Resolving Priority Inversions in Composable Conveyor Systems

Shivakumar Sastry[a,*], Aniruddha Gokhale[b]

[a] *Department of Electrical and Computer Engineering, University of Akron, Akron OH 44325-3904, USA*
[b] *Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN 37325, USA*

Abstract

The well known problem of priority inversions that occurs in classical real-time systems also manifests in decentralized cyber-physical systems. Using a specific example of composable conveyor systems, we show how priority inversions hinder the transport of entities through the conveyor systems. We present a novel adaptation of the classical priority inheritance protocol for resolving these cyber-physical priority inversions. While the approach resolves cyber-physical priority inversions, the structure and constraints of the conveyor systems cause the jitter associated with the end-to-end latency of the highest priority parts to increase. Further, these constraints also limit the applicability of the classical priority ceiling protocol in this class of cyber-physical systems. Simulation results demonstrate the correctness and reasonable communication overhead of the approach.

Keywords: Real-time Systems, Priority Inversion, Composable Conveyors, Networked Systems

1. Introduction

Composable, or reconfigurable, conveyor systems (CCS) are representative cyber-physical systems that capture the spatio-temporal interactions that can occur in a variety of emerging automation systems, particularly in the area of material handling. These cyber-physical systems are flexibile, easy to use, and can be dynamically reconfigured to assure real-time Quality of Service (QoS) in operational theaters such as warehouses, manufacturing lines, package sorting facilities (*e.g.*, FedEx and UPS), or front-line logistics for future military deployments. CCS are composed using basic building blocks that have pre-defined behaviors. These systems are interesting to study because tasks, which involve the

*Corresponding Author
Email addresses: `ssastry@uakron.edu` (Shivakumar Sastry), `a.gokhale@vanderbilt.edu` (Aniruddha Gokhale)

end-to-end transport of an entity in the system, evolve simultaneously both in time and space. Since these systems are well-structured, the desired behaviors and the unintended consequences that result from spatio-temporal interactions between tasks can be studied in a systematic manner. The insights gained from such a study can be applied to a large-class of cyber-physical systems.

Realizing CCS in practice is challenging. The intertwined dynamics of the cyber-physical elements, *e.g.*, the logic embedded in individual micro-controllers of CCS units, the wireless transceivers and protocols for messaging and coordination between the micro-controllers in physically adjacent units, and spatio-temporal evolution of the entities along end-to-end paths in the conveyor system present formidable challenges for addressing several design and operational issues. For example, a CCS designer may want to understand if a particular topology can yield a desired QoS without actually having to deploy and test the system; the designer may want to understand the robustness or resilience of the topology with respect to one or more failures. Our recent work [1, 2] addressed some of these questions. We developed intuitive abstractions and analysis tools to enable CCS designers to experiment with different layouts and analyze the QoS that could be achieved from the topology. In this paper, we address the issue of priority inversions that can occur in CCS because of the spatio-temporal interactions between tasks.

Consider a new conveyor system that must be designed to sort packages based on their service category. For example, "next morning delivery," "next afternoon delivery," or "ground delivery" are typical categories that are commonly used. This requirement imposes a notion of "priority" on the packages that are handled by the sorting system. Packages with different priorities arrive via Inputs (sources) and these packages move along paths to some Output. The paths of the conveyor system are formed by a sequence of physically adjacent units called Segments. Each Segment unit comprises a belt that can move the entity from one end to the other. One or more Segments in the system can also be incident with Turn units. The Turn units can merge multiple upstream paths to a single downstream path; alternatively, a Turn unit can also fork a single upstream path to multiple downstream paths. To improve the utilization of the units and resilience of the topology, it is necessary for many of these Input-Output paths to overlap. One consequence of such overlaps is that when two paths merge, low priority packages that are moving along one path can block higher priority packages that need to use the same path for an unbounded duration of time - thus resulting in a classical priority inversion [3]. Although the cyber-physical priority inversion problem may not cause significant disruptions in a package sorting facility, it will be a significant problem for assembly plants where parts much reach their designated positions in a timely manner.

The priority inversion described above is a cyber-physical phenomenon for the following reasons. As will be explained more precisely in Section 2.2, the cyber-physical priority inversion occurs because of both the physical topology of the conveyor system and the cyber decisions that impact the flow of entities on the system. In fact, the cyber-physical priority inversions are caused by an unfortunate consequence of the temporal sequence of priorities of the arriving entities, the physical location of the Inputs where the entities arrive on the system, the physical topology of the conveyor system, and the temporal sequence of routing decisions made at the different Turn units of the system. Such cyber-physical priority inversions are further exacerbated when the system is dynamically reconfigured because small changes to the structure can significantly impact the QoS that can be achieved using the conveyor system.

Classical techniques for resolving cyber-level priority inversions in centralized (cyber-only) real-time systems are well-known. The priority inheritance protocol (PIP) and priority ceiling protocol (PCP) in [3] and the stack resource policy in [4] are excellent solutions. In this paper we show that PIP can be effectively adapted to resolve the cyber-physical priority inversions in CCS; on the other hand, PCP cannot be readily adapted to address the problem.

The rest of the paper is organized as follows: Section 2 presents the problem statement more formally and surveys related work. We discuss the adaptation of PIP in Section 3. We show why PCP cannot be readily adapted in Section 4. Our results and discussion are in Section 5 and we conclude in Section 6.

2. Problem Statement and Related Work

We now illustrate more formally how the cyber-physical priority inversion problem occurs and briefly describe the related work. To better understand the problem statement, we first provide the model of composable conveyors we assume in this work.

2.1. Model of Composable Conveyor Systems

The conveyor systems we consider move entities from inputs ($\mathcal{I}$) to outputs ($\mathcal{O}$). These systems are composed using two kinds of units — *Segments*($\mathcal{S}$) and *Turns*($\mathcal{X}$) — that have fixed behaviors [5]. Each unit is autonomously regulated by a local microcontroller that interacts with adjacent units over wireless links to coordinate the transfer of entities. A Segment moves an entity over a fixed distance, in one of two assigned directions. Input and Output units are Segments that can move entities only in one direction. A Turn has four ports that can be configured to either bring in entities or remove entities. We assume that units can handle only one entity at a time. When two or more entities simultaneously arrive at the input ports of a Turn, only one entity is accepted by the Turn. We assume that a Turn will not accept an entity only when it is accepting a different entity with a higher priority. The entity that is not accepted for transfer must wait on the unit until it is accepted for transfer; such a wait will propagate to further upstream units because each unit can only hold a fixed number of entities.

We assume that Segments and Turns have a fixed direction that remains unchanged and there are no failures. The underlying directed graph of the system is acyclic. All units in the system can handle at most one entity; however, a unit can simultaneously *transfer-in* and *transfer-out* an entity. There are adequate buffers at the inputs to hold entities that are not yet admitted to the system. Recall, that multiple paths along which entities move overlap at one or more conveyor units. A physical entity that is already on a unit cannot be "preempted." In addition, the sequence of entities on adjacent Segments of the conveyor system cannot be physically reordered. These three reasons collectively cause priority inversions to occur as we explain below. Such inversions are inevitable in systems where resources must be preferentially allocated to tasks. It is, however, important to ensure that such inversion does not occur for an unbounded duration of time.

A conveyor system can be represented as a directed graph $G = (U, E)$. The nodes of G, $u_i \in U$ represent the units, *i.e.*, Segments, Turns, Inputs, and Outputs. An edge $(u_i, u_j) \in E$ represents the relation that an entity can move from unit u_i to unit u_j. Entities that arrive via input $I_k \in \mathcal{I}$ are delivered to a specific output $O_j \in \mathcal{O}$ along a path

$$P(I_k, O_j) =< u_1 = I_k, u_2, \cdots, u_n = O_j >$$

where $u_i \in U$. Such paths can either be pre-computed when the system cannot be reconfigured, or discovered and maintained when the system is reconfigurable using standard shortest path algorithms [6]. Since the paths merge at Turns, some of the paths may overlap and share common units.

2.2. Cyber-Physical Priority Inversion Scenarios in Composable Conveyor Systems

Consider the conveyor system shown in Figure 1. Here, three entities, namely τ_1^a, τ_2^b, and τ_3^c, have arrived via input I_1, I_2 and I_3, respectively. Suppose Segments S_1 and S_2 simultaneously send a request to Turn X_2 to transfer an entity. Assuming that the entities arriving via lower numbered inputs have higher priorities, i.e., $prio(I_2) > prio(I_3)$, X_2 must accept τ_2^b. Because S_1 cannot accept a new entity until its current entity is transferred out, the higher priority entity, τ_1^a, on X_1 is blocked by the lower priority entity, τ_3^c, on S_1. This blocking is an expected consequence of using priorities in the system. However, since the number of entities that can arrive via input I_2 is not bounded, priority inversion (i.e., blocking for unbounded time) can occur.

Using the definition of priority inversion in [3], we can infer that a cyber-physical priority inversion occurs whenever a high priority entity stream is intercepted by a low priority entity stream and this latter stream is intercepted by a medium priority entity stream. Figure 2 illustrates one such scenario.

At unit $X_m, m > 3$, the low priority entity stream from I_m intercepts the entity stream from I_1. Further downstream at X_3, the entity stream from I_3 intercepts the entity stream from I_m. In this example, the streams from I_m and I_1 are merged after X_m. Since $prio(I_3) > prio(I_m)$, entities from I_m can be blocked by entities from I_3 at unit X_3. When the units along the path $P(I_m, X_3)$ each have entities, and an entity from I_3 blocks an entity from I_m at X_3, a high priority entity that arrives via I_1 at unit X_m will be blocked. Since $prio(I_1) > prio(I_3)$ and the number of entities that arrive via I_3 are not bounded, cyber-physical priority inversion occurs.

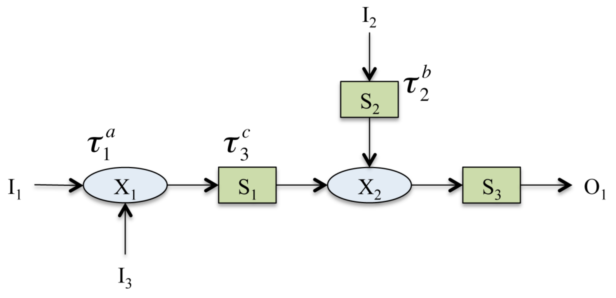

Figure 1: When multiple entity streams, each with a different priority, share common conveyor units, priority inversion can occur.

It is important to recognize that the cyber-physical priority inversion actually occurs only when certain entities arrive at the Turns at specific times; *i.e.*, just because cyber-physical priority inversion *can occur*, it does not mean that cyber-physical priority inversion *will occur for every entity.* For example, in the example shown in Figure 2, suppose every entity from I_3 arrives at X_3 after an entity from I_m has been accepted by X_3, cyber-physical priority inversions will not occur. However, the lack of certainty, *i.e.*, *whether or not cyber-physical priority inversion will occur*, makes it difficult to predict the throughput, end-to-end latency, jitter, and the utilization of these systems.

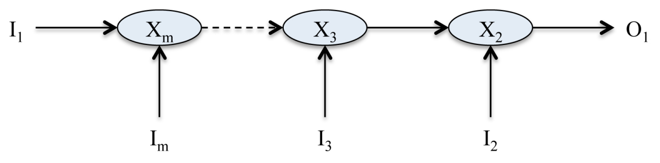

Figure 2: Priority inversion can occur when a low priority entity stream (I_m) intercepts (at X_m) a high priority entity stream (I_1) and, subsequently, a medium priority entity stream (I_3) intercepts (at X_3) the same high priority entity stream.

For certain topologies and combinations of entity arrival rates at the inputs, it is possible to offset the entity arrivals temporally or adjust the priorities of the inputs based on offline analysis. Such approaches are not applicable in general because the arrival rates are usually not known precisely and the solutions result in poor utilization and throughput. Moreover, the systems we consider are open in that we assume that entities of different priorities may arrive via different Inputs. Since the cyber-physical priority inversions are likely to occur in these systems, it is necessary to these inversions using an online protocol.

2.3. Related Work

A token-based scheme for achieving distributed mutual exclusion in a completely interconnected system is reported in [7]. Tasks executing on a node can access one or more resources by issuing a request with a specific priority. Each resource is guarded by a token; every node maintains a local queue of requests for its resource. The distributed queues are updated as requests were generated, acquired, or released. The paper presents four protocols to resolve priority inversion — priority ceiling (PCP), dynamic priority ceiling (DPCP), priority ceiling emulation (PCEP), and priority inheritance (PIP). The ceiling protocols associate a ceiling priority with each resource; this value represents the highest priority among all tasks that could request the resource.

In PCP the ceiling priority is computed via off-line analysis and in DPCP, this is computed online during resource contention. DPCP requires the state of all resources to determine the ceiling priority and is, hence, not scalable. Both DPCP and PCP may require priorities to change multiple times while a resource is acquired resulting in increased overheads. PCEP simplifies PCP by increasing the task priority to the ceiling priority when a resource is acquired. This eliminates priority changes when a resource is held, but could result in unnecessary blocking that PCP does not require. With the exception of DPCP, the ceiling protocols are suitable for systems with static priorities. PIP does not require a ceiling priority to be computed, and is suitable for systems with dynamic priorities.

In PIP, a task that uses a resource inherits the priority of the current highest priority task that is blocked on the resource. The structural neutrality of the system model, i.e., any task (node) could request any resource, limits the applicability of these protocols to the conveyor systems we consider.

Priority inheritance was successfully used to support remote procedure calls [8]. The system comprises nodes that have a predefined set of computations (tasks). Each task can invoke tasks on other nodes. All the nodes communicate with each other. Every incoming request to a node results in the creation of a dedicated task for the request that persists until it sends a response to the requester. Each node could only support a finite number of active tasks and this finite number represents the resources at that node. A task that invokes additional tasks in other nodes remains blocked until a response is received. Requests that arrive at a node are queued until sufficient resources are available to service it.

The authors present a distributed protocol that prevents deadlocks by limiting when a request could be issued. Priority inversions can occur when requests have priorities. To prevent unbounded priority inversion, PIP is used as follows: if a task τ_1 invokes a task τ_2 on another node, and τ_2 cannot be serviced by the remote node, all tasks and requests blocked on this resource inherit the priority of the highest task using the resource. The system model, the task model, and the structure of connectivity among all the nodes are not compatible with that necessary for the conveyor systems we consider.

In [9], priority inheritance is used in a multistage packet switching network to prevent priority inversions. This network routes packets from N inputs to

N outputs along fixed paths that each contain $\log_2(N)$ routers. Each router has two in-ports, two out-ports, and capacity to hold a single packet. Before sending packets, each router with a packet forwards the priority of its packet to the next router along its path. A router accepts the packet from its in-ports with the highest priority. The restricted topology of the network severely limits the applicability of this technique in the conveyor systems we consider. Flow control techniques [10] are not directly applicable because the routes over which entities move are not known in advance, and also because there are no buffers in the system where entities can be stored and forwarded. While scheduling techniques [11] can be used to minimize priority inversions, online techniques that can cope with varying arrival rates and changes in the system structure present formidable challenges.

3. Adapting the Priority Inheritance Protocol

We now present modifications to the existing cyber-level priority inheritance protocol to resolve cyber-physical priority inversions in composable conveyor systems. We also present a proof of correctness for the modified protocol, and an unintended consequence that results in increasing the jitter associated with the end-to-end latency of the highest priority entities.

3.1. Resolving Priority Inversions in Composable Conveyor CPS

Since we adopt and adapt the known cyber-level solution in the context of a cyber-physical problem, to make this protocol applicable we must map some of the physical entities into the cyber realm. In our formulation of the modified protocol, we view the movement of an entity from an input I_k to output O_j along $P(I_k, O_j)$ as a *task*. Entities arrive sporadically at I_k with a minimum inter-arrival time of T_k and a relative deadline D_k, *i.e.*, the entities must be delivered to O_j before D_k. The conveyor building blocks or units along the path $P(I_k, O_j)$ are the resources that are required by the task. All the entities are assigned a unique nominal priority. We use τ_k^j to refer to the j^{th} entity that arrived via input I_k.

A conveyor unit u_k is now imagined to be processing a entity within a finite time C_{u_k}. Time in the system advances in discrete ticks of duration $t_{tick} > \max_k\{C_{u_k}\}$. All units of the conveyor are synchronized at the beginning of each tick. In each tick, the units communicate to adjust entity priorities, agree on which entities can be transferred to corresponding downstream units, and ultimately physically transfer such entities. Note that because the electronic messages between the nodes propagate at a speed that is several orders higher than the speed at which entities move from one unit to another, we can assume that all the units can exchange electronic messages before the transfer of each entity.

Algorithm 1 depicts our cyber-physical priority inheritance protocol (CP-PIP) for composable conveyor systems. In the protocol, u_k is the unit that currently has entity τ_i^a. The objective is to determine the current priority for each entity on the system, $prio(\tau_i^a)$. We assume that each unit has already executed a topology discovery algorithm and knows its downstream unit, u_d for each entity that it handles and the length of the longest path in the conveyor system, LP[1]. The function `haveMessage()` will be true for a unit u_j whenever u_j has an entity τ_j^x and $prio(\tau_j^x) <$ *received-priority*.

Algorithm 1: A Cyber-Physical Priority Inheritance Protocol (CP-PIP)

```
Input   : $nominal\_priority$, $u_k$, $\tau_i^a$
Output: $prio(\tau_i^a)$
$CP-PIP(nominal\_priority, u_k, \tau_i^a)$
    $current\_priority = nominal\_priority(\tau_i^a)$
    $u_d = \text{downstreamUnit}(\tau_i^a, u_k)$
    send($u_d$, $current\_priority$)
    for $round \leftarrow 1$ to $LP$ do
        if haveMessage() then
            $current\_priority = received\_priority$
            send($u_d$, $current\_priority$)
        end
    end
    return $current\_priority$
end
```

The basic idea behind our modified protocol is as follows: to alleviate priority inversion, we allow a lower priority entity to *temporarily inherit* the priority of a higher priority entity it is blocking. The priorities of entities are used by the Turns to determine which entity must be preferentially accepted when multiple entities arrive simultaneously. Since there can be several Segments between Turns, it is necessary to propagate the inherited priorities over several downstream units to ensure that priority inversion cannot occur. In the worst case, and in certain pathological system topologies, this propagation may span all the units in the system and severely degrade the QoS.

In each time tick of the system, we determine the *current priority* of every entity. The priority of every entity is reset to its nominal priority, *i.e.*, the priority that was assigned to it when it arrived on the system. Every conveyor unit sends an `inheritance request` to its downstream unit on which the entity will travel. If a unit has an entity with a lower priority, it inherits the higher priority from the `inheritance request` it receives. The inherited priorities are propagated further to downstream units until no downstream priorities need to be elevated. Note that this inheritance is not permanent, and is only specific

[1]Since we have assumed that the underlying graph is a directed acyclic graph, the length of the longest path can be efficiently computed.

to the entity that is carried by the unit. The inherited priorities need not be propagated further when (a) a downstream unit does not have an entity, (b) a downstream unit already has a higher priority entity, or (c) beyond the last Turn along which the entity will move to its output bin.

Once the priority inheritance requests for the concerned entity are propagated to all the applicable downstream units, every unit will subsequently send a `entity transfer request` to its downstream using its current priority. Every unit that receives a `entity transfer request` must respond by sending either an `accept response` or a `reject response`. Since a unit that is currently handling an entity cannot accept a new entity from its upstream unit until it receives a `accept response` from its downstream unit, the `accept response` messages must also be propagated along several units in the same time tick. Recall that a downstream unit sends a `reject response` only when it has sent an `accept response` to a different upstream unit. All the involved units will then physically transfer their respective entities simultaneously to their downstream units or wait until they receive an `accept response`. Note that if there is a high-priority entity waiting in an upstream unit, all the involved entities would inherit an elevated priority; if there is no such high-priority entity, then the entities only use their nominal priority. This approach works because the number of hops over which the `inheritance request` and `accept response` messages are propagated is bounded and the time scales of electronic messages are several orders of magnitude smaller than the time scales of physical entity transfers.

3.2. Correctness of the Cyber-Physical Priority Inheritance Protocol

Note that in our approach, to prevent priority inversions, the higher priority of a blocked entity must be propagated downstream via the `inheritance request` messages. We can understand the correctness of this approach by considering the protocol shown in Algorithm 1 and the lemmas below.

Lemma 1. *After n rounds of Algorithm 1, a high priority of entity τ_i^a on unit u_k will be inherited by downstream unit u_j along the path on which τ_i^a moves, whenever u_j is less than or equal to n hops away from u_k and has the potential to block τ_i^a.*

Proof. Each unit knows its neighboring units, *i.e.*, upstream and downstream units, in the conveyor system and the entities along which entities move. Further, we have assumed no communication failures. Consequently, the proof follows immediately from induction on n. □

Lemma 2. *When Algorithm 1 terminates, all the current priority of entities on all the units in the conveyor system is the highest priority that the entity must use.*

Proof. The scenario when priority inversions occur in the worst-case is depicted in Figure 3. Let, $prio(I_j) > prio(I_k)$ whenever $j < k$. Here, entities from I_1

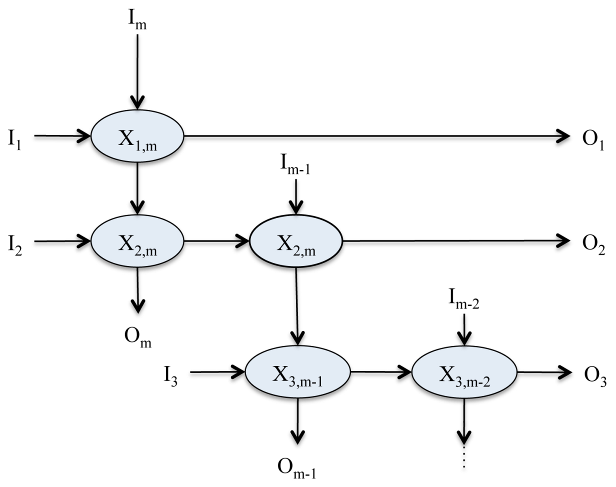

Figure 3: In the worst-case, priority inversions can occur on all units along a longest path in the system.

have the highest priority and entities from I_m have the lowest priority. Priority inversions also occur only when entities arrive in a specific temporal sequence.

Suppose m is even (for ease of notation). Consider the scenario when entities from I_m are already on all units along the path $P(I_m, T_{2,m})$ and an entity arrives at $X_{2,m}$ from I_2. $X_{2,m}$ accepts the entities from I_2. If an entity arrives from I_1 at $X_{1,m}$, priority inversion occurs. Suppose now that before the entities from I_2 arrive at $X_{2,m-1}$, entities from I_{m-1} fill the units along the path $P(I_{m-1}, X_{3,m-1})$ and entities arrive from I_3 at $X_{3,m-1}$. Once again, since $X_{3,m-1}$ must accept the entities from I_3, priority inversions occur $X_{2,m-1}$ and $X_{1,m}$. This pathological example can be extended further by reasoning in a similar manner. Nevertheless, the length of the path along which the priority inversions have occurred in this worst-case, is still less than or equal to the length of the shortest path. Consequently, the priority of the highest entity, *i.e.*, the entity from I_1 will be inherited by an entity from $I_{m-(\frac{m}{2}+1)}$ before it can be blocked by an entity $\tau^a_{\frac{m}{2}}$, *i.e.*, an entity that arrives from input $I_{\frac{m}{2}}$. □

Lemma 3. *When all units use the current priorities of their entities, no priority inversions can occur.*

Proof. From Lemma 2 we know that the highest priority of an entity in the conveyor system has been inherited by an entity that can be farthest away along the longest path and can potentially be intercepted by a medium priority entity. Moreover, this priority has been inherited in the same time tick in which the next entity transfer must happen. Since no medium priority entity can block the highest priority entity in the system, the Lemma follows. □

It follows from Lemma 1, Lemma 2, and Lemma 3 that

Theorem 1. *Algorithm 1 correctly resolves all priority inversions in every time tick of the system.*

3.3. Same Priority Blocking

Blocking is a pervasive phenomenon in any priority-based resource allocation framework. An entity can be blocked because a downstream Turn is respecting a priority assignment, or there is a communication failure, or priority inversion occurred. In our view, an entity is blocked for the duration of a time tick; however, the same entity can continue to remain blocked, perhaps for a different reason, in several consecutive time ticks.

Definition 1. *An entity τ_i^j is blocked on unit u_k whenever u_k does not receive an `accept response` in the same time tick after it sends a `entity transfer request`.*

With reference to the system in Figure 7, an entity from I_4 can be blocked

1. at X_{16} because entities from I_5 are on units X_1 and S_{34}, and unit X_5 is accepting entities from I_3, or
2. at S_2 because X_2 is accepting entities from I_3, or
3. at S_2 because X_{10} is accepting entities from I_1.

It is interesting to note that the structure of the conveyor systems causes the jitter associated with the end-to-end latency for the highest priority entities to increase when the CP-PIP protocol is used. When entities with the same current priority arrive at a Turn via different paths, *i.e.*, via different ports of the Turn, the best possible strategy for the Turn is to accept the entities in a round-robin manner. Note that since CP-PIP is used, such entities must have the same priority. This is a special kind of blocking, in which entities are blocked by other entities of the same priority. Hence, we refer to this as *Same Priority Blocking.*

Same Priority Blocking is a direct consequence of using CP-PIP because inputs have unique priorities and the system admits no cycle. Consequently, there is no opportunity for entities of equal nominal priority to compete for some unit in the same round. Once again, it is important to recognize that just because Same Priority Blocking *can* occur, it does not mean that *it will* occur.

4. Limitations of Using Priority Ceiling Protocol for Composable Conveyor Systems

The objective of the PCP protocol [3, 7] is to compute a ceiling priority for each resource; this priority is the highest priority among all tasks that could use the resource. When a task acquires a resource, the priority of the task is temporarily elevated to the ceiling priority of the resource. At first sight, it appears that a similar idea will be useful in the cyber-physical conveyor systems we consider.

This idea is appealing because once the ceiling priorities are computed, we need to recompute these priorities only when the topology of the system changes and there is no need to execute a priority inheritance protocol such as the one shown in Algorithm 1. Since the paths along which entities (tasks) move and the topology of the system is known, we can compute the ceiling priorities. Let $ceil(u_i)$ be the ceiling priority for unit u_i. Whenever an entity τ_k^j initiates a `request` from u_i, it uses $ceil(u_i)$ as its current priority. Using this approach, there would not be any need for messaging between the conveyor units to temporarily inherit priorities.

The structure of the conveyor systems allows us to compute the ceiling priorities in the two possible ways. Despite the attractiveness of the PCP, we now present counter examples to demonstrate that the ceiling priorities are inadequate for resolving cyber-physical priority inversions in conveyor systems.

4.1. Case 1: Direct Paths Computation

Let $\mathcal{P}_{u_i} = P_1, P_2, \cdots, P_K$ be a set of paths in the conveyor system such that $u_i \in P_j, 1 \le j \le K$. Then,

$$ceil(u_i) = \max_{1 \le j \le K} \{prio(I_j)\} \tag{1}$$

where $prio(I_j)$ is the nominal priority of the entities on P_j. Consider the system in Figure 4; entities that arrive via I_k are sent to O_k. The number that is shown on each unit is the ceiling priority computed using direct paths computation.

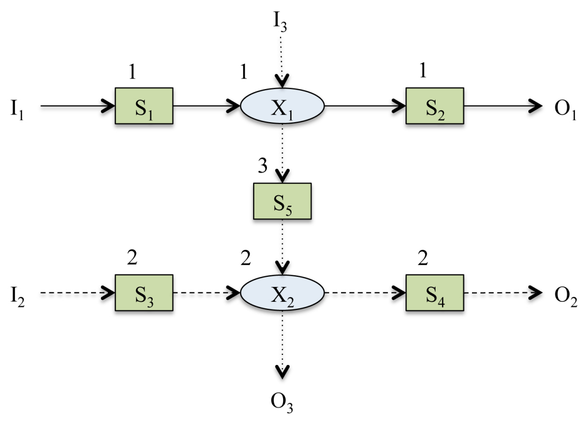

Figure 4: Direct Paths Computation Example. For each unit u_i, $ceil(u_i)$ is the highest priority among all entities that can move through this unit.

Consider the counter-example in Figure 5; the lowest priority entities from I_3 use the ceiling priorities of the resource and move along the path $P(I_3, O_3)$. This entity stream is blocked by higher priority entities from I_2. Because X_1 cannot accept τ_1^1 from S_1 until it has moved out τ_3^2, the high priority entity from

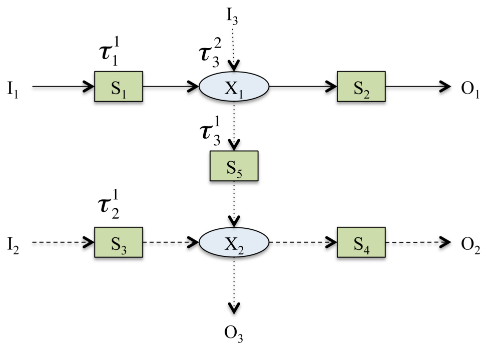

Figure 5: Counter Example. Ceiling priorities obtained using Direct Paths Computation are inadequate to prevent priority inversion.

I_1 is being blocked by a low priority entity from I_3. Meanwhile, τ_2^1 on S_3 and τ_3^1 on S_5 are competing for X_2. Since $ceil(S_5) = prio(I_3) < ceil(S_3) = prio(I_2)$, X_2 will accept τ_2^1. Since the number of entities from I_2 is not bounded, the blocking time for τ_3^1 on S_5, and hence the blocking time for τ_3^2 on X_1, is also unbounded.

Ceiling priorities computed here are inadequate to prevent priority inversions because the computation does not consider all the ways in which a high priority entity stream can be blocked by a lower priority entity stream. In the system shown in Figure 5, an entity from input I_3 that was on X_1 used $ceil(T_1)$ to move from X_1 to S_5. However, on S_5, this entity must use $ceil(S_5) = prio(I_3)$. Consequently, this entity gets blocked by higher priority entities from I_2. However, because of the physical structure of the conveyor system and because X_1 cannot "preempt" τ_3^2, the ceiling priority of S_5 also influences the priority-handling behavior of X_1.

4.2. Case 2: Blocking Paths Computation

In this approach, the ceiling priority of a unit is determined not only by the paths that include the unit, but also by considering the paths (tasks) it could block when it is blocked. Let $\mathcal{AP}_{u_i} = P_1, P_2, \cdots, P_K$ be a set of directed paths such that, $P_j = P(I_j, u_i), 1 \leq j \leq K$, if such a path exists. That is, these are the directed paths that start at some input and end at unit u_i. Then, the ceiling priority using the Blocking Paths Computation is

$$ceil'(u_i) = \max_{1 \leq j \leq K}\{prio(I_j)\} \qquad (2)$$

where $prio(I_j)$ is nominal priority of entities in P_j. Here, the ceiling priority of a unit is the highest priority among all paths that include u_i and the paths that can be blocked when an entity on u_i is blocked. For the system in Figure 5, all units except S_3 would have a ceiling priority $prio(I_1)$ because there is a directed path from I_1 to all other units. While this approach can resolve the priority

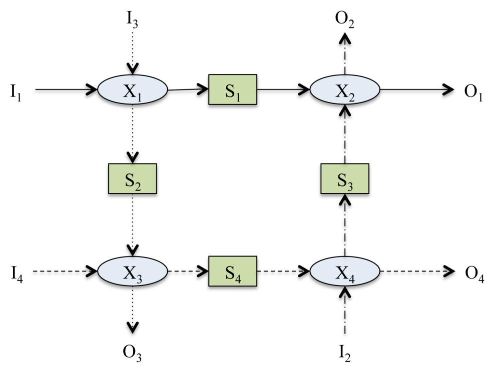

Figure 6: Counter Example for Blocked Paths Computation. Entities from all the inputs use $ceil'(u_i) = prio(I_1)$ and hence the priority assignments in the system are not respected.

inversion that occurred in the counter example in Figure 5, it introduces a new problem.

In certain topologies, this approach can result in ceiling priorities for all units being equal to the highest priority in the system. Consider the system shown in Figure 6. Since there is a directed path from I_1 to all units, $ceil'(u_i) = prio(I_1)$, irrespective of the input from which an entity arrives, it always uses the ceiling priority in the system. Hence, the priority assignments in the system are not respected.

In summary, we have seen that the structure of the conveyor systems allows us to compute ceiling priorities in two ways — namely, the direct paths computation and the blocking paths computation. The two counter-examples we present show that the ceiling priorities that are obtained in both approaches are inadequate to resolve priority inversions.

5. Evaluating the Cyber-Physical Priority Inheritance Protocol

We now present simulation results that demonstrate the effectiveness of the cyber-physical priority inheritance protocol (CP-PIP) in composable conveyor systems. The motivation for conducting the experiments was to understand whether resolving cyber-physical priority inversion impacts the QoS achieved for high priority entities, and whether the number of communication rounds necessary in Algorithm 1 were reasonable.

5.1. Simulation Approach

We designed a simulator for the composable conveyor systems using the OMNet++ discrete event simulation framework [12]. The results reported in this section are based on the representative package sorter conveyor system (Figure 7) that captures many of the challenges for cyber-physical priority inversion.

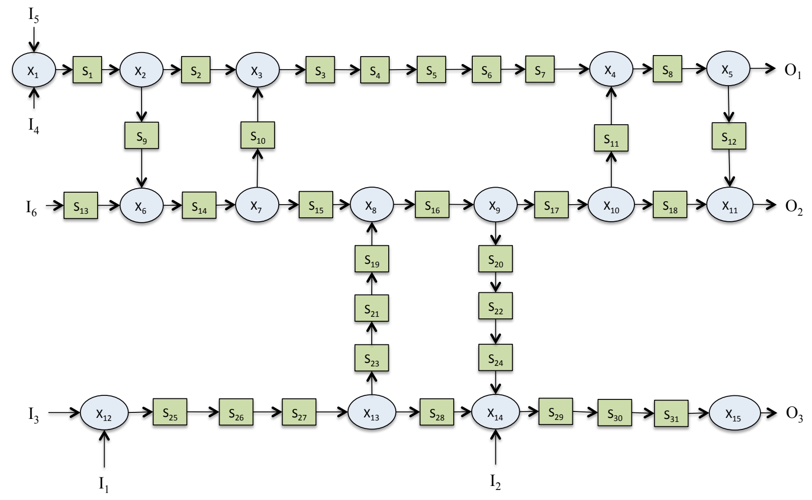

Figure 7: A Package Sorting Conveyor system. Entities arriving via Input I_k have priority k with entities from I_1 having the highest priority.

Entities move from one of the six inputs, $I_1, I_2 \cdots I_6$, to one of the three outputs, O_1, O_2, and O_3; entities from I_1 move through the turns X_{12}, X_{13}, X_8, X_9, X_{10}, and X_{11} to reach O_2. Entities from I_2 move to O_3 via the turns X_{14} and X_{15} and entities from I_3 move to O_3 via the turns X_{12}, X_{13}, X_{14}, and X_{15}. Entities from I_4 move to O_1 along the units on the top row of the figure. Entities from I_5 move to O_3 via the turns X_1, X_2, X_6, X_7, X_8, X_9, X_{14}, and X_{15}. Finally, entities from I_6 move through the turns X_6, X_7, X_3, X_4, and X_5 to O_1. Entities from I_1 had the highest priority and those from I_6 had the lowest priority, as described earlier.

Notice that the path $P(I_4, O_1)$ does not intersect with $P(I_1, O_2)$ or $P(I_2, O_3)$. When entities from I_5 are on the units on $P(X_6, S_{15})$, the entities from I_1 can block the ones from I_6. That is, entities from I_6 are rarely affected by entities from higher priority inputs I_1 and I_2. When entities from I_6 are on units in $P(S_{13}$,$S_{10})$, and the unit X_3 is transferring entities from I_4, the entities from I_5 can be blocked an for unbounded duration. Thus, entities from I_4 have the highest priority stream in the top part of the system in Figure 7, and entities from I_1 have the highest priority have the highest priority stream in the bottom part of this system. For this reason, we do not present the latencies along all the paths; since our focus is on evaluating the effect of priority inversions, we focus on the QoS for entities arriving via I_1.

To carry out the simulations, we injected entities into the conveyor system from each of the six inputs; the injection rate at Input I_k was R_k entities per second (eps). To evaluate the QoS achieved by entities from Input I_1, we varied R_1 from 0.1 to 1; simultaneously, to understand how entities arriving via other inputs affect the QoS of entities from I_1, we fixed the entity injection rates from all other Inputs, $R_k, 2 \leq k \leq 6$ at fixed values - namely, 0.1, 0.4, 0.6, 0.8 and 1; these values are shown as "RR" in the figure legends in this section. These injection rates correspond to a sporadic task model with $T_k = \frac{1}{R_k}$. We simulated the system under two scenarios — no protocol was used to resolve priority inversions in the first scenario, and in the second scenario, the current priorities of the entities were derived using the CP-PIP (Algorithm 1). Each scenario was simulated for 3600 seconds. The maximum capacity of the turn units was 0.8 eps and, hence, not all the generated entities were admitted into the system.

5.2. Impact of Resolving Cyber-Physical Priority Inversions

Figure 8 presents the average end-to-end latencies achieved (Y-axis) for entities from I_1 in both scenarios as the injection rate R_1 (X-axis) changes. The different lines illustrate the results for different values of entity injection rates (RR) at all the other Inputs. As illustrated in the figure on the left, entities from I_1, i.e., the ones with highest priority in the system, experience large average latencies that are sensitive to the injection rates. Notice that when the injection rates at all the other inputs are low, i.e., $RR = 0.1$, the average latency for entities from I_1 is low. When R_1 is increased, the average latency also increases. The steady increase noted from $R_1 = 0.6$ is because of congestion in the system. Notice that when the injection rates at the other Inputs are increased, the latency for entities from I_1 also increases; this is likely because of unresolved priority inversions. The figure on the right shows that even when the injection rates at all the other inputs are high, the latencies for entities from I_1 remains low. The increase in latency at $R_1 = 0.8$ is because the turn units have a maximum capacity[2] of about 0.8 eps and congestion occurs even when only the entities from I_1 move through the system.

To confirm that Algorithm 1 respects the priorities in the system, we carried out a simulation in which we jointly increased the injection rates at I_1 and I_3 while holding the injection rates at all the other inputs at fixed rates as described before. The average end-to-end latencies for entities from I_3 that were observed are shown in Figure 9. Note that as the injection rates increased, the latencies for entities from I_3 increased, while the latencies for entities from I_1 did not change from that shown in Figure 8.

Figure 10 presents the throughput achieved in the two scenarios, i.e., with and without a protocol for priority inversions. Notice that when no protocol was used (left), even when R_1 increased, the throughput remained low, especially when the entity injection rates at all the other Inputs were high. The spike at

[2]In our simulation, every turn unit required 1.15s to process each entity; hence the capacity was $\frac{1}{1.15} = 0.869$.

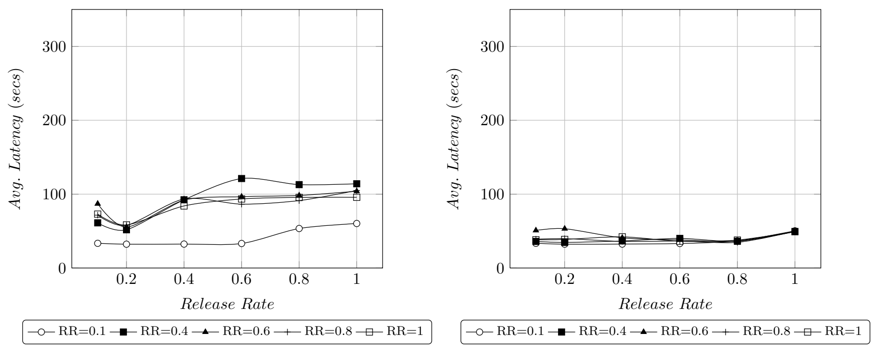

Figure 8: Average end-to-end latency for entities moving along $P(I_1, O_2)$ in the Package Sorter system when R_1 increased. The different lines show the effect of injection rates (RR) at all the other Inputs. The figure on the left presents average latency when no protocol was used to resolve priority inversions. The figure on the right shows that resolving priority inversions using Algorithm 1 improves the average latency for entities from I_1 even when the injection rates at the other Inputs increase.

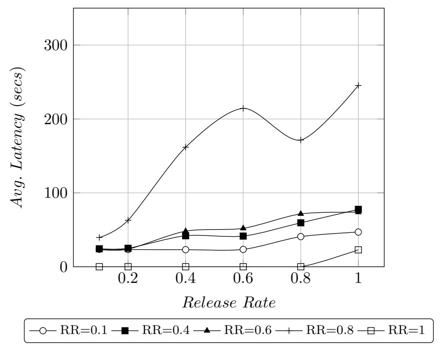

Figure 9: Average Latency for entities from I_3 increase with injection rate because Algorithm 1 respects priorities in the system and gives preferential treatment to entities from I_1.

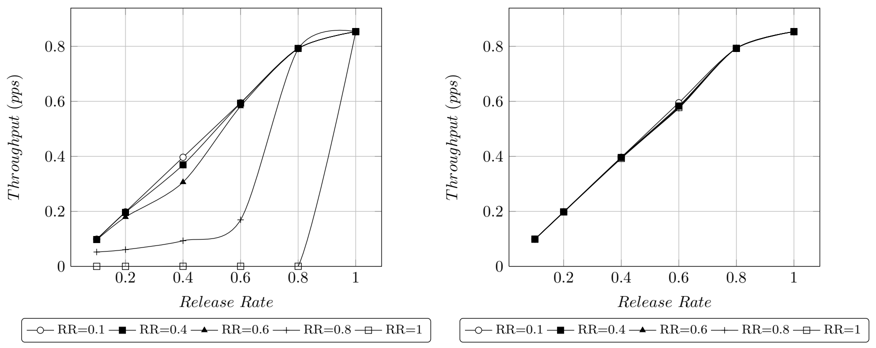

Figure 10: Throughput of entities from I_1 as R_1 increased. When no protocol was used to resolve the priority inversions (left) the throughput was low even when R_1 increased. When Algorithm 1 was used to resolve priority inversions, the throughput increased linearly with R_1, as expected.

the end is because all the entities were eventually flushed out of the system in the simulation. The figure on the right shows that the throughput of entities from I_1 increased linearly with R_1 when Algorithm 1 was used to derive the current priorities of the entities.

From the results in this section, we can conclude that Algorithm 1 effectively resolves cyber-physical priority inversions. The protocol performs well in the presence of congestion and achieves the best possible throughput. The results in the next section demonstrate the scalability of the approach.

5.3. Communication Rounds

The units in the conveyor system must propagate `inheritance request` messages downstream over multiple hops; `entity transfer request` messages always propagate one hop downstream, and `accept response` messages propagate multiple hops upstream. Since the `entity transfer request` messages always travel one hop, their overhead is fixed. To empirically validate the scalability of the proposed algorithm, we simulated the conveyor system in Figure 7 using the setup as discussed in the preceding evaluation. The number of priority inversions that occur and the number of hops over which each kind of message was propagated was recorded. The simulator did not limit the communication; instead, we messages were allowed to propagate until they could not propagate any further.

For the system shown in Figure 7, the maximum number of communication rounds for propagating the `inheritance request` messages is from I_5 to O_3, i.e., $LP = P(I_5, O_3)$. The actual number of hops over which the `inheritance request` message is propagated depends on the current priorities of the entities on the downstream units and entity injection rate at all the inputs. When the injection rates increase, the number of entities in the system increases and, hence, the chance of priority inversions occurring also increases. The number of priority inversions that occur is also a function of the exact sequence in which

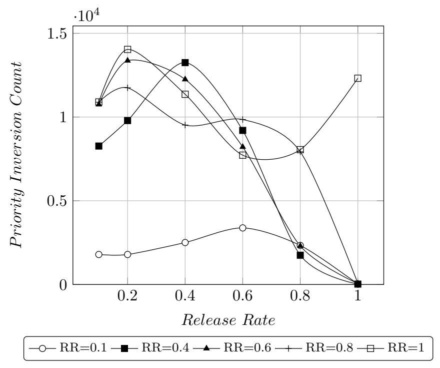

Figure 11: Total Number of priority inversion observed for different entity injection rates.

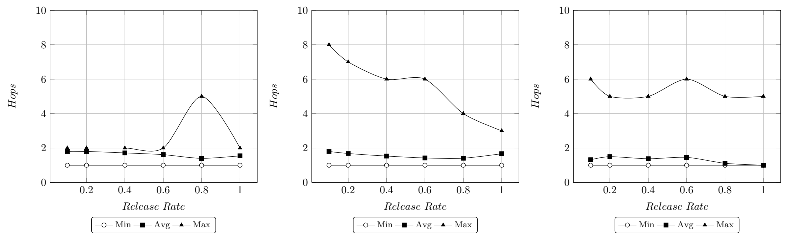

Figure 12: The average number of hops over which and maximum number of hops `inherit priority` message was propagated.

entities arrive on the system from different Inputs. Figure 11 presents the number of priority inversions that occurred; we obtained this by counting the number priority inheritance requests in the simulation. Notice that when the injection rates at all the inputs was low, i.e., $RR = 0.1$, the number of priority inversions was low. The number does not increase linearly as the injection rates increase because priority inversion.

Figure 12 presents the minimum, average, and maximum number of hops over which the `inheritance request` message was propagated in three scenarios, namely $RR = 0.1$ (left), $RR = 0.4$ (center) and $RR = 1$ (right). When $RR = 0.1$, the maximum number of hops over which these messages are propagated is low and the average is close to the maximum. As the injection rate at I_1 increased, the maximum increased consistent with the number of priority inversions that occurred (Figure 11). The figures for the cases $RR = 0.4$ (center) and $RR = 1$ are similar. Although we noted that the length of the longest path is an upper bound on the number of communication rounds necessary, this bound is not tight and these empirical results demonstrate the need to develop tighter bounds in the future.

The maximum number of hops over which the `accept response` messages propagated were also observed. For the Package Sorter, the reverse path from O_3 along $P(I_5, O_3)$ to X_8 followed by the reverse path from X_8 along $P(I_1, O_2)$

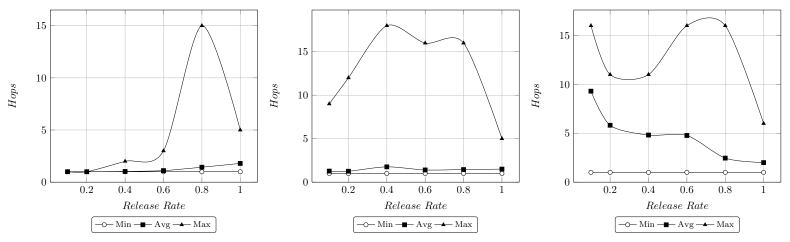

Figure 13: Minimum, average and maximum number of hops `accept response` message was propagated along the upstream units.

to I_1 represents the worst-case for the `accept response` messages. If every unit on this path had an entity and each of these was able to advance to the next unit, we would require 20 communication rounds to complete the transfers. The minimum, maximum, and the average number of hops observed for the `accept response` messages is shown in Figure 13. Notice that the number of hops over which these messages propagated were considerably higher than the number of hops over which the `inheritance request` messages were propagated. This is because priority inversions do not occur for all the possible entity transfers on the system. Further, `inheritance request` messages stop propagating when it encounters an unit that is processing an entity with current priority higher than the inherited priority. The units over which the `accept response` messages propagate are determined the decisions made at turns and the number of entities that can advance at any system time tick.

5.4. Discussion

In cyber-physical systems, the primary objective for resolving cyber-physical priority inversions is to improve the predictability for high-priority entities. The structure of the conveyor systems and the distributed realization of online protocols for resolving cyber-physical priority inversions introduces interesting challenges. First, the results showed that while the modified CP-PIP can resolve priority inversions, same priority blocking occurs and this results in increased jitter, and consequently reduced predictability, for the highest-priority entities. It is very interesting to examine techniques to mitigate the jitter for the highest priority entities.

We also encountered two counter examples that demonstrate that it is challenging to compute ceiling priorities that can effectively resolve priority inversions in these conveyor systems. It is interesting to explore whether there are alternative methods for determining ceiling priorities that are viable.

Finally, the cyber-physical equivalent of the task model considered in this paper was simple because each unit could handle only one entity. Depending on the application of the conveyor system, this task model must change. For example, a unit may handle multiple entities or batches of k entities from the same input and with the same nominal priority may move in the system. It remains to be seen how cyber-physical priority inversions manifest under these variations of the task models.

6. Conclusions

In this paper we presented how priority inversions, which are well-known in classical real-time systems, manifest in composable conveyor systems. When entities that arrive on the conveyors via some input have priorities, cyber-physical priority inversions can occur because multiple paths in the system share common resources. These inversions cause a higher priority entity to wait potentially for an unbounded duration on lower priority entities. To address this problem we presented an adaptation of the classical priority inheritance protocol that applies to cyber-level real-time systems to resolve cyber-physical priority inversion. While this protocol correctly resolves priority inversion, it increases the jitter for the entities with the highest priority in the system under certain situations. We presented two examples to show that a simple computation of ceiling protocols based on the paths along which entities move are inadequate to resolve cyber-physical priority inversion. In the case of direct path computations, cyber-physical priority inversions continue to occur and in the case of blocked path computation the priorities of the entities are no longer respected.

The simulation results confirmed that cyber-physical priority inversion is indeed resolved by the priority inheritance protocol and that the jitter caused for the entities with the highest priority is significant. The results also demonstrated that the performance of the priority ceiling protocol using the blocked paths computation was no better than that achieved using no protocol to resolve priority inversion. Priority inheritance continues to perform as expected in the presence of congestion. The communication overhead required for propagating the inherited priorities in the system was significantly lower than the estimated bounds. There are several interesting questions that remain to be addressed in this investigation. For example, it would be interesting to design scalable protocols that resolve priority inversion and deliver improved QoS in such systems. Considering failures and reconfiguration of the conveyor topologies in the system while still addressing the cyber-physical priority inversion problem will form the basis of our future work.

Acknowledgments

The authors thank Kranthi Mamidisetty, Brandon Archer, and Mukesh Chippa for executing some of the simulations. This work was supported in part by a grant from National Science Foundation under grant CNS-0720736. The opinions expressed are those of the authors and do not necessarily represent that of the National Science Foundation.

References

[1] K. An, A. Trewyn, A. Gokhale, S. Sastry, Model-driven Performance Analysis of Reconfigurable Conveyor Systems used in Material Handling Applications, in: Second IEEE/ACM International Conference on Cyber Physical Systems (ICCPS 2011), IEEE, Chicago, IL, USA, 2011, pp. 141–150.

[2] K. An, A. Trewyn, A. Gokhale, S. Sastry, Formal and Practical Aspects of Domain-specific Languages: Recent Developments, IGI Global, 2012, Ch. Design and Transformation of a Domain-specific Language for Reconfigurable Conveyor Systems, Chapt 19, pp. 553–571.

[3] L. Sha, R. Rajkumar, J. P. Lehoczky, Priority inheritance protocols: An approach to real-time synchronization, IEEE Transactions on Computers 39 (9) (1990) 1175–1185. doi:http://dx.doi.org/10.1109/12.57058.

[4] T. Baker, Stack-based scheduling of real-time processes, Real-time Systems 3 (1991) 67–99.

[5] B. Archer, S. Sastry, A. Rowe, R. Rajkumar, Profiling the primitives of networked embedded automation, in: Proceedings of the IEEE Conference on Automation Science and Engineering, Bangalore, India, 2009, pp. 531–536.

[6] R. Bellman, Dynamic Programming, Princeton University press, Princeton, N.J, 1957.

[7] F. Mueller, Priority inheritance and ceilings for distributed mutual exclusion, in: IEEE Real-time Systems Symposium, 1999, pp. 340–349.

[8] C. Sanchez, H. Sipma, C. Gill, Z. Manna, Distributed priority inheritance for real-time and embedded systems, in: A. Shvartsman (Ed.), Principles of Distributed Systems, Vol. 4305 of Lecture Notes in Computer Science, Springer Berlin, 2006, pp. 110–125.

[9] K. Toda, K. Nishida, S. Sakai, T. Shimada, A priority forwarding scheme for real-time multistage interconnection networks, in: Real-Time Systems Symposium, 1992, 1992, pp. 208 –217.

[10] D. Cansever, Decentralized algorithms for flow control in networks, in: Proceedings of the 28th Conference on Decision and Control, Athens, Greece, 1986, pp. 2107–2112.

[11] Y. Ma, C. Chu, C. Zuo, A survey of scheduling with deterministic machine availability constraints, Computers and Industrial Engineering 58 (2010) 199–211.

[12] A. Vargas, OMNET++Discrete Event Simulation System (2010).
URL `www.omnetpp.org`