_x has a lower base priority than J_i. From observation (1), we know that the remaining job in $LPR(i, t_m)$ is scheduled. Since exactly one lower-base-priority job is scheduled at time t_m, we conclude that $LPR(i, t_m) = \{J_x\}$. Therefore, J_x holds a resource in sr_x^i.

By Lemma 15, J_x did not enter $LPR(i, t_m)$ at any point in time during $[t_a, t_m]$. Hence, J_x's critical section started prior to t_a. By initial assumption, J_x has executed for more than $\sum_{c=1}^{m-1} \varphi_{i,q}^c$ time units during $[t_a, t_m]$ (recall that J_x must be scheduled to cause stalling interference). Thus, J_x's current critical section — accessing a resource in sr_x^i other than ℓ_q (which, by initial assumption, is not held by any task) — is longer than $\varphi_{i,q}^1$, which however is defined to be the maximum critical section length with respect to resources in sr_x^i (other than ℓ_q). Contradiction. ■

In addition to Constraint 27, we impose a constraint to limit the total cumulative stalling interference that J_i can incur due to all of the lower-base-priority tasks.

Constraint 28: In any G-FP schedule of τ under the (m,n)-configured P-PCP:

$$i > m \implies \sum_{T_x \in \tau^L} I_x^S \leq \sum_{\forall \ell_q} N_{i,q} \cdot \sum_{c=1}^{m} c \cdot \varphi_{i,q}^{m-c+1}.$$

Proof: By Lemma 14, if J_i incurs stalling interference at some time t then J_i has issued a request and suspended at time t. Suppose that J_i issues a request for ℓ_q at time t_a that is granted at time t_b, and that ℓ_q is available throughout $[t_a, t_b)$.

Under the (m,n)-configured P-PCP, we have $|HPR(i,t)| + |LPR(i,t)| \geq m$ for all $t \in [t_a, t_b)$. Hence, at any time instant $t \in [t_a, t_b)$, there are at least $|HPR(i,t)| \geq m - |LPR(i,t)|$ resource-holding higher-base-priority jobs.

Since resource-holding jobs are ready, when there are c ($1 \leq c \leq m$) jobs in $LPR(i,t)$, at least $m-c$ higher-base-priority jobs are scheduled at time t. Thus at most c jobs cause J_i to incur stalling interference at time t when $|LPR(i,t)| = c$.

By Lemma 16, there are a total of at most m jobs in $LPR(i,t)$ during $t \in [t_a, t_b)$. By Lemma 14, at least one job in $LPR(i,t)$ is scheduled at any point in time at which J_i incurs stalling interference. Hence, in the worst case, these jobs execute sequentially in the order of decreasing request length.

Consequently, while $|LPR(i,t)| = m$, J_i incurs stalling interference due to at most m lower-base-priority jobs for at most $\varphi_{i,q}^1$ time units; while $|LPR(i,t)| = m-1$, J_i incurs stalling interference due to at most $m-1$ lower-base-priority jobs for at most $\varphi_{i,q}^2$ time units; and while $|LPR(i,t)| = m-3$, J_i incurs stalling interference due to at most $m-3$ lower-base-priority jobs for at most $\varphi_{i,q}^3$ time units, *etc.*

Hence during $[t_a, t_b)$, lower-base-priority jobs cause J_i to incur stalling interference for a total of at most $\sum_{c=1}^{m} c \cdot \varphi_{i,q}^{m-c+1}$ time units. Consequently, across J_i's $\sum_{\forall \ell_q} N_{i,q}$ requests, lower-base-priority jobs cause J_i to incur stalling interference for a total of at most $\sum_{\forall \ell_q} N_{i,q} \cdot \sum_{c=1}^{m} c \cdot \varphi_{i,q}^{m-c+1}$ time units. ■

Next, we rule out stalling interference from a task T_x in case T_x and all tasks with base priorities lower than T_x do not access any resource with priority ceiling at least i. Recall that $\Pi(\ell_q)$ denotes the priority ceiling of resource ℓ_q.

Constraint 29: In any schedule of τ under the P-PCP:

$$\forall T_x \in \tau^L \text{ s.t.} \sum_{l \geq x} \sum_{\Pi(\ell_q) \leq i} N_{l,q} = 0 : I_x^S = 0.$$

Proof: By contradiction. Suppose a task T_x causes J_i to incur stalling interference even though $\sum_{l \geq x} \sum_{\Pi(\ell_q) \leq i} N_{l,q} = 0$ holds. Since $\sum_{l \geq x} \sum_{\Pi(\ell_q) \leq i} N_{l,q} = 0$, T_x and all tasks with lower base priorities always have an effective priority lower than

J_i. Suppose a job J_x causes J_i to incur stalling interference at some time t. Recall that J_x must be scheduled to cause stalling interference. We consider two cases.

Case 1: J_i is ready and not scheduled at time t. This is implies that m higher-effective-priority jobs are scheduled at time t. Since T_x has a lower effective priority than J_i, it is also not scheduled at time t. Contradiction.

Case 2: J_i is suspended and incurs indirect blocking at time t. Then there exists a job J_a that holds the resource that J_i seeks to acquire. By the definition of indirect blocking, J_a is ready, but not scheduled. Due to PI, J_a's effective priority is at least as high as J_i's base priority. Since T_x has a lower effective priority than J_i, it is also not scheduled at time t. Contradiction.

Case 3: J_i is suspended and incurs stalling interference at time t. Then J_i seeks to acquire a resource that is available, but is ineligible to do so since $|HPR(i,t)|+|LPR(i,t)| \geq m$. Note that neither J_x nor any J_l with $l < x$ is in $LPR(i,t)$ since T_x and all lower-priority tasks do not access any resources with a priority ceiling matching or exceeding J_i's base priority.

Therefore, there exist $|HPR(i,t) \cup LPR(i,t)| \geq m$ resource-holding, and hence ready, jobs with effective priorities higher than T_x's effective priority. T_x is thus not scheduled at time t. Contradiction. ∎

Next, we consider constraints for indirect and preemption pi-blocking. First of all, it is worth noting that Constraint 13 that is derived for the PIP in §V-B also applies to the P-PCP. We reiterate it in the following.

Constraint 30: When using the P-PCP:

$$\forall \ell_q : \sum_{T_x \in \tau^L} \sum_{v=1}^{N^i_{x,q}} X^I_{x,q,v} + X^P_{x,q,v} \leq \sum_{T_h \in \tau^H} N^i_{h,q}.$$

Proof: Under the P-PCP, in order to cause J_i to incur indirect or preemption pi-blocking, a lower-base-priority job must inherit the priority of a task with base priority higher than J_i. Since priority queues are used, each request of a higher-base-priority task is directly blocked by at most one lower-base-priority request. Thus the total cumulative number of higher-base-priority requests to a resource ℓ_q bounds the number of indirect or preemption pi-blocking that J_i can incur due to lower-priority requests to ℓ_q. ∎

Notably, the presented analysis of the P-PCP to this point does not reduce worst-case blocking in any case compared to PIP. Although the P-PCP limits the number of jobs that simultaneously hold shared resources, which Easwaran and Andersson [12] argued to potentially reduce priority inversion under a "reasonable" priority assignment,[4] in the worst case, tasks scheduled under the P-PCP are also exposed to *additional* delays (*i.e.*, stalling interference). Prior work did not investigate whether and when this tradeoff improves schedulability compared to the PIP; there is neither a detailed proof nor an empirical evaluation reported in [12].

To reflect the P-PCP's potential benefits as suggested by Easwaran and Andersson [12], we next *interpret* the intuition given in [12] (w.r.t. the potential benefit to reduce indirect and preemption pi-blocking) as a constraint and leave the corresponding proof as future work.

[4]Easwaran and Andersson consider a priority assignment to be "reasonable" if $d_i \leq d_j$ implies that T_i has a base priority higher than T_j, with the exception of the m highest priority tasks [12].

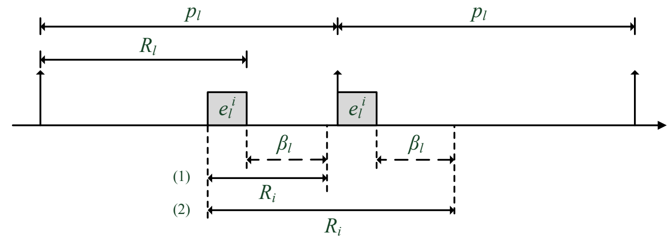


Fig. 9: Computing β_l.

In preparation, we introduce some additional notation. Let $e^i_l = \sum_{\forall \ell_q \in sr^i_l} N_{l,q} \cdot L_{l,q}$ denote the cumulative maximum execution time that any job of T_l can use the resources that have priority ceilings higher than the base priority of T_i. Let β_l be the *shift* value of T_l, which is intuitively defined to bound the relative release time of J_i and J_l such that the worst-case pi-blocking incurred by J_i due to jobs of T_l will not change if the release time of J_i is shifted to an earlier point within a range of β_l time units. Following the intuition given in [12], an illustration of the computation of β_l is sketched in Fig. 9.

$$\beta_l = \begin{cases} R_i + R_l - p_l - 2e^i_l & R_i > p_l - R_l + 2e^i_l, \\ R_i - e^i_l & R_i \in (e^i_l, p_l - R_l + e^i_l], \\ 0 & \text{otherwise.} \end{cases}$$

Further, let Γ_i be the set of (up to) m tasks in τ^L with the smallest β values, and let

$$R'_i = R_i - \min_{l>i} \sum_{\forall \ell_q \in sr^i_l} N_{l,q} \cdot L_{l,q}.$$

We use $N^{i\prime}_{l,q} = \eta_l(R'_i) \cdot N_{l,q}$ to denote the maximum number of requests that jobs of T_l can issue to resource ℓ_q during an interval of length R'_i. Then each task in $\tau^L \setminus \Gamma_i$ causes J_i to incur indirect and preemption pi-blocking at most $N^{i\prime}_{l,q}$ times.

Constraint 31: In any G-FP schedule of τ under (m,n)-configured P-PCP and a "reasonable" priority assignment:

$$\forall T_x \in \tau^L \setminus \Gamma_i : \forall \ell_q \in sr^i_x : \sum_{v=1}^{N^i_{x,q}} X^I_{x,q,v} + X^P_{x,q,v} \leq N^{i\prime}_{x,q}.$$

Refer to [12] for an explanation and justification of the bound.

In rare cases, Constraint 31 can lead to non-monotonic response-time estimates, *i.e.*, when iteratively searching for a fixed point as discussed in §V-C, the response-time estimate can potentially decrease from one iteration to the next when using Constraint 31. This is possible due to the definition of Γ_i, which is the set of m tasks (if any) in τ^L with the smallest β_l values. Since β_l depends on both R_i and R_l, the composition of Γ_i can change in non-monotonic ways during the fixed-point search. Consequently, the set of tasks to which Constraint 31 applies, namely $\tau^L \setminus \Gamma_i$, is not fixed, and hence the structure of the LP may not be static. Convergence of the fixed-point search is no longer guaranteed in this case.

Fortunately, this potential issue affects only Constraint 31 due to the underlying original definitions. (Easwaran and Andersson's response-time analysis [12] is similarly affected.)

Furthermore, non-monotonic response-time estimates are exceedingly rare: we applied the proposed P-PCP analysis, including Constraint 31, to more than 12 million task sets in our experiments and observed the effect a total of 14 times.

I. *Complete Set of Results*

For the sake of completeness and transparency, the complete schedulability dataset and all graphs stemming from the experiments discussed in §VI are made available for download at

- https://www.mpi-sws.org/~bbb/papers/data/rtss15.zip,

and for browsing at

- https://www.mpi-sws.org/~bbb/papers/data/rtss15/.

The results were obtained with SchedCAT, an open-source schedulability analysis toolkit, which is available online at:

- https://www.mpi-sws.org/~bbb/projects/schedcat.