RESERVATION BASED PROTOCOL FOR RESOLVING PRIORITY INVERSIONS IN COMPOSABLE CONVEYOR SYSTEMS

A Thesis

Presented to

The Graduate Faculty of The University of Akron

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Abdelrhman Mahamadi

December, 2014
RESERVATION BASED PROTOCOL FOR RESOLVING PRIORITY INVERSIONS IN COMPOSABLE CONVEYOR SYSTEMS

Abdelrhman Mahamadi

Thesis

Approved:  
Advisor  
Dr. Shivakumar Sastry

Accepted:  
Department Chair  
Dr. Abbas Omar

Committee Member  
Dr. Nghi Tran

Dean of the College  
Dr. George K. Haritos

Committee Member  
Dr. Jin Kocsis

Dean of the Graduate School  
Dr. Rex Ramsiey

Date

ii
ABSTRACT

A reservation based protocol for resolving priority inversions in decentralized real-time systems that are useful in the domain of advanced manufacturing is presented. The problem of priority inversions is well-known in the real-time systems literature. When there is prioritized access to shared resources, it is natural consequence for a low-priority task to be serviced before a high-priority task for a bounded duration of time, especially when the resource allocation cannot be preempted. Priority inversion is said to occur when a high-priority task is blocked for an unbounded duration of time by low-priority tasks. There are classical algorithms in the literature to resolve priority inversions in centralized, real-time systems.

Recent advances in computing and communications technologies, and the emerging needs for advanced manufacturing have motivated the quest for new, decentralized, platforms that can regulate the behavior of these systems. Toward this long-term objective, this work focused on a particular class of manufacturing systems called Composable Conveyor Systems (CCS). The manner in which priority inversions manifest in CCS is illustrated. It is shown that the classical protocols for resolving priority inversions cannot be directly adapted in all the operational scenarios of CCS. The new reservation based protocol for CCS is presented and its properties are established. Simulation results that demonstrate the efficacy of the new protocol are
presented. In the future, this protocol can be extended to address the needs of additional operational scenarios of CCS and a larger class of decentralized systems for advanced manufacturing.
ACKNOWLEDGMENTS

I would like to articulate my praise and gratitude to Almighty Allah for all physical and mental strength he offered my throughout my life and during this thesis work. I would also like to express my sincere gratitude to my advisor, Dr. Shivakumar Sastry, who was extremely supportive, available and very encouraging. I am so thankful to my lab mate Mukesh kumar Chippa, who offered much help through my masters. I am so thankful to my beautiful wife who dedicated her life to me. I am also thankful to my family loving mom, encouraging dad, my brothers and my lovely sister for their non-stopping love, support, encouragement and trust during my master degree especially and my life generally.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF FIGURES</th>
<th>viii</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II. BACKGROUND AND RELATED WORK</td>
<td>5</td>
</tr>
<tr>
<td>2.1 System Model</td>
<td>9</td>
</tr>
<tr>
<td>III. CCS RESOURCE ALLOCATION PROTOCOLS</td>
<td>15</td>
</tr>
<tr>
<td>3.1 Non Preemption Protocol</td>
<td>15</td>
</tr>
<tr>
<td>3.2 Priority Inheritance Protocol</td>
<td>16</td>
</tr>
<tr>
<td>3.3 Priority Ceiling Protocol and Stack Resource Policy</td>
<td>19</td>
</tr>
<tr>
<td>3.4 Reservation Based Protocol</td>
<td>20</td>
</tr>
<tr>
<td>IV. COMPUTING AVERAGE EXPECTED WAITING TIME</td>
<td>27</td>
</tr>
<tr>
<td>4.1 Expected average Waiting Time Computation for NPP</td>
<td>27</td>
</tr>
<tr>
<td>4.2 Expected average Waiting Time Computation for PIP</td>
<td>32</td>
</tr>
<tr>
<td>4.3 Expected average Waiting Time Computation for RBP</td>
<td>35</td>
</tr>
<tr>
<td>V. RESULTS AND DISCUSSIONS</td>
<td>36</td>
</tr>
<tr>
<td>5.1 Validating Analytical Results</td>
<td>36</td>
</tr>
<tr>
<td>5.2 Efficacy of PIP</td>
<td>38</td>
</tr>
<tr>
<td>5.3 Efficacy of the Reservation Based Protocol</td>
<td>41</td>
</tr>
</tbody>
</table>
VI. CONCLUSIONS AND FUTURE WORK .................. 48

BIBLIOGRAPHY ............................................ 51
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>When multiple entity streams, each with a different priority, share common conveyor units, priority inversion can occur.</td>
<td>12</td>
</tr>
<tr>
<td>2.2</td>
<td>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.</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>Deadlock Example</td>
<td>18</td>
</tr>
<tr>
<td>3.2</td>
<td>ChainedBlocking Example</td>
<td>18</td>
</tr>
<tr>
<td>4.1</td>
<td>Simple topology to analyze waiting time.</td>
<td>27</td>
</tr>
<tr>
<td>5.1</td>
<td>Under NPP, all the entities experience the same waiting time irrespective of the priorities. Notice that under PIP, the high priority entities experience a smaller waiting time while the waiting time for all other entities is roughly the same. This happens because of priority saturation. Under RBP, the priority is respected and higher priority entities experience smaller waiting times. Corresponding simulation results shown with dashed lines show that expected behavior is achieved in simulation.</td>
<td>37</td>
</tr>
<tr>
<td>5.2</td>
<td>A representative CCS topology that was used to evaluate the performance of the PIP and RBP protocols.</td>
<td>39</td>
</tr>
<tr>
<td>5.3</td>
<td>Average Latency of entities under SM1 at Input $I_1$ with NPP and PIP. The X-axis shows the Inter Arrival Times between entities at inputs $I_2$ through $I_6$.</td>
<td>40</td>
</tr>
<tr>
<td>5.4</td>
<td>The throughput of the entities released from $I_1$ with respect to the IAT of other inputs.</td>
<td>41</td>
</tr>
<tr>
<td>5.5</td>
<td>Average latencies and throughputs for entities with different priorities from $I_1$ under PIP and RBP, notice that RBP respects priorities</td>
<td>42</td>
</tr>
</tbody>
</table>
5.6 Average latencies and throughputs for entities with different priori-
ties from $I_2$ under PIP and RBP, notice that RBP respects priorities . 43

5.7 RBP prevents deadlocks from occurring. . . . . . . . . . . . . . . . . . . 45

5.8 Applying both PIP in the left and RBP in the right to the speical
topology in figure 3.2 and plotting the average waiting times of
tentities from $I_1$ at $X_1$ and $X_2$ both to test for chained blocking . . . . . 46

5.9 This figure shows the number of hops/Entity for reservation Request
messages for different priority levels generated from different inputs.
Priorities are assigned with $SM2$, with low release rates, except at
$I_3$ which releases entities with fixed priority level of 3 at high release
rate. Reservations for Entities with lower prorities from $I1$ are can-
celled by high priority requests from $I_3$ and therefore produces more
reservation request messages. Simillarly, entities with lower priority
from $I_5$ are cancelled from reservations requests of $I_3$, as $I_5$ and $I_3$
send entities to the same output unit. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
CHAPTER I

INTRODUCTION

Recent advances in computing and communication technologies have offered new platforms for the control and regulation of our manufacturing systems [1]. Despite the rapid adoption of early digital communications and distributed systems technologies into automation and manufacturing systems, the large installed base, economic, and safety concerns have created an inertia for change. Over the last few years, wireless communications technologies [2] and new decentralized platforms [3] have been explored. The integration of fine-grained cyber capabilities at multiple levels of traditional automation systems offers new advantages such as flexible reconfiguration to address emerging market demands, energy efficiency and scalable or composable performance. The adoption of such technologies also exposes several fundamental issues that must be revisited and addressed to ensure safe and robust operational environments in these critical systems. Composable or reconfigurable conveyor systems (CCS) are simple and well-structured systems that represent many of spatio-temporal interactions that can occur in a variety of automation systems. CCS are particularly useful in material handling applications and require the tight integration of cyber, i.e., control, computation and communication, and physical, such as material flow and physics, considerations for successful operations. The considered CCS are com-
posed using basic building blocks called Segments and Turns that have predefined behaviors; such a system is specified more precisely in the next section. These systems are flexible 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. These systems are interesting to study because tasks, which involve the 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 larger class of cyber-physical systems [4, 5]. Consider a CCS that is used to sort entities (e.g., packages) based on their service category. For example, "next morning delivery", "next afternoon delivery" or "ground delivery" are a few typical categories that are commonly used. This requirement imposes a notion of "priority" on the packages that are handled by the sorting system. Entities with different priorities arrive via one or more inputs (sources to the CCS) and move along paths in the CCS to some Output. The paths of the conveyor system are formed by a sequence of physically adjacent conveying units, i.e., Segments or Turns. To improve connectivity, utilization, and resilience of the CCS, the topology is designed so that many paths overlap, i.e., share common units or subpaths. One consequence of such overlaps is that when two paths merge, low priority entities that are moving along one path can block higher priority entities that need to use the same path for an
unbounded duration of time and thus resulting in a classical priority inversion. As is well-known in the real-time systems literature, lower-priority tasks may in fact be serviced before higher-priority tasks for a bounded duration of time whenever there is a need to share common resources. The situation where lower-priority tasks are serviced before higher-priority tasks for an unbounded duration of time is undesirable. This is the classical priority inversion problem [6, 7]; in centralized, real-time systems, this problem has been solved using classical protocols such as the Priority Inheritance Protocol (PIP), the Priority Ceiling Protocol (PCP) and the Stack Resource Protocol (SRP). When priority inversions occur in CCS, it disrupts the service architecture, i.e., fundamentally affects what priorities mean in the CCS and how the entities with specific priorities must be handled. When the CCS topology is known in advance and the arrival distributions of entities and their priorities are fixed, the activities may be scheduled a priori to avoid priority inversions; however, when the topology changes dynamically — either because of changes in demand or because of failures, and the arrival distributions cannot be fixed, a dynamic online protocol is needed to avoid priority inversions. Since the units of CCS are modular and autonomous, such a protocol must be decentralized, adaptable, and scalable. To address the above need for guaranteeing that priority inversions will not occur, the classical protocols for managing priority inversions have been investigated in order to know how could they be adopted to the domain of CCS. At first sight, it appears that the classical protocols can be directly adopted. The results in this thesis show that the decentralized nature of CCS and the variety of task models (described in more detail in
Section 2.1.1) that CCS admits expose several subtleties. Also, results here show that the classical PIP effectively resolves priority inversions in CCS in one task model but, it causes the priorities of all entities to be elevated in other task models. As is well-known, PIP does not prevent deadlocks or chained blocking and this was also observed in CCS. It is also discovered that classical protocols such as PCP and RSP could not be applied in the context of CCS. To effectively resolve priority inversions in CCS, a new Reservation Based Protocol (RBP) is proposed that performs better than PIP and avoids both deadlocks and chained blocking.

The remainder of this thesis is organized as follows: the CCS and its task models in more detail along with the related work in resolving priority inversions are described in the background Chapter. In Chapter 3, classical resource allocation protocols are described along with their adaptation to CCS and the issues that must be resolved. In Chapter 4, the analysis of the resource allocation protocols and the Average Expected Waiting Time (AEWT) computation for entities from a particular input with a certain priority are presented. In Chapter 5 the results and discussions are presented. Conclusions are given in Chapter 6.
CHAPTER II
BACKGROUND AND RELATED WORK

The priority inversion problem has been extensively studied in the literature [6]. Classical protocols for resolving priority inversions are the Priority Inheritance Protocol (PIP), Priority Ceiling Protocol (PCP) and the Stack Resource Protocol (SRP). For completeness, in PIP, priority inversions are resolved by temporarily elevating the priorities of the tasks that are currently blocked. In PCP and SRP, a system-wide ceiling protocol is used to determine whether or not to accept a task; once accepted, a task is guaranteed to complete with no further blocks. In the context of CCS, decentralized protocols for resolving priority inversions are under focus.

A token-based scheme for achieving distributed mutual exclusion in a completely interconnected system is reported in [8]. 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. The ceiling protocols associate a ceiling priority with each resource; this value represents the highest priority among all tasks that could request
the resource. While the ceiling priority is computed via off-line analysis in PCP, this is computed online during resource contention in DPCP. Further, 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. 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 in [8], i.e., any task (node) could request any resource, limits the applicability of these protocols in CCS. Here, entities move over specific paths and an entity can only "access" the units that lie on the Input-Output path for that entity.

Priority inheritance was successfully used to support remote procedure calls [9]. 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 $\tau_1$ invokes a task $\tau_2$ on another node, and $\tau_2$ 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 considered in this thesis.

In [10], 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 considered. Flow control techniques [11] 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 [12] 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.
In [13], the problem of ensuring temporal isolation in the presence of priority inversion is addressed. Here, strong temporal isolation means any misbehavior of a timing constraint by a task will not affect other tasks in the system. In classical priority inheritance protocol [6] a low priority task could inherit a higher priority and continue executing as long as it wants and this behavior could cause other tasks to miss their deadlines. In [13], the authors propose a reservation based scheme in that uses inheritance in the context of a reservation. If a lower priority task is executing and holding a resource during a specific reservation, then a higher priority task could preempt the lower priority and execute. However, if the higher priority task is requesting the same resource that is held by the lower priority task, then the lower priority task will inherit the reservation of the higher priority task and continue executing the critical section. After the lower priority task finishes executing the critical section, the inherited amount of reservation is given back to the higher priority task to continue executing. This design mitigates inversions and maintains strong temporal isolation.

In [14], the authors suggest methods to resolve priority inversion in centralized shared resources real time system. This work is interesting because one of these methods is reservation based. Here, a task must request a reservation and wait for the resource to be granted or denied before using the resource. The reservation mechanism works via two messages \((\text{hold}_j, r(t))\) and \((\text{next}_j, r(t))\); the hold request specifies the upper bound on the amount of time that the task will hold the resource \(r\) when it is granted access; the next request indicates the lower bound on the next time the task
will request resource \( r \). They introduced two polices for handling reservations. In the first policy, the requesting task must specify the upper bound time by which it will release the resource and that time has to be less that or equal to the start of the next reservation of all higher priority tasks (for the same resource). The second policy is more restricted; the upper bound release time of the requested resource should be less than or equal to all other reservations of higher priority tasks to all the resources and not only the requested resource as in the first policy.

2.1 System Model

The considered conveyor systems are composed using instances of two kinds of units \( \text{Segments} (\mathcal{S}) \) and \( \text{Turns} (\mathcal{X}) \); these units move entities from inputs (\( \mathcal{I} \)) to outputs (\( \mathcal{O} \)) [3]. Each unit is autonomously regulated by a local microcontroller. The microcontrollers in adjacent units interact over wireless links to coordinate the transfer of entities. A Segment moves an entity over a fixed length in one of two assigned directions. A Turn has four ports; each port can be configured either to move an entity into the turn or to move an entity away from the turn. A turn can handle only one entity at a time. When two or more entities simultaneously arrive at a Turn, only one entity is accepted by the Turn based on some policy, e.g., priority of the entity. Input units inject entities into the conveyor system and Output units consume units from the system.

A conveyor system can be represented as a directed graph \( G = (U, E) \). The nodes of \( G, u_i \in U \) represent the units, \textit{i.e.}, Segments, Turns, Inputs and Outputs.
An edge \((u_i, u_j) \in E\) represents the relation that an entity can transfer 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

\[ P(I_k, O_j) = \langle u_1 = I_k, u_2, \ldots, u_n = O_j \rangle \]

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 [15]. Since the paths merge at Turns, some of the paths may overlap and share common units. Because the units work asynchronously, any unit can simultaneously transfer-in an entity and transfer-out a (different) entity.

This simple model of conveying offers a versatile and scalable approach to realizing large conveyor systems that can be rigorously analyzed for structural and functional properties.

It is assumed that there are adequate buffers at the inputs to hold entities that are not admitted to the system. A physical entity that is already on a unit cannot be preempted. In addition, the sequence of entities on the units cannot be rearranged, except at the turns where priorities are used to determine how to reorder the sequence of entities. These constraints and the structure of the CCS cause priority inversions to occur as explained below. Such inversions are inevitable in systems where resources must be preferentially allocated. It is, however, important to ensure that such inversion does not occur for an unbounded duration of time.
2.1.1 Task Model for CCS

The primary activity in CCS is to move an entity from some input to the output where the entity must be delivered. This end-to-end transport of an entity is viewed as a task, $\tau_k$, in CCS. The notation $\tau^j_k$ could be used to refer to the $j^{th}$ entity that arrives on the CCS via input $I_k$. Depending on how priorities are assigned to tasks in CCS, two basic models for CCS are defined as follows:

- **Model 1:** A fixed priority, $\text{prio}(I_j) \in \{1, 2, \cdots, L\}$ is assigned with every input $I_k \in \mathcal{I}$. By convention, it is assumed that lowered numbered inputs have higher priorities. For every task $\tau^j_k$, $\text{prio}(\tau^j_k) = \text{prio}(I_k)$ is assigned. This means that in a Model 1 CCS, only entities with priority $\text{prio}(I_k)$ can arrive via Input $I_k$.

- **Model 2:** The inputs have no priorities assigned to them. For any task $\tau^j_k$, $\text{prio}(\tau^j_k) \in \{1, 2, \cdots, L\}$. This means that any entity with any allowed priority can arrive via any input of the CCS.

In addition, only fixed paths between inputs and outputs are considered. The units are assumed to operate without failures. This view of end-to-end transport of an entity as a task is versatile and can be extended various operational scenarios. For example, ideas of periodic, aperiodic and sporadic tasks for CCS can be defined in a natural manner [7]. This thesis only focus on aperiodic tasks that have stochastic arrivals both for the analysis in Chapter 4 and for the simulation results, reported in Chapter 5. Further, only the simple model of a task in which one task corresponds
to one entity is considered and models in which tasks may be preempted are not considered.

2.1.2 Priority Inversion in CCS

Consider a Model 1 CCS illustrated in Figure 2.1 with three tasks $\tau_1^a$, $\tau_2^b$, and $\tau_3^c$. Suppose Segments $S_1$ and $S_2$ simultaneously send a request to Turn $X_2$ to transfer an entity. Since $\text{prio}(I_2) > \text{prio}(I_3)$, $X_2$ must accept $\tau_2^b$. Because $S_1$ cannot accept a new entity until its current entity is transferred out, the higher priority entity, $\tau_1^a$, on $X_1$ is blocked by the lower priority entity, $\tau_3^c$, on $S_1$. Since the number of entities that can arrive via input $I_2$ is not bounded, this scenario results in priority inversion.

![Figure 2.1](image)

Figure 2.1: When multiple entity streams, each with a different priority, share common conveyor units, priority inversion can occur.

Using the definition of priority inversion in [6], it could be inferred that a priority inversion occurs whenever a high priority entity stream is intercepted by a low priority entity stream and this low priority entity stream is intercepted by a medium priority entity stream. Figure 2.2 illustrates one such scenario.
At turn $X_m, m > 3$, 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 $\text{prio}(I_3) > \text{prio}(I_m)$, 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 a 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 turn $X_m$ will be blocked. Since $\text{prio}(I_1) > \text{prio}(I_3)$ and the number of entities that arrive via $I_3$ are not bounded, priority inversion occurs.

It is important to recognize that the priority inversion actually occurs only when certain entities arrive at the Turns at specific times; i.e., just because priority inversion can occur, it does not mean that priority inversion will occur for every entity. For example, in the example shown in Figure 2.2, suppose every entity from $I_3$ arrives at $X_3$ after an entity from $I_m$ has been accepted by $X_3$, priority inversions will not occur. However, the lack of certainty, i.e., whether or not priority inversion will occur, makes it difficult to predict the throughput, end-to-end latency, jitter, and the utilization of these systems.
Figure 2.2: 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. For these reasons, it is necessary to resolve priority inversions in these systems using an online protocol.
CHAPTER III

CCS RESOURCE ALLOCATION PROTOCOLS

In this chapter, how classical protocols for resource allocation that resolve priority
tensions can be adapted for the CCS domain is discussed. Before discussing the
protocols, the NPP protocol is described which is a baseline used for comparison.
After describing the key ideas of PIP, PCP and SRP, the idea of priority saturation
in CCS is illustrated. Here, because of the behavior of these classical protocols, the
temporary priorities of all the entities are elevated to the same value and, hence,
the priority assignment is no longer respected. Also, how deadlocks and chained
blocking manifests in CCS are illustrated. Finally, the proposed reservation based
protocol (RBP) is described. RBP proven to yield optimality with respect to total
latency for all the admitted entities in CCS. It is also proven that RBP prevents
deadlocks and chained blocking.

3.1 Non Preemption Protocol

The simplest resource sharing protocol is the non preemption protocol (NPP). Since
a turn in CCS can only handle one entity at a time, by definition, CCS tasks must use
these resources under mutually exclusive access. Thus, the handling of an entity at a
A critical section in CCS\(^1\) can be viewed as the execution of a critical section in CCS. Under NPP, when a task is holding a resource no other task can preempt the holding task irrespective of the priorities involved. Thus, in effect, using NPP in CCS requires nothing to be done, i.e., it is the default behavior of the system. When two tasks with different priorities request access to a turn at the same time, the turn will grant access to the task with the higher priority. In some scenarios, using NPP might result in a priority inversion or large waiting time for high priority entities.

### 3.2 Priority Inheritance Protocol

The classical Priority Inheritance Protocol (PIP) can be adapted for Model 1 CCS in a natural manner. Here, when a higher priority entity is being blocked by a lower priority entity, the lower priority entity inherits the higher priority temporarily until the higher priority entities are serviced. Such an adaptation was reported in earlier work \[5\]. In addition to focusing only on Model 1 CCS, this earlier work \[5\] assumed a directed acyclic graph structure for the CCS topology and hence assumed that no deadlocks or chained blocking could occur. Further, only empirical simulation results for the PIP protocol were reported. In this thesis, more general setting of Model 1 and Model 2 are considered and the assumptions for the underlying topology are relaxed. Also analytical closed form expressions for the Average Expected Waiting

\(^1\)In general, a critical section in CCS can be a subpath that involves units that are shared along two Input-Output paths.
Time (AEWT) for entities are presented. As explained in Section 4, these results could be used to compute the AEWT for entities with a specific priority level based on the input from which they arrive, the topology of the CCS, and the protocol used to mitigate priority inversion, i.e., NPP, PIP and the new RBP protocol. The use of PIP for Model 2 of the CCS is investigated and it has been discovered that a subtle problem that is similar to priority saturation could occur in some circumstances. Under the relaxed assumptions of the underlying topology, it is also discovered that both deadlocks and chained blocking occurs in CCS when using PIP as is already well-known for centralized real-time systems [7].

3.2.1 Deadlocks when using PIP

Consider the two-input, two-output CCS topology illustrated in Figure 3.1. Suppose that all the entities from $I_1$ are being delivered to $O_2$; and all the entities from $I_2$ are being delivered to $O_1$. It could be recognized from this topology that the four classical conditions for deadlock, namely, mutually exclusive resource access, non preemption, hold and wait, and circular wait, are satisfied in the CCS subsystem comprising the units $X_1$, $S_3$, $X_2$ and $S_2$. This deadlock can occur whether NPP is used or PIP is used. PIP cannot resolve this deadlock because priority elevation will not break any of the four conditions for deadlock.

3.2.2 Chained blocking when using PIP

Consider the three-input, three-output CCS topology shown in Figure 3.2. There are three inputs $I_1$, $I_2$ and $I_3$ from which the entities are delivered to the outputs $O_1$, $O_2$ and $O_3$. Under the relaxed assumptions of the underlying topology, it is also discovered that both deadlocks and chained blocking occurs in CCS when using PIP as is already well-known for centralized real-time systems [7].
Consider the situation when $I_2$ and $I_3$ are releasing entities at a very high rates. In this situation, the turn $X_1$ is handling entities from $I_2$ most of the time; similarly, turn $X_2$ is handling entities from $I_3$ most of the time. Now, consider the situation when an entity arrives via $I_1$ going towards $O_1$. This entity will be blocked at $X_1$ and, again, at $X_2$. Such blocking will occur in CCS under both NPP and PIP. When PIP is used, the inherited priorities can reduce the blocking time; however, the chained blocking cannot be eliminated.
3.2.3 Priority Saturation when using PIP

Now consider the Model 2 CCS topology in Figure 3.2. Recall, that in this model entities with different priority levels are arriving via all the three inputs. Suppose that entities from all the three inputs are delivered to $O_3$. Consider the situation when all the resources, i.e., CCS units, shown in the figure have been allocated to lower priority tasks. Suppose now that a high priority task $\tau^j_2$ arrived. The entities on $S_2$, $X_1$, $S_3$, $X_2$ and $S_7$ inherit this higher priority, as expected under PIP. There is however a subtle problem here. After the resources have been allocated to several lower priority tasks, the arrival of a few higher priority tasks can elevate the priorities of all the tasks in the system thus resulting in priority saturation. This problem is more severe when the overlaps between the end-to-end paths is higher; such overlaps are, however, important to improve connectivity and reliability of the CCS.

3.3 Priority Ceiling Protocol and Stack Resource Policy

In centralized real-time systems, the classical Priority Ceiling Protocol (PCP) was designed to avoid deadlocks and chained blocking in fixed-priority systems [6]. Similarly, the Stack Resource Policy (SRP) was used in dynamic priority systems. However, in CCS both these protocols cannot be applied as noted in [5]. The main issue is that in many CCS topologies, the ceiling priorities of all the turns will be the same value - and hence this results in priority saturation because every entity in the CCS will inherit this ceiling priority and the system will behave as if there were no priorities
assigned at all. The reason why priority saturation occurs in PCP and SRP is because of the structure of the CCS and how the paths in CCS overlap. On the other hand, the reason why priority saturation occurs under PIP in Model 2 CCS is because of the dynamic pattern of entity arrivals that cannot be controlled. These issues limit the adaptation of the classical priority inversion protocols for CCS and motivated the design of a reservation based protocol for resolving priority inversions.

3.4 Reservation Based Protocol

RBP is inspired by PCP and SRP in the sense that before a task is released, i.e., the entity is transferred into the CCS by an input, the resources required by this entity are allocated. This means that every turn along the end-to-end path of this entity has a reservation for the task at a specific future time. Thus, after the task is released, it will experience no blocking; no task is released without a reservation. Turns that receive multiple reservation requests will adjust their local schedules based on the nominal priorities of the entities. The RBP protocol ensures that the schedules at all the turns are consistent to enable entity transfers without blocking. Upon arrival of a new entity at an input, the input unit sends a reservation request to all the turns on behalf of the newly arrived entity. This request includes an estimated time at which it will arrive at the downstream turns. The turn units in the CCS coordinate to adjust their reservation schedules based on the current set of reservation requests that were received at the turn from the inputs in the CCS. When the current reservation slot for an entity $\tau^a_k$ on a turn is adjusted, then the reservation for $\tau^a_k$ on all the downstream
turns may also need to be adjusted. Note that if any of the downstream turns have a conflicting reservation form a higher priority entity, then the reservation schedule on a set of turns along the path must be sequentially adjusted until there are no more conflicts. This process will converge because the reservation schedule for each turn can have as many slots as are needed to accommodate all the current requests. Since deadlines for the entities are not considered, the RBP algorithm is feasible. Thet lie along the end-to-end path of the entity. The algorithm produces as output the offset that must be applied to the entity before it is injected into the CCS. Below we present the notation for the RBP protocol and describe how the reservation schedule is built at each Turn.
Notation

$\mathcal{X}_k$ Set of turns $\{X^k_1, X^k_2, \ldots, X^k_{N_k}\}$, where $X^k_i$ is the $i^{th}$ turn in $P(I_k, O_j)$, i.e., along the path from $I_k$ to $O_j$.

$C(u_i, \tau^j_k)$ Time spent by $\tau^j_k$ on unit $u_i$.

$C(X_i, X_j)$ Time required for entity to move from turn $X_i$ to $X_j$.

$\text{RAT}(\tau^a_k, u_i)$ Requested Arrival Time for $\tau^a_k$ at unit $u_i$.

$\text{CAT}(\tau^a_k, u_i)$ Confirmed Arrival Time for $\tau^a_k$ at unit $u_i$,

$$\text{CAT}(\tau^a_k) = \text{RAT}(\tau^a_k) + \text{Offset}(\tau^a_k).$$

$\text{Offset}(\tau^j_k)$ Delay for $\tau^j_k$ before it can be admitted to CCS.

$\text{slots}(X_i)$ $[S_1, S_2, \ldots, S_{UB}]$ where $S_i$ is a slot in the schedule table of $X_i$ to store the scheduled entities

$\text{locks}(X_i)$ $[L_1, L_2, \ldots, L_{UB}]$ where $L_i$ is a boolean variable to store whether or not the slot is locked
Algorithm 1: RBP Messaging Structure

for all $I_k \in \mathcal{I}$ do
    Send RREQ$\left(I_k, X_k^i, \tau^i_k, C(I_k, X_k^i) + RAT(\tau^i_k, I_k)\right)$
    Offset$\left(\tau^i_k\right) \leftarrow 0$
    Update RREQ$\left(X_k^i, \tau^i_k, C(I_k, X_k^i) + RAT(\tau^i_k, I_k)\right)$
repeat
    Offset changed $\leftarrow$ false
    for $i = 1$ to $N_k - 1$ do
        Send RREQ$\left(X_k^i, X_k^{i+1}, \tau^i_k, RAT(\tau^i_k, X_k^i) + C(X_k^i, X_k^{i+1}) + \text{Offset}(\tau^i_k)\right)$
        Update RREQ$\left(X_k^{i+1}, \tau^i_k, RAT(\tau^i_k, X_k^i) + C(X_k^i, X_k^{i+1}) + \text{Offset}(\tau^i_k)\right)$
    end for
    if Offset changed = true then
        Offset changed $\leftarrow$ false
        for $i = N_k$ to 2 do
            Send RREQ$\left(X_k^i, X_k^{i-1}, \tau^i_k, RAT(\tau^i_k, X_k^i) - C(X_k^{i-1}, X_k^i) + \text{Offset}(\tau^i_k)\right)$
            Update RREQ$\left(X_k^{i-1}, \tau^i_k, RAT(\tau^i_k, X_k^i) - C(X_k^{i-1}, X_k^i) + \text{Offset}(\tau^i_k)\right)$
        end for
    end if
until Offset changed = false
$CAT(\tau^i_k, X_k^i) = RAT(\tau^i_k, X_k^i) + \text{Offset}(\tau^i_k)$
update RCNF$\left(X_k^i, \tau^i_k, CAT(\tau^i_k)\right)$
end for

Algorithm 2: Reservation Request

INPUT: Update RREQ$\left(X_k^i, \tau^i_k, RAT(\tau^i_k, X_k^i)\right)$
OUTPUT: Offset$\left(\tau^i_k\right)$
if $(\text{slot}(RAT(\tau^i_k)) = 0) \lor ((\text{lock}(RAT(\tau^i_k)) = 0) \land (\text{prio}(\text{slot}(RAT(\tau^i_k))) < \text{prio}(\tau^i_k)))$ then
    slot$\left(RAT(\tau^i_k)\right) = \tau^i_k$
    return
else
    Offset$\left(\tau^i_k\right) = \text{Offset}(\tau^i_k) + C(\tau^i_k, X_k^i)$
    Offset changed $\leftarrow$ true
    Update RREQ$\left(X_i, \tau^i_k, RAT(\tau^i_k, X_i) + \text{Offset}(\tau^i_k)\right)$
end if
Algorithm 3: Reservation Confirmation

INPUT: Update RCNF\(X^k_i, \tau^k_j, CAT(\tau^i_k, X^k_i)\)

OUTPUT: Update RCNC\(X^l_i, CAT(\tau^m_l, X^l_i)\)

if \(i = 1\) then
  for \(i = 1\) to \(N_k - 1\) do
    Send RCNF\(X^k_i, X^k_{i+1}, CAT(\tau^j_k, X^k_i) + C(X^k_i, X^k_{i+1})\)
    Update RCNC\(X^k_{i+1}, CAT(\tau^i_k, X^k_{i+1})\)
  end for
else
  for \(i = N_k\) to \(2\) do
    Send RCNF\(X^k_i, X^k_{i-1}, CAT(\tau^j_k, X^k_i) - C(X^k_{i-1}, X^k_i)\)
    Update RCNC\(X^k_{i-1}, CAT(\tau^i_k, X^k_{i-1})\)
  end for
end if

Algorithm 4: Reservation Cancellation

INPUT: Update RCNC\(X^k_i, CAT(\tau^j_k, X^k_i)\)

OUTPUT: Send RCNC\(X^k_i, I_l, \tau^m_l\)

if \(slot(CAT(\tau^j_k, X^k_i)) = \tau^m_l\) then
  Send RCNC\(X^k_i, I_l, \tau^m_l\)
end if
Theorem 1. The length of the schedule created at each turn $X_i \in \mathcal{X}$ by the Reservation Based Protocol is optimal with respect to the end-to-end latency of the admitted entities in the CCS.

Proof. For simplicity, we only consider the case where the local schedule at a turn does not have any unscheduled slots. Let $\sigma$ be the optimal schedule produced by the Reservation Based Protocol (RBP) and let $\sigma_i$ be the local schedule induced by $\sigma$ at the turn $X_i$. Let $L(\sigma_i)$ be the length of the schedule at the turn $X_i$. Since $\sigma$ is optimal, it follows from the Bellman’s principle of optimality that the induced schedule at each turn along every end-to-end path, $\sigma_i$ is also optimal. Consider a specific turn $X_k$ and let $M_k$ represent the number of entities that requested access and have a reserved slot $X_k$ in $\sigma_k$.

Suppose now that there exists another schedule $\sigma'$ that is also feasible, i.e., all the entities that were allocated reservation slots in $\sigma$ also have an allocation in $\sigma'$. Suppose that $L(\sigma') = L(\sigma) - 1$. This means that some turn along the path must have a shorter schedule than it did under $\sigma$ and let $X_k$ be the turn where $L(\sigma'_k) < L(\sigma_k)$. This can happen only when at least one of the $M_k$ entities that requested access to $X_k$ was not receive a reservation slot in $\sigma'_k$. This contradicts the assumption that $\sigma'$ was feasible and hence the theorem. \hfill \Box

Lemma 1. The Reservation Based Protocol prevents deadlocks.

Proof. The four necessary conditions for a deadlock to occur in any system are: (1) Mutually Exclusive Access to shared resources, (2) Hold and Wait, (3) Non-
preemption of Resources, and (4) Circular Wait. The standard technique to ensure that deadlocks do not occur is to break one of these four necessary conditions.

In a CCS, all turns are accessed in a mutually exclusive manner by the tasks and tasks cannot be preempted because of physical constraints. Circular waits can occur because of the topology of the CCS. Consequently, the only necessary condition that could be broken without constraining CCS topologies is Hold and Wait.

In the RBP, no task will be admitted to the system if all the resources required by the task are not allocated. Further, once the task is admitted to the CCS, all the resources allocated to the task are locked and cannot be pre-empted. This means that by construction of the protocol, the Hold and Wait condition can never occur, i.e., no task can be accessing a turn and requesting access to other turns in the system. Therefore, deadlocks can never occur when using the RBP.

Lemma 2. The Reservation Based Protocol prevents chained blocking.

Proof. When chained blocking occurs, the entity is blocked at each turn along its path because the turn is processing a higher priority entity that is moving along a different path.

Chained blocking can only occur when using the RBP when the entity does not already have a reservation on the turn where it is blocked. This contradicts the fact that no entity is admitted to the system without already having a reservation slot on every turn along its end-to-end path. Therefore, chained blocking can never occur when using RBP.
CHAPTER IV

COMPUTING AVERAGE EXPECTED WAITING TIME

In this chapter, closed-form expressions to estimate the Average Expected Waiting Time (AEWT) before which an entity with a specific priority level is delivered to its output are presented.

4.1 Expected average Waiting Time Computation for NPP

Consider the two-input, one-output CCS topology shown in Figure 4.1. Recall that in a Model 1 CCS, entities are assigned the priority of the port via which they enter.

If a portion of the CCS could be viewed as a queueing system with a deterministic server and assume Poisson arrival rate at the Input, this would be a classical $M/D/1$ queueing system. Further, if it is assumed that all the entities are homoge-
neous, then the service rate at the turn is constant. In such a system, AEWT for an
entity at a turn is

$$AEWT = \frac{2 - \rho}{2\mu \times (1 - \rho)}$$

(4.1)

where $\rho = \frac{\lambda}{\mu}$, $\lambda$ is the arrival rate and $\mu$ is the service rate. The quantity $\rho$ is usually
called the utilization of the queue.

Since the CCS topology in Figure 4.1 has two inputs, and the service at
the turn depends on the priority of the entities, the above $M/D/1$ model cannot be
utilized directly. The excepted number of entities that will be processed at the turn
before the particular entity of interest will be processed is needed to be calculated.
The fact that these entities could be from different queues must also be accounted
for, and the priorities of the entities must also be considered. For example, an entity
that arrives via $I_1$ (with priority $prio(I_1)$) will only wait for entities that were injected
into the CCS before itself. Hence, the AEWT for such an entity with priority 1 is

$$AEWT(w_1) = Q_1 \times \frac{1}{\mu}.$$  

(4.2)

However, an entity with priority 2, which arrives via $I_2$, may have to wait for
entities from $I_1$ and $I_2$ to be processed before it can be processed. Consequently,

$$AEWT(w_2) = (Q_1 + Q_2) \times \frac{1}{\mu}.$$  

(4.3)
In general, at any turn in the CCS, for an entity with priority $i$

$$AEWT(w_i) = \sum_{k=1}^{i} Q_k \times \frac{1}{\mu}$$  \hspace{1cm} (4.4)$$

where $Q_k$ is the expected number of entities that arrive via input $I_k$ at this turn, and are yet to be processed by the turn.

$$Q_k = \frac{2\rho_k - \rho_k^2}{2 \times (1 - \rho_k)}$$  \hspace{1cm} (4.5)$$

Now consider Model 2 CCS. Here, an entity with any allowed priority level can arrive via any of the inputs in the system. This cannot be viewed as a queue because there are different kinds of entities that must each be serviced differently because of their priority. The queuing analysis is needed to be augmented to compute AEWT in this model.

It is known that any entity $\tau_k^a$ that arrived via $I_k$ must wait for all the entities that arrived from the same input $I_k$ before itself. The question is how many entities from other inputs are going to be processed before $\tau_k^a$ is processed at the turn. Priority inversions that can occur at the turn is also needed to be accounted for.

With reference to the CCS in Figure 4.1, let $\tau_1^a$, with $prio(\tau_1^a) = 5$ be followed by $\tau_1^b$ and $prio(\tau_1^b) = 1$. Clearly, $\tau_1^a$ will arrive at $X_1$ before $\tau_1^b$. Suppose that $\tau_2^a$ with $prio(\tau_2^a) = 4$ arrived at $X_1$ at the same time when entity $\tau_1^a$ arrives at $X_1$. Since $X_1$ must respect the priorities of the entities, and because $X_1$ can only handle one entity at a time, it must accept $\tau_2^a$ before it accepts $\tau_1^a$. As long as there are entities from
that have higher priority than \( \text{prio}(\tau_1^a) \), the highest priority task \( \tau_1^b \) in the system will continue to be blocked by lower priority entities.

Generalizing this phenomenon, let \( Q_k \) be the expected number of entities that arrived via \( I_k \) at the turn, and let \( L \) be the number of possible priority levels. Given an entity \( \tau_k^a \) with priority \( j \), let \( bn_j \) be the number of other entities that would be processed at the turn before \( \tau_k^a \) is processed. Then the AEWT for this entity with priority \( j \) at that turn is

\[
AEWT(w_j) = bn_j \times \frac{1}{\mu}.
\]

(4.6)

The simple CCS topology shown in Figure 4.1 is considered. As an example, consider how \( bn_1 \) could be estimated, i.e., the number of entities that will be processed at the turn \( X_1 \) before a high priority entity \( \tau_1^a \) leaves \( X_1 \).

\[
E(bn(\tau_1^a)) = \sum_{m=1}^{N_2} m \times p_1(m)
\]

(4.7)

where \( N_2 \) is max number of entities can come from input \( I_2 \) based on the release rate at the input and \( p_1(m) \) is the probability that \( m - 1 \) entities from \( I_2 \) will the processed at \( X_1 \) before \( \tau_1^a \) will be processed.

As an example, when \( \tau_1^a \) is waiting at \( X_1 \), the turn can either process this entity or some other entity from \( I_2 \). If \( X_1 \) processes \( \tau_1^a \), the wait time is just the service time at the turn. Since \( L = 5 \), the entity from \( I_2 \) can have priority 1 with probability \( \frac{1}{5} \). In this case, the wait time at \( X_1 \) will be \( \frac{2}{\mu} \). The entity from \( I_2 \) will have a lower priority with probability \( \frac{4}{5} \), and in this case the wait time will also be
the same as the service time on the turn, i.e., \(1/\mu\). In general, if there is an entity with priority \(j\) waiting at the turn, the probability of the lead entity from \(I_2\) with lower priority is

\[
p_j(1) = \frac{L - j}{L} \tag{4.8}
\]

Similarly, if an entity with priority \(j\) is waiting at the turn, the probability that an entity from \(I_2\) will be processed before the waiting entity is processed is

\[
p_j(2) = \frac{1}{L} + \frac{(j - 1) \times (L - j + 1)}{L^2} \tag{4.9}
\]

In this case one entity from \(I_2\) will be processed before \(\tau_1^a\) this happens only when the priority of \(\tau_1^a\) is equal to that of \(\tau_1^a\), or the lead entity from \(I_2\) has higher priority than \(\tau_1^a\) and all other entities from \(I_2\) have lower priority than \(\tau_1^a\).

Generalizing, the probability that an entity with priority \(j\) must wait at the turn for \(m\) units of time in a CCS with \(L\) priority levels is

\[
p_j(m) = \frac{L - j + 1}{L} \times \left(\frac{j - 1}{L}\right)^{m-1}. \tag{4.10}
\]

where \(3 \leq m \leq N_2\).

On the other hand, when \(I_2\) is releasing entities at a very low rate, i.e., much slower than the service rate at the turn, \(bn_j\) for any entity will be 1 because there will be no queue buildup.

In the most general case, when an entity \(\tau_k^a\) with priority \(j\) is not waiting at
a turn,

$$AEWT(w_j) = (bn_j + (Q_1 - 1) \times bn_A) \times \frac{1}{\mu} \quad (4.11)$$

where $bn_A = (\sum_{k=1}^{L} bn_k)/L$. This is because the priorities of the entities are uniformly distributed among the allowed levels.

4.2 Expected average Waiting Time Computation for PIP

As known from real time scheduling PIP protocol will result into three types of blocking. In this section these forms of blocking will be considered and AEWT is computed in their presence. Through out this section, it is assumed that the entity of interest is the one that arrived most recently.

4.2.1 Direct blocking

In this form, a higher priority task is blocked because of a lower priority task at a shared resource, i.e., a turn in the CCS. This is a natural consequence of using priorities in the presence of shared resources.

To calculate the direct blocking time in a CCS it is assumed that the electronic messages in the CCS are one or two orders of magnitude faster than the physical transport of entities. This means once the entity arrives at the system it will send its nominal priority to be inherited by all the lower priority entities ahead. Now the blocking time due to direct blocking will be the waiting time in a M/D/1 queue at each turn along the path over which the entity must travel. Let $\lambda_m$ be the arrival rate at input $I_m$ that merge at a turn, and the service rate of the turn be $\mu$. If $\rho$
represents the utilization factor as before, then AEWT due to direct blocking is

\[ AEWT_{direct}(w_j) = \frac{2 - \rho}{2 \mu(1 - \rho)} \]  \hspace{1cm} (4.12)

4.2.2 Push through blocking

In this form of blocking of a medium priority task is blocked by a lower priority task because the lower priority task has inherited a higher priority. In CCS at each turn an entity suffer a push through blocking only if the priorities were elevated for the entities coming from a different input at the turn. This kind of blocking is not applicable for all the entities; for example, an entity with a priority 1 will not suffer this type of blocking because no other entity can inherit a priority higher than 1. This form of blocking depends on the priority level \( j \) of the entity. The number of entities which are coming from other input ports and going to be processed before the entity \( \tau_1^a \) will be processed is calculated.

\[ AEWT_{Push_j}(\tau_1^a) = \sum_{m=0}^{N} m \times p_j(m) \]  \hspace{1cm} (4.13)

where \( p_j(m) \) is the probability that \( m \) entities will be processed before an entity with priority \( j \) will be processed is

\[ p_j(m) = \frac{j \times (L^m) \times ((L - j)^{N-m})}{(L^N)} . \]  \hspace{1cm} (4.14)
4.2.3 Transitive inheritance blocking

This form of blocking happens when a medium priority task is in a nested critical section and a lower priority task can block a higher priority task by transitive inheritance. In CCS there is no nested critical section, and hence the blocking can happen only by direct blocking or push through blocking.

From the above, the total waiting time for an entity $\tau_1^a$ is given by the addition of both direct blocking and push through blocking:

$$AEWT_j(\tau_1^a) = AEWT_{direct}(\tau_1^a) + AEWT_{push}(\tau_1^a)$$ (4.15)

but as mentioned earlier this waiting time is calculated assuming that $\tau_1^a$ is the last entity in the queue. But if another entity arrives from the same input after $\tau_1^a$ with different higher priority then $\tau_1^a$ will inherit the priority of the new entity and that will change the expected waiting time. To get the general expression for any arbitrary entity arriving from $I_1$ first the expected waiting time as if this entity is the last entity in the queue is calculated, then that time is divided over the length of the queue to get how much expected time that entity will stay at each spot of the queue $Q_1$. Then only the first portion of that waiting time $\frac{E(w(\tau_1^a))}{Q_1}$ is considered and called the basic time for that entity $w_j(\tau_1^a)$, the expected waiting time during any of the next slots in the queue may change according to the priorities of the next coming
entities as follows:

\[
AEWT_j(\tau^a) = w_j(\tau^a) + (Q_1 - 1) \times (w_j(\tau^a) \times \frac{(L - j + 1)}{L} + \sum_{r=1}^{j-1} \frac{1}{L} \times w_r(\tau^a)) \quad (4.16)
\]

4.3 Expected average Waiting Time Computation for RBP

Applying the reservation based protocol to any CCS topology will make calculations of blocking time much easier, since each entity sends reservation requests to the turns, then the turn will send back either conformation for the requested time or shifting the requested time according to weather there are other higher priority entities requesting the turn at the same time. The number of entities which are going to use the turn before any new arriving entity is the total number of entity already admitted in the system plus the number of possible entities which are waiting in the queue i.e not admitted yet and will be admitted before the concerned entity \( \tau^a \). Let \( S_t \) be the total number of entities already admitted to the system and they have reservations in turn \( X_1 \), and let \( Q_t \) be the total number of entities those who are requesting turn \( X_1 \) but yet they are waiting in the admission queue, then the expected average waiting time for entity with priority \( j \) will be:

\[
AEWT_j(\tau^a) = S_t + (Q_t \times \frac{j}{L}) \quad (4.17)
\]
CHAPTER V
RESULTS AND DISCUSSIONS

The analysis in the preceding chapter yielded closed form expressions to compute the average expected waiting time for entities with different priorities when using the NPP, PIP and RBP protocols for resource allocation. In this chapter, the validity of the analytical results is demonstrated by comparing them with the performance achieved using discrete-event simulations. Using a representative conveyor system topology, it is confirmed that the proposed RBP protocol in fact performs better than the PIP protocol as determined through analysis. These results confirm that while PIP efficiently resolves priority inversions when entities of only one priority are injected from an input, i.e., System Model 1, different mechanisms, such as the RBP protocol are necessary to resolve priority inversions in more general system models.

5.1 Validating Analytical Results

Consider the CCS topology shown in Figure 4.1. This CCS system represents a two-input one-output CCS. This simple system is used to validate the analytical results for computing the average expected waiting time (AEWT). We allow entities with one of five priority levels, i.e., $L = 5$, to enter the system either from $I_1$ or $I_2$. All entities are destined to output $O_1$. Clearly, when high priority entities enter the system from
lower priority entities that arrive via $I_1$ must wait on Segment $S_1$ before they can be processed by turn $X_1$. The service rate and arrival rates were set as follows: $\mu_1 = 1.01$ entity/sec (for $X_1$), $\lambda_1 = 0.49$ (for $I_1$) and $\lambda_2 = 0.49$ entities/sec (for $I_2$). The AEWT was determined using Equation (4.11), Equation (4.16) and Equation (4.17) under the NPP, PIP and RBP protocols, respectively. The computed values are shown in Figure 5.1 using solid lines.

Figure 5.1: Under NPP, all the entities experience the same waiting time irrespective of the priorities. Notice that under PIP, the high priority entities experience a smaller waiting time while the waiting time for all other entities is roughly the same. This happens because of priority saturation. Under RBP, the priority is respected and higher priority entities experience smaller waiting times. Corresponding simulation results shown with dashed lines show that expected behavior is achieved in simulation.

The corresponding simulated values under these three protocol scenarios are also shown in Figure 5.1. Notice that in all the three scenarios, the actual values obtained in simulation are considerably lower than the values determined using the
analytic expressions. This is because the analytic expressions assume the worst sce-
nario and such a situation may not come to pass in practice. Nevertheless, notice that
the trends of the results are similar both through analysis and through simulation.
When RBP is used, the priorities of the entities are respected, i.e., higher priority
entities experience lower waiting times than the lower priority entities. Having estab-
lished the validity of the analytic expressions for AEWT under the three protocols,
now results that demonstrate the benefit of the PIP and RBP protocols are presented.

5.2 Efficacy of PIP

A representative CCS topology shown in Figure 5.2 was used as the reference system
to evaluate the efficacy of the protocols to resolve priority inversions. There are six
input ports and three output ports, in the lower part of this topology entities arriving
from $I_1$ are going towards $O_2$, while $I_2$ and $I_3$ are sending entities to $O_3$, in the upper
part of the topology, $I_4$ and $I_6$ both are sending to $O_1$ and $I_5$ sends entities to $O_3$.
Notice that $I_3$ is sharing $X_{1,2}$ with $I_1$ also $X_{1,4}$ and $X_{1,5}$ with both $I_2$ and $I_5$, the later
shares $X_8$ and $X_9$ with $I_1$, besides $X_1$ shared with $I_4$ which shares $X_3$, $X_4$ and $X_5$
with $I_6$.

The simulation was executed for System Model 1 (SM1) with NPP and PIP,
the average end-to-end latency was observed for entities arriving via input $I_1$. Entities
were injected from inputs $I_2, I_3, \cdots, I_6$ at the same Inter Arrival Time (IAT) between
entities; this IAT was systematically changed from 0.5 sec to 8 sec. For each value
that was selected, entities were injected from $I_1$ at two different IATs — one slow (10
sec) and one fast (1 sec). Recall that under SM1, entities are assigned the priorities of the inputs. Consequently, entities that are injected from $I_1$ have the highest priorities. The average end-to-end, i.e., from input to the output, latency experience by entities from $I_1$ was recorded and is shown in Figure 5.3. Notice that under NPP, for low Inter Arrival Time between entities of $I_1$, the entities from $I_1$ appear to constant latencies when changing other inputs IAT; this is expected — there are large number of entities from $I_1$ with the highest priority in the system and consequently, the latency will not be affected by other inputs IAT. While under the high Inter Arrival Time of $I_1$ scenario, the latency of the high priority entities (from $I_1$) becomes very high because of the experienced priority inversions and when the IAT at the other inputs increases (i.e., farther along the X-axis), the latency of these entities from $I_1$ decreases.

Figure 5.2: A representative CCS topology that was used to evaluate the performance of the PIP and RBP protocols.
Notice that under PIP, the latency is consistently low; this is expected because under SM1, PIP enables the high priority entities to get preferential service in CCS. The throughput of entities from $I_1$ was recorded and is shown in Figure 5.4. Notice that under low Inter Arrival Times of $I_1$, the entities from $I_1$ appear to have high throughput in both scenarios; this is expected — there are large number of entities from $I_1$ with the highest priority in the system and consequently, the throughput will be high. Notice that under the high Inter Arrival Time of $I_1$ scenario, the throughput of the high priority entities (from $I_1$) falls because of the experienced priority inversions and when the IAT at the other inputs increases (i.e., farther along the X-axis), the throughput of these entities from $I_1$ increases. Notice that under
PIP, the throughput is consistent; this is expected because under SM1, PIP enables the high priority entities to get preferential service in CCS.

![Graph](image)

**Figure 5.4:** The throughput of the entities released from $I_1$ with respect to the IAT of other inputs.

Despite the efficacy of PIP in SM1, the priorities of the entities saturate under PIP in System Model 2 (SM2). Recall, in SM2 entities are not assigned a priority based on the input. Instead, entities with any priority can enter the system from any input. Under this scenario, the inherited priorities of the many entities get elevated in the CCS.

5.3 Efficacy of the Reservation Based Protocol

To understand how RBP works in SM2, three simulations were carried out in which RBP and PIP were compared. First, RBP was evaluated in the context of the re-
representative CCS topology shown in Figure 5.2. The average latency and throughput experienced by entities arriving at $I_1$ under RBP and PIP are shown in Figure 5.5. The X-axis represents entities that were assigned different priority levels. Recall, in simulation of SM2, every entity is assigned a random priority level that is uniformly drawn from $1, 2, \cdots, L$. Notice that entities that have higher priorities experience lower latencies under RBP. In contrast, the latencies experience by entities under PIP is roughly the same. The figure on the right shows throughput. Notice that under RBP, higher priorities receive preferential service in CCS and hence the throughput of these entities is higher. To confirm that the entities from a different input of the

![Figure 5.5: Average latencies and throughputs for entities with different priorities from $I_1$ under PIP and RBP, notice that RBP respects priorities](image)

(a) **Average Latency** ($I_1$)  
(b) **Throughput** ($I_1$)

CCS experience different latencies and throughputs, the latencies and throughput of entities that arrive at $I_2$ were examined. These results are shown in Figure 5.6. Once
again, it is noticed that under PIP (in SM2), the priorities saturate and hence the entities experience the same average latency even when their priorities are different. In contrast, these priorities are respected under RBP. The throughput plot on the right also shows the expected trend. As discussed in Chapter 3, PIP does not man-

![Average Latency (I₂) and Throughput (I₂)](image)

(a) **Average Latency (I₂)**  
(b) **Throughput (I₂)**

Figure 5.6: Average latencies and throughputs for entities with different priorities from I₂ under PIP and RBP, notice that RBP respects priorities

age deadlocks and chained blocking can also occur. In earlier work [5], it have been assumed that the topologies are constrained to be directed acyclic graphs. Since in this thesis that assumption is no longer held, it was important to make sure that the protocol for priority inversion, RBP, indeed prevents deadlocks from occurring and avoids chained blocking. To study both these issues, additional simulations we carried out using special CCS topologies that were presented as examples in Chapter 3. For these simulations, L = 5 levels, μ₁ = μ₂ = 1 entities/sec, λ₁ = 0.5 entities/sec
for input $I_1$, and $\lambda_2 = 0.5$ entities/sec for $I_2$. To ensure that RBP prevents deadlocks from occurring, the CCS topology shown in Figure 3.1 was simulated for one hour and the end-to-end latencies and throughput for entities with different priorities from $I_1$ were captured. These results are shown in Figure 5.7. The two figures on the top show the average latency and throughput for entities under PIP. Notice that the average latency is $-1$ — however, the throughput is also zero. This means that no entities arrived at the outputs and hence the latencies were not computed in the simulator. In contrast, the bottom two figures show the latency (left) and throughput of the entities that was achieved using the RBP. Chained blocking is the phenomenon where entities would get blocked at each turn along their path. To ensure that RBP prevents chained blocking, the same simulation parameters were used as described in the preceding paragraph. In addition, $\lambda_3 = 0.5$ for $I_3$ was set. The topology shown in Figure 3.2 was simulated for one hour under RBP and PIP. The entities that were injected from $I_1$ were tracked and the waiting time these entities experienced at the two turns, $X_1$ and $X_2$, were recorded. Figure 5.8 shows the results. Notice from the figure on the left that entities experience delays both at turn $X_1$ and $X_2$. In contrast, once an entity is admitted into the CCS, it experiences no delays along its path under RBP as confirmed by the figure on the right.
Figure 5.7: RBP prevents deadlocks from occurring.
Figure 5.8: Applying both PIP in the left and RBP in the right to the special topology in figure 3.2 and plotting the average waiting times of entities from $I_1$ at $X_1$ and $X_2$ both to test for chained blocking.
Figure 5.9: This figure shows the number of hops/Entity for reservation Request messages for different priority levels generated from different inputs. Priorities are assigned with SM2, with low release rates, except at $I_3$ which releases entities with fixed priority level of 3 at high release rate. Reservations for Entities with lower priorities from $I_1$ are cancelled by high priority requests from $I_3$ and therefore produces more reservation request messages. Similarly, entities with lower priority from $I_5$ are cancelled from reservations requests of $I_3$, as $I_5$ and $I_3$ send entities to the same output unit.
CHAPTER VI
CONCLUSIONS AND FUTURE WORK

This thesis focused on resolving priority inversions in composable conveyor systems (CCS). Composable conveyor systems are decentralized, real-time, systems that are composed using instances of two kinds of units, namely Segments and Turns. Each unit is autonomously regulated by an embedded micro controller that is also integrated with a low power radio transceiver. The primary objective of the CCS is to move entities from inputs to outputs. The transport of each entity through the CCS is viewed as a real-time task. CCS is interesting to study because these real-time tasks evolve both in space and in time.

Priority inversion is a classical problem that is well-known in real-time systems literature. Here, because of the confluence of a sequence of events, a low priority task may block a high priority task for an unbounded duration of time. This is undesirable and, hence, it is important to resolve priority inversions. In small scale systems, it may be possible to analyze the topology and design schedules that could avoid priority inversions. However, when the CCS topology is changing either because of new user demands or because of failures, online protocols are necessary to mitigate priority inversions.
The classical protocols that are used to resolve priority inversions in centralized, real-time systems are the Priority Inheritance Protocol (PIP), Priority Ceiling Protocol (PCP) and the Stack Resource Policy (SRP). Because of the different nature of tasks in CCS, it was necessary to consider two different system models. In a model 1 CCS, every entity that arrives via an input is assigned the priority of the port. In model 2, entities with any priority may arrive via any input. The classical PIP resolved priority inversions in a model 1 CCS. On the other hand, both PCP and SRP result in a problem called priority saturation. In this situation, the priorities of all the entities are temporarily elevated to the same high value and consequently, the priority assignments of the entities are no longer respected in the system. It was shown that in a model 2 CCS, PIP also results in priority saturation. As is well-known in the literature, both deadlocks and chained blocking could occur when PIP was used and it was shown that these two phenomena also occur in CCS topologies. However, unlike centralized real-time systems, PCP and SRP could not resolve the problems with PIP because of the priority saturation problem. Thus, it was necessary to design a new online protocol for resolving priority inversions in CCS.

A new Reservation Based Protocol (RBP) was presented. RBP was inspired by PCP and SRP in the sense that any entity that was admitted to the CCS had a confirmed reservation on all the resources along its path in the system. It was shown that RBP effectively resolved priority inversions both in model 1 and in model 2 CCS. To evaluate these protocols, closed form expressions were derived for the average expected waiting time (AEWT) for an entity with a specific priority at a specific
It was shown that the closed form expressions derived for all the protocols were good upper bounds for the waiting time demonstrated through simulations. Extensive discrete event simulations were carried out to evaluate the protocols for priority inversions using a representative CCS topology. The results confirmed that RBP performed better than PIP and prevented both deadlocks and chained blocking.

There are several interesting avenues for future research in this area. Some of the next steps include more general task models and dynamic path maintenance to support failures of conveyor units and changing user requirements.
BIBLIOGRAPHY


