Priority Assignment for Real-Time Wormhole Communication in On-Chip Networks

Zheng Shi and Alan Burns
Real-Time Systems Research Group, Department of Computer Science
University of York, UK, YO10 5DD
{zheng, burns}@cs.york.ac.uk

***Abstract*—Wormhole switching with fixed priority preemption has been proposed as a possible solution for real-time on-chip communication. However, none of current priority assignment policies works well in on-chip networks due to some inherent properties of the protocol. In this paper, a novel heuristic branch and bound search algorithm is introduced to explore the possible priority ordering. Differing from the traditional exhaust algorithm which costs exponential complexity, our algorithm can effectively reduce the search space. In addition, this algorithm can ensure that if a priority ordering exists that makes the traffic-flows schedulable, this priority ordering will be found by the search algorithm. By combining with schedulability analysis, a broad class of real-time communication with different QoS requirements can be explored and developed in a SoC/NoC communication platform.**

I. Introduction

On-chip networks (NoCs) [1], [2], has emerged as a new design paradigm to overcome the limitation of current bus-based communication infrastructure [3], and has had an increasing importance in today's System-on-Chip (SoC) designs. The typical architecture of on-chip networks consists of multiple *intellectual property (IP)* modules connected through an interconnection network. This architecture offers a general and fixed communication platform which can be reused for a large number of SoC designs.

As an emerging design paradigm, NoC has some new features [4]: better wire utilization through sharing reduces the traffic congestion and power consumption, a segmented architecture enables parallelism through pipelining and has high scalability, the computation decoupled from communication leads to IP module reusage and interconnections to be designed separately.

Multiple IP-cores based design using NoC allows multiple applications to run at the same time. These applications execute the data processing and exchange information through the underlying communication infrastructure. Some of applications have very stringent communication service requirements, the correctness relies on not only the communication result but also the completion time bound. A data packet received by a destination too late could be useless. These critical communications are called *real-time* communications. For a packet transmitted over the network, the communication duration is denoted by the *packet network latency.* The maximum acceptable duration is defined to be the *deadline* of the packet. A *traffic-flow* is a packet stream which traverses the same route from the source to the destination and requires the same grade of service along the path. For *hard real-time* traffic-flows, it is necessary that all the packets generated by the traffic-flow must be delivered before their deadlines even under worst case scenarios. In another words, the maximum network latency for each packet can not exceed its deadline. A set of real-time traffic-flows over the network are termed *schedulable* if all the packets belonging to these traffic-flows meet their deadlines under any arrival order of the packet set.

As a popular switching control technique, wormhole switching [5] has been widely applied for on-chip networks due to its greater throughput and smaller buffering requirement [6]. However, routing packets in a wormhole network is partially non-deterministic because of the contention in communication. These contentions cause possible delay and jitter, leading to the violation of the timing constraints of the packets. In order to make it possible to predict the maximum network latency for each packet, it is possible to schedule the real-time communications using a priority-based approach [7]–[9] that allows the serving of traffic-flows with higher priorities before the traffic-flows with lower priorities. One novel schedulable analysis approach [10] has been proposed recently to predict whether all the real-time packets can meet their timing properties at the static design phase. By evaluating diverse inter-relationships and service attributes among the traffic-flows, this approach can predict the packet transmission latency for a given traffic-flow based on two quantifiable different delays: direct interference from higher priority traffic-flows and indirect interference from other higher priority traffic-flows.

One of the common problems that need to be solved in the development of priority-based communication is, how to assign priority to each traffic-flow so that all the traffic-flows loaded on the network will meet their time constraints. The main purpose of this paper is to explore the priority assignment policy in real-time on-chip networks. We present at the beginning a case to show that none of current priority assignment policies works well in wormhole switching networks. Then a heuristic branch and bound search algorithm is introduced to find the possible priority ordering. Differing from the traditional exhaust algorithm which costs exponential complexity, our algorithm can effectively reduce the search space. In addition, our algorithm can ensure that if a priority ordering exists that makes the traffic-flows schedulable, this priority ordering will be found by the search algorithm. By combining with schedulability analysis, a broad class of real-time communication with different QoS requirements can be explored and developed in a SoC/NoC communication platform.

The rest of this paper is organized as follows: section II introduces the related works and their drawbacks. The real-time communication model and associated schedulablility analysis are reviewed in section III. Section IV describes a group of novel heuristic algorithms for searching the assignment of priorities in real-time on-chip networks. In section V, by experiments based on simulation, the evaluation results of our algorithms are presented. Finally, section VI concludes the paper.

II. Related Works and Drawbacks

Priority assignment policies for fixed priority scheduling on single-processor system have been studied widely. The early work, proposed by Liu and Layland [11], introduced the *Rate Monotonic (RM)* priority assignment policy, such that the task with shortest period is given the highest priority. However, RM policy requires a restrictive system model, deadline and period must be identical for each task in the system. Later, Lenug and Whitehead [12] lifted this constraint and showed the *Deadline Monotonic (DM)* priority assignment policy, allowing a task's deadline to be less than the period; where the shorter deadline the task has, the higher priority value is assigned. Audsley in [13], [14] designed a general assignment technique with polynomial time complexity that is guaranteed to find a schedulable priority ordering if one exists in a system. Recently, Zuhily and Burns [15] introduced the *Deadline minus Jitter Monotonic (D-JM)* priority ordering for a task set with non-zero release jitter.

Priority assignment in wormhole switching networks was firstly addressed by Mutka [16], which is based on the RM approach. Li and Mutka [17] later presented *Laxity Monotonic (LM)* priority assignment policy and a related run-time adjustment method. The laxity is defined as the difference between the deadline and maximum basic network latency, which corresponds to the amount of blocking a packet can tolerate. Priorities are assigned based on laxity with the highest priority going to the traffic-flow with the smallest laxity. Hary's approach [8] improved the original LM and RM algorithms by accounting for the numbers of hops a traffic-flow travels. The priority of a traffic-flow which travels a long distance is increased.

However, none of these priority assignment polices is referred to as *optimal*. An optimal priority assignment approach means it can provide a schedulable priority ordering whenever such an ordering exists. The following case shows that none of RM, DM or LM can produce a schedulable priority ordering corresponding to a set of traffic-flow with relationships illustrated in Figure 1 and attributes in Table I. The time units are not necessary in this case as long as all the traffic-flows use the same base.

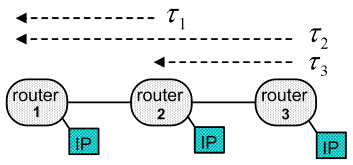

Fig. 1. A Case of Real-Time Traffic-Flows

Real-Time Traffic-Flow	Basic Network Latency	Period	Deadline
τ_1	1	2	2
τ_2	1	2.5	2.5
τ_3	1.5	3.25	3.25

TABLE I
Traffic-Flows Description

The priority orderings produced by RM, DM or LM policies is exactly the same in this case, τ_1 gets the highest priority, then τ_2; τ_3 is assigned the lowest priority. The analysis method given in [10] (outlined in section III-C) calculates the worst case network latency of τ_1, τ_2 and τ_3 as 1, 2, 3.5 respectively. Flow τ_3 misses its deadline and so the system is unschedulable with this priority ordering. If these priority assignment policies are optimal then we should not be able to find another priority ordering which results in all the deadlines being met. However if we assign the traffic-flows with the following priority ordering, priority $\tau_2 >$ priority $\tau_1 >$ priority τ_3, then the corresponding worst case network latency for each traffic-flow are 1 for τ_2, 2 for τ_1 and 2.5 for τ_3. All the traffic-flows are now schedulable. This case shows the current priority assignment policies are not optimal for the priority-based real-time on-chip communication.

III. Network Model and Schedulablility Analysis

This section gives an outline of the real-time on-chip network's structure, the network characteristics and traffic pattern models that are needed to formulate the schedulability analysis.

A. Priority-based wormhole switching

Wormhole switching is a very popular *cut-through* switching strategy for on-chip networks [6]. The packets arriving at an intermediate node are immediately forwarded to the next node without buffering. Each packet in a wormhole network is divided into a number of fixed size flits [5]. The header flit takes the routing information and governs the route. As the header advances along the specified path, the remaining flits follow in a pipeline way. If the header flit encounters a link already in use, it is blocked until the link becomes available. In this situation, all the flits of the packet will remain in the router along the path and only a small flits buffer is required in each router.

In order to support a real-time communication service, we employ the flit-level preemption method implemented by using virtual channels, which has similar structure as in [8], [18]. The arbitration with priority method uses priority preemption to provide delivery guarantees for hard deadline packets. We assume there are as many virtual channels as priority levels at each router's port. Each virtual channel is assigned a different priority. The traffic-flows loaded in the wormhole network have priorities associated with them. Each packet generated by a traffic-flow inherits the corresponding priority of the traffic-flow. A packet with priority i can only request the virtual channels associated with priority value i. At any time the packet with the highest priority always gets the privilege to access the output link. In addition, a higher priority packet can also preempt a lower priority packet during its transmission but not while an actual flit is being communicated. If the highest priority packet can not send data because it is blocked elsewhere in the network, the next highest priority packet can access the output link.

B. *Traffic-flow model*

A wormhole switching real-time network Γ comprises n real-time traffic-flows $\Gamma = \{\tau_1, \tau_2, \dots \tau_n\}$. Each traffic-flow τ_i is characterized by attributes $\tau_i = (P_i, C_i, T_i, D_i, J_i^R, J_i^I)$. We assume that all the traffic-flows which require timely delivery are periodic or sporadic. The lower bound interval on the time between releases of successive packets is called the period T_i for the traffic-flow τ_i. The maximum basic network latency C_i is the maximum transmission latency when there is no traffic-flow contention [10]. Each traffic-flow τ_i has a priority value P_i, $P_i > P_j$ means τ_i has higher priority than τ_j. The priority assignment for each traffic-flow is the main topic of this paper and the related details will be discussed in section IV. Each real-time traffic-flow has a deadline D_i constraint which means all the packets belonging to this traffic-flow have the restriction that they should be delivered from a source router to a destination router within this delay bound even in the worst case situation. Our model has the same restriction as [7], [9], [18] that each traffic-flow's deadline must be less than or equal to its period, $D_i \leq T_i$ for all $\tau_i \in \Gamma$. J_i^R is the release jitter [19] and denotes the maximum deviation of successive packets releases from its period. If a packet from τ_i is generated at time a, then it will be released for transmission by time $a + J_i^R$ and have an absolute deadline of $a + D_i$. J_i^I is interference jitter which denotes the maximum deviation of successive packets start service time (see section III-C2).

In addition, some dependency relationships also exist among the set of traffic-flows. Kim *et al* [18] introduced two kinds of interferences to deal with the relation between the traffic-flows, *direct interference* and *indirect interference*, and corresponding direct and indirect interference sets S_i^D and S_i^I of the observed traffic-flow τ_i, S_i^D and $S_i^I \in \Gamma$. The direct interference relation means the higher priority traffic-flow has at least one physical link in common with the observed traffic-flow. Thus, these traffic-flows will force a direct contention with the observed one. With the indirect interference relation, on the contrary, the two traffic-flows do not share any physical link but there is (are) intervening traffic-flow(s) between the given two traffic-flows. S_i^I includes all the higher priority traffic-flows that do not share any links with τ_i but share at least one link with a traffic-flow in S_i^D.

C. *Schedulablility analysis*

In this paper, we make use of the simple schedulability analysis approach for real-time communication in wormhole switching on-chip networks which is given by Shi and Burns [10]. The schedulability test is based on computation of the *worst case network latency* for each traffic-flow. If the worst case network latency R of a flow is no more than its deadline, $R_i \leq D_i$, then the traffic-flow is schedulable. If all the traffic-flows loaded on a network are schedulable, then the traffic-flow set is called schedulable.

The worst case network latency occurs when the following two conditions are met:

- All the traffic-flows release packets at their maximum rate and all the packets experience their maximum basic network latency.
- When the packet from the observed traffic-flow is released at the same time, all the higher priority packets finish waiting and start to receive service.

To determine the upper bound of network latency for a real-time traffic-flow, the maximum basic network latency and contention interference need to be computed. The maximum basic network latency can be calculated by static analysis of traffic-flow pattern and the network features. The interference, under the fixed priority preemption policy, is determined by all the higher priority traffic-flows and their resource competing relationships. Shi and Burns give an approach to quantify the analysis based on two distinguishing interferences: direct interference and indirect interference.

1) Direct higher priority interference: When only direct higher priority traffic-flows exist, the worst case network latency can be iteratively computed by using the following equation:

$$R_i = \sum_{\forall \tau_j \in S_i^D} \lceil \frac{R_i + J_j^R}{T_j} \rceil C_j + C_i \qquad (1)$$

where S_i^D is the set of direct higher priority traffic-flow of τ_i. We assume the packet from the observed traffic-flow is released simultaneously with all the packets from higher priority traffic-flows, this triggers the conditions of worst case network latency. The packet from τ_i may be blocked by more than one packet from each τ_j, $\tau_j \in S_i^D$, since the packet releases are periodic. The $\lceil \frac{R_i + J_j^R}{T_j} \rceil$ is the maximum number of packets a higher priority traffic-flow can release before τ_i completes.

2) Indirect higher priority interference: The observed traffic-flow may suffer interference when indirect higher priority traffic-flow exists, even when they do not share any physical link. The reason is that when the competition occurs between the indirect and direct higher priority packets, the latter will experience an unexpected deferral. This deferral between a packet being generated and being served is modelled as interference jitter. The interference jitter induces the practical minimum arrival interval between successive packet arrivals of a higher priority traffic-flow which is shorter than the original assumption T. This phenomenon will introduce an extra delay upon the observed traffic-flow. Thus, the worst case network latency for τ_i needs to take this extra delay into account when indirect interference exists. Eq.(2) denotes this relation (see [10] for this derivation):

$$R_i = \sum_{\forall \tau_j \in S_i^D} \lceil \frac{R_i + J_j^R + J_j^I}{T_j} \rceil C_j + C_i \tag{2}$$

We notice that this possible extra delay only occurs if $R_i + J_j^R + J_j^I > T_j$ and the amount is no more than C_j. The interference jitter of a traffic-flow can be obtained by finding the maximum deviation between two successive packets start service time. Consider the minimum and maximum of packet start service time are 0 and $R_j - C_j$, the upper bound of interference jitter is:

$$J_j^I = R_j - C_j \tag{3}$$

IV. Efficient Priority Assignment

Wormhole switching with the fixed priority preemption has been proposed as a solution for real-time on-chip communication. Therefore, an efficient priority assignment policy is desirable. In this section, we explore the possible method of priority assignment, such that this assignment ensures schedulability.

A. Properties of the model

Assuming that priorities have been assigned to each traffic-flow, the schedulability of a traffic-flow set can be determined by worst case network latency analysis shown in the last section. During the analysis process, we notice some special properties. These properties can help us reason about the effect of changing priority ordering of traffic-flows.

Corollary 1. *The network latency of traffic-flow is not dependent on lower priority traffic-flows.*

Theorem 1. *The network latency of a traffic-flow is dependent on the higher priority traffic-flows and their relative priority ordering.*

Proof: In priority-based preemption scheduling model, the observed traffic-flow τ_i suffers two different interferences, interference from direct higher priority traffic-flows and interference from indirect higher priority traffic-flows. For any traffic-flows τ_j meets the conditions: $\tau_j \in S_i^D$ and $P_j > P_i$, τ_j can force a direct competing interference. For any traffic-flows τ_k meets the conditions: $\tau_k \in S_i^I$ and $P_k > P_i$, τ_k can force an indirect competing interference. Thus, the interference suffered by the observed traffic-flow is determined by the set of all the higher priority traffic-flows. In addition, it is not hard to find if we swap two higher priority traffic-flows, the indirect interference relation may be changed. A typical case has been shown in Figure 1, with priority descending ordering τ_1, τ_2 and τ_3, τ_3 has direct competition with τ_2 and indirect competition with τ_1. However, if we swap priority of τ_1 and τ_2, τ_3 in this situation only suffers direct interference from τ_2. Thus, the priority ordering of higher priority traffic-flows also determines the network latency of lower priority traffic-flows. ■

Theorem 1 proves the priority ordering is the major factor in the analysis of worst case network latency for real-time on-chip communications. Obviously, the general priority assignment policies of Audsley [13], [14] which require that the response time is not dependent upon the higher priority ordering, are unfortunately not applicable in this case.

B. Lower and upper bounds of worst case network latency

There is no known efficient algorithm for finding a schedulable assignment of priorities that takes less than exponential time. Consider a set of n real-time traffic-flows, each traffic-flow is mapped to a unique priority level by some priority assignment policy. There are $n!$ distinct priority orderings over this set. Except for a very small size of traffic-flow set, we can not check all the possible orderings because of exponential time complexity. However, we notice an interesting property that not all the traffic-flows can meet their deadlines with the specific priority level. Let us revisit the case shown in Figure 1. If τ_2 is assigned the lowest priority, no matter what the ordering of τ_1 and τ_3, τ_2 is determinately not schedulable. If we can find these 'unsuitable' traffic-flows at the corresponding priority, we can reduce the search space significantly.

Two useful concepts: *lower bound* and *upper bound* of worst case network latency are introduced. The lower bound of worst case network latency $R_i^{'}$ is the network latency of a traffic-flow τ_i at some priority level when only direct interference is considered. On the contrary, the upper bound of worst case network latency R_i^* can be obtained for a traffic-flow τ_i when maximum interference jitter is considered. Actually, the real network latency can not be worked out before we know the priority assignment for all the traffic-flows (Theorem 1). But the possible range can be evaluated aforehand.

The proposed priority assignment algorithm starts from the lowest priority and moves up to highest priority. During the process of priority assignment, we use an assignment status to mark each traffic-flow, namely, *assigned* or *unassigned*. Thus, all the assigned traffic-flows have lower priorities and all the unassigned traffic-flows are assumed to be higher priorities than the assigned ones. When we try to assign priority k to an unassigned traffic-flow τ_i, the lower bound of network latency can be obtained by the following approach:

1) Except for the assigned traffic-flows, all the unassigned traffic-flows which share resource link with τ_i are inserted into set S_i^D.
2) Utilizing recurrence relation Eq.(4) to find the lower bound of worst case network latency.

$$R_i^{'} = \sum_{\forall \tau_j \in S_i^D} \lceil \frac{R_i^{'} + J_j^R}{T_j} \rceil C_j + C_i \tag{4}$$

Here, all the unassigned traffic-flows which have a resource competing relationship with τ_i are treated as direct interferences.

Additionally the upper bound of network latency also can be obtained by a similar approach:

1) Except for the assigned traffic-flows, all the unassigned traffic-flows which share resource link with τ_i are inserted into set S_i^D.
2) For any traffic-flow $\tau_j \in S_i^D$, if an unassigned traffic-flow τ_k is found which is in common with τ_j and not in common with τ_i, τ_j will produce an interference jitter. Here we consider the fact that the interference jitter J_j^I is no more than $D_j - C_j$ in the worst case, so $J_j^I \le D_j - C_j$ is true.
3) Utilizing the following relation Eq.(5) to find R_i^*

$$R_i^* = \sum_{\forall \tau_j \in S_i^D} \lceil \frac{R_i^* + J_j^R + D_j - C_j}{T_j} \rceil C_j + C_i \tag{5}$$

Here we evaluate the network latency based on worst case interference jitter scenario and obtain the upper bound of R_i.

Theorem 2. *The worst case network latency evaluated by Eq.(1) and Eq.(2) for τ_i is never less than its lower bound, $R_i \ge R_i^{'}$ is always true.*

Proof: For a set of n traffic-flows $\Gamma = \{\tau_1, \tau_2, \dots, \tau_n\}$ loaded on the network, we assume that priority level between $n, \dots, k+1$ have been assigned and k is the available lowest priority. Let priority k be assigned to τ_i . After that, all the unassigned traffic-flows will be allocated priorities between $k-1, \dots, 1$ in non-particular order, but have higher priority than τ_i. Thus all the unassigned traffic-flows which share resource link with τ_i will force a direct competition upon it; Eq.(1) shows this situation. Besides that, some unassigned traffic-flows also have relation of indirect interference with τ_i. This can force an extra 'hit' on τ_i. The worst case network latency can be achieved in this situation by Eq.(2) which takes both direct and indirect interferences into account. On the contrary, the lower bound of network latency only considers the direct interference. Thus, the network latency evaluated by Eq.(1) and Eq.(2) for τ_i is never less than its lower bound. ■

So for a flow τ_i, if its network latency lower bound is greater than its deadline, $R_i^{'} > D_i$, then $R_i > D_i$ is a reasonable conclusion. By evaluating the lower bound of network latency, Theorem 2 gives an approach to eliminate all the 'unsuitable' traffic-flows where $R_i^{'} > D_i$ at the specific priority. Next we will use the upper bound of worst case network latency to find the traffic-flow which is guaranteed for schedulablility at the specific priority.

Theorem 3. *If a traffic-flow τ_i is assigned the lowest priority and satisfies the condition $R_i^* \le D_i$, then if a schedulable priority ordering exists for Γ, a schedulable ordering also exists with τ_i assigned the lowest priority.*

Proof: Let us assume that a schedulable priority ordering exists over a traffic-flows set Γ with τ_i is assigned priority i. Since τ_i meets the condition $R_i^* \le D_i$ when it is assigned the lowest priority, so we move τ_i to the bottom of the priority ordering. The traffic-flows originally assigned priority $i+1$, $\dots$, n are promoted one level, Figure 2 shows this transformation process. Clearly, the traffic-flows assigned priority 1, $\dots$, $i-1$ remain schedulable as nothing has changed to affect their schedulability. The traffic-flows set which originally are assigned priority $i+1, \dots, n$ must remain schedulable because the interference on them has decreased due to τ_i now is moving to the lowest priority. Since τ_i is schedulable at the lowest priority level, a schedulable ordering exists with τ_i assigned the lowest priority.

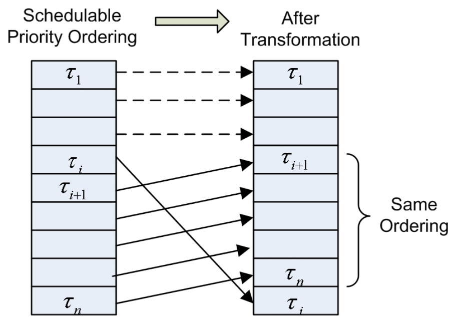

Fig. 2. Transformation of Priority Ordering

■

Theorem 3 implied that when a traffic-flow satisfies the condition $R_i^* \le D_i$ at the lowest priority, there must be a schedulable priority ordering with τ_i assigned the lowest priority no matter what the ordering of the rest of the traffic-flows. Each time when we find a traffic-flow meeting the above condition we can assign this flow the lowest priority. We can apply this process repeatedly and reduce the search space of priority assignment.

The next Theorem will tell us how to determine unschedulablility before a full priority assignment is given. Let $\Theta(n,k)^\Gamma$ denote a partial priority ordering from priority n to k for a set Γ with n traffic-flows. $\Theta(n,1)^\Gamma$ is a full priority ordering for n traffic-flows.

Theorem 4. *During the process of priority assignment, at the priority level k, a priority assignment policy has produced a partial priority levels ordering $\Theta(n, k-1)^{\Gamma}$ for a traffic-flows set Γ. If no unassigned traffic-flow meets the condition: $R_i^{'} \leq D_i$, $\tau_i \in \Gamma$, at the priority level k, then no schedulable priority ordering exists that assigns the same traffic-flows with priority n to k as $\Theta(n, k-1)^{\Gamma}$.*

Proof: For any priority assignment policy, each priority needs to map to a specific traffic-flow. During the process of priority assignment, at the priority level k, there are $k!$ distinct priority orderings exist. Consider a unassigned traffic-flow τ_i, there should be $(k-1)!$ priority orderings with τ_i at priority level k. If τ_i can not meet the condition $R_i^{'} \leq D_i$, no matter what the priority ordering of other unassinged traffic-flows, τ_i is un-schedulable at the priority level k according to Theorem 2. If all the unassigned traffic-flows can not meet the condition $R_j^{'} \leq D_j$ at the priority k, $\tau_j \in \Gamma$, all the $k!$ priority orderings are all un-schedulable. Then, no schedulable priority ordering exists that assigns the same traffic-flows with priority n to k as $\Theta(n, k-1)^{\Gamma}$. ∎

The properties of Theorem 2, Theorem 3 and Theorem 4 help to reduce the possible search space for finding the schedulable priority ordering. Therefore, we will use these properties as guidelines while designing an algorithm for priority assignment.

The intuition for the algorithm is as follow. At each priority level k, if any traffic-flow τ_j exists which $R_j^* \leq D_j$, then assign τ_j to level k; otherwise compute the set of traffic-flows with $R' \leq D$. If this set is empty, level k can not be assigned a flow. If the set is non-empty, choose one member τ_i using a heuristic and assign τ_i to level k. If this choice does not lead to a schedulable system, try the other members of the set. Repeat for all priority levels.

C. Heuristic Priority Assignment

1) Heuristic search algorithm: We give a heuristic search algorithm (HSA) to find a schedulable priority assignment if it exists. The search algorithm proceeds by performing a heuristic guided search to produce a possible priority ordering. When this priority ordering is completed, the schedulability analysis presented in section III-C is executed to test this ordering. If this priority ordering passes this test, we have found a schedulable priority assignment. If not, the algorithm backtracks to generate another one.

The priority search algorithm starts from the lowest priority. The upper bound of worst case network latency for each unassigned traffic-flow is evaluated at the current priority P (line 10), $P = n$ at the beginning. The unassigned traffic-flows which meeting the condition $R_i^* \leq D_i$ is assigned priority P (lines 4 - 8). At the specific priority, if no traffic-flow with the condition $R_i^* \leq D_i$ exists (line 9), the algorithm starts calculating the lower bound of worst case network latency (line 12). We construct n prioritized candidate lists *Priority_P_CandidateList*, $1 \leq P \leq n$, each list is related to a priority level P. Initially, each candidate list is empty. Theorem 2 implies only the traffic-flows which meet the condition $R_i' \leq D_i$ might be potentially schedulable at the specific priority. All the unassigned traffic-flows meet the condition $R_i' \leq D_i$ are inserted into corresponding priority level P candidate list (lines 2 - 3). After that, if priority P candidate list is not empty, an appropriate traffic-flow is selected from the candidate list to assign to priority P and it is removed from this list (lines 18 - 20). When more than one traffic-flow is available at priority P, to make the search more efficient, we select the "most promising" candidate first. A set of heuristic functions is proposed, the details of which are described below. The algorithm iterates this process at each priority until $P < 1$ (line 21) which means all the traffic-flows are assigned priority. Finally, the complete schedulable test is called.

```
procedure LowerBoundAnalysis($Priority\ P$)
 for each $unassigned\ \tau_i \in \Gamma$
    do { $R_i' \leftarrow LowBoundofNetworkLatency(\tau_i, P)$                  (1)
       { if $R_i' \leq D_i$                                                   (2)
       {   then $Insert(\tau_i, Priority\_P\_CandidateList)$                  (3)

procedure UpperBoundAnalysis($Priority\ P$)
 for each $unassigned\ \tau_i \in \Gamma$
    do { $R_i^* \leftarrow UpperBoundofNetworkLatency(\tau_i, P)$               (4)
       { if $R_i^* \leq D_i$                                                  (5)
       {   then { $assigned(\tau_i, P)$                                       (6)
       {        { $P \leftarrow P - 1$                                        (7)
       {        { return ( true )                                             (8)
       {   else return ( false )                                              (9)

main
 for $P \leftarrow n$ to 1
    do { if UpperBoundAnalysis($P$)                                           (10)
       {   then $Continue$                                                    (11)
       { LowerBoundAnalysis($P$)                                              (12)
       { while $Priority\_P\_CandidateList\ is\ EMPTY$                        (13)
       {   do { $P \leftarrow P + 1$                                          (14)
       {      { if $P > n$
       {      {   then return ($FAIL$)                                        (15)
       {      { $\tau_i \leftarrow GetTrafficFlowWithPriority(P)$             (16)
       {      { $unassigned(\tau_i)$                                          (17)
       { $\tau_i \leftarrow GetBestCandidate(Priority\_P\_CandidateList)$     (18)
       { comment: Select the traffic-flow with largest heuristic value next
       { $assigned(\tau_i, P)$                                                (19)
       { $P \leftarrow P - 1$                                                 (20)
       { if $P < 1$                                                           (21)
       {   then { if ($SchedulableTest() == $ true )                          (22)
       {        {   then return ($SUCCESS$)                                   (23)
       {        {   else { $P \leftarrow P + 1$                               (24)
       {        {        { $GoTo$ (13)                                        (25)
```

Fig. 3. Heuristic Search Algorithm

2) Heuristic Functions: The heuristic policy helps select the most likely candidate when more than one traffic-flow meets the condition $R' \leq D$. Our heuristic policy is based on an assumption of *tolerable additional interference*, namely, how much additional interference can be tolerated by a traffic-flow at a priority level without missing the deadline. This is a quite natural assumption, the more interference can be tolerated for a traffic-flow, the lower priority level is assigned. This policy leaves the chance for the traffic-flows which have less tolerable additional interference to get higher priorities. How to quantify the tolerable interference for each traffic-flow at a specific priority is an issue.

Here we propose two different baseline evaluation schemes. The first scheme $H1$ is the simple intuitive scheme which considers the absolute difference between D_i and R'_i as the tolerable interference for a traffic-flow τ_i. The traffic-flow with largest difference value is selected from the corresponding candidate list. Let H_i denote the heuristic value for τ_i.

H1 Heuristic Function

$$H_i = D_i - R'_i \tag{6}$$

The second baseline scheme $H2$ utilizes the sensitivity analysis technique [20] to evaluate the maximum tolerable additional interference. The sensitivity analysis can capture the bounds within which a parameter (basic network latency C in this case) can be varied without violating the timing constraints. Let $R'(C)$ denote the calculation function of lower bound of worst case network latency, $R'_i = R'(C_i)$. If a traffic-flow τ_i meets the condition $R'_i \leq D_i$, utilizing the sensitivity analysis, we can obtain maximum permissible increase of C_i without violating system schedulability. Let ΔC denote the tolerable increase.

H2 Heuristic Function

$$H_i = \max(\Delta C) \; where \; R'(C_i + \Delta C) \leq D_i \tag{7}$$

Besides this, we also consider the fact that the traffic-flow which travels a long distance likely encounters more blocking interference than the short distance. Thus the distance factor also should be taken into account. The following two heuristic functions are the improved versions of $H1$ and $H2$ which consider the number of hops a traffic-flow travels.

H3 Heuristic Function

$$H_i = (D_i - R'_i)/hops \tag{8}$$

H4 Heuristic Function

$$H_i = \max(\Delta C)/hops \; where \; R'(C_i + \Delta C) \leq D_i \tag{9}$$

The distance is inverse proportional to the tolerable additional interference so that the heuristic value calculated by $H3$ and $H4$ decreases when a traffic-flow traverses more hops. Accordingly, this traffic-flow has less possibility to be choosen at the lower priority level, which is as desired. When traffic-flow traverse a single hop, the heuristic value produced by $H3$ and $H4$ is equal to the result calculated by $H1$ and $H2$ respectively.

Also, we notice that for any given flow τ_i, all the higher priority traffic-flows in S_i^D will take a proportion of available network communication capacity $\sum_{\forall j \in S_i^D} C_j/T_j$. Thus, only a fraction of the network service capacity 1-$\sum_{\forall j \in S_i^D} C_j/T_j$ remains available to accommodate the current flow τ_i. The flow with higher free network service capacity is a better candidate to be served with lower priority. Thus the available network capacity is also modelled as the heuristic metric. The following two heuristic functions are the improved versions of $H1$ and $H2$ which consider network service capacity.

H5 Heuristic Function

$$H_i = \frac{D_i - R'_i}{\sum_{\forall j \in S_i^D} C_j/T_j} \tag{10}$$

H6 Heuristic Function

$$H_i = \max(\frac{\Delta C}{\sum_{\forall j \in S_i^D} C_j/T_j}) \; where \; R'(C_i + \Delta C) \leq D_i \tag{11}$$

3) Backtracking policy: It is not difficult to see that the search proceeds like a depth first search tree, each possible priority ordering is a branch of the search tree. However, the heuristic functions do not always select the right branch which induces a schedulable priority assignment. Thus, backtracking becomes an important part of the algorithm to help correct the search path. During the search iteration process, backtracking occurs at two possible places.

Middle unschedulable branch : Sometimes there is no traffic-flow satisfying the condition: $R'_i \leq D_i$ at the priority level P. In this situation, Theorem 4 implies that there is no schedulable priority ordering based on the traffic-flows which have been selected. The algorithm in this situation backtracks its search process by unassigning the selected traffic-flow at the priority $P + 1$ and tries to find another appropriate one, pseudo code lines 13 - 19 describe this process. Therefore, we prune some middle unschedulable branches to make the search more efficient.

End unschedulable branch: When a complete priority ordering has been found (line 21), the full schedulable test is called. If the priority ordering does not pass the schedulability test, the algorithm has to backtrack to find other appropriate priority orderings by following the same procedure to the middle unschedulable branch (lines 24 - 25).

Because of the restricted condition and pruning branch policy, only a small search space is produced compared with the original $n!$. The algorithm utilizes the backtracking policy to search all the possible priority orderings. Thus, if a schedulable priority ordering for a set of traffic-flow loaded on the network exists, our algorithm is assured to find this priority ordering.

V. Experiment Evaluation

This section describes experiments that have been conducted to quantify the performance of the heuristic priority assignment search algorithms, HSA. Consider the fact that a low dimensional mesh is quite common in current on-chip networks [1], [2], so we conduct our evaluation based on a 6×6 2D mesh. The dimension-order X-Y routing is used because it is simple and can be applied to any on-chip network without extra cost. Although we focus on the architectures inter-connected by 2D mesh networks with X-Y routing schemes, our algorithm can be adapted to other regular architectures with different networks topologies or different deterministic routing schemes.

Each traffic-flow is characterized by its period T, deadline D, basic communication latency C and transmitting path. Random source/destination nodes pairs are chosen for each traffic-flow. The basic network latency C is chosen from the range [16, 1024] time units with uniform probability distribution function. Another important metric period T is calculated by a random utilization variable u_i. We employ the uniform distribution algorithm [21] to generate a set of uniform distributed random utilization variable. Utilizing this utilization variable, each flow's period is calculated as $T_i = C_i/u_i$. The traffic-flow deadline D is set to be T for all the traffic-flows. For a set of generated traffic-flows, we ensure the flow set does not violate predefined constraints: maximum link utilization U_{max} or average link utilization U_{avg}. The link utilization U_{link_j} for a single link j can be found by summing the usages of all the m traffic-flows pass on this link.

$$U_{link_j} = \sum_{i=1}^{m} C_i/T_i \tag{12}$$

The maximum link utilization is the maximum number of all the links in this network, $U_{max} = max(U_{link_j})$ where $\forall link_j \in mesh$. The average link utilization is given by:

$$U_{avg} = (\sum_{j=1}^{M} U_{link_j})/M \tag{13}$$

where M is the total number of links in the network.

Except for the evaluation with the varied size of flows set, each time a generated test set contains 30 flows and each investigation level on the result diagrams is the average of 1000 randomly generated flow sets.

A. Evaluation of different heuristic functions

The efficiency of the HSA algorithm depends heavily on its heuristic search policy, so in this experiment, we examined the performance of the six different heuristic functions. During the heuristic search processing, if a promising candidate is found, it will be assigned the current priority and search processing moves to the higher priority level; otherwise, none of the current flows meeting the condition, the search algorithm backtracks to a lower priority level. So a reasonable metric to compare the efficiency of different heuristic functions is a measure of the number of priority assignments, namely, how many priority assignment operations are required to find a schdulable priority ordering. We design two groups experiments, first we compare these heuristic functions with the overall link utilizations variation. And then, we varied the number of traffic-flows from 40 to 100 with a fixed link load. All the experiment measures are taken under the maximum and average link load schemes separately.

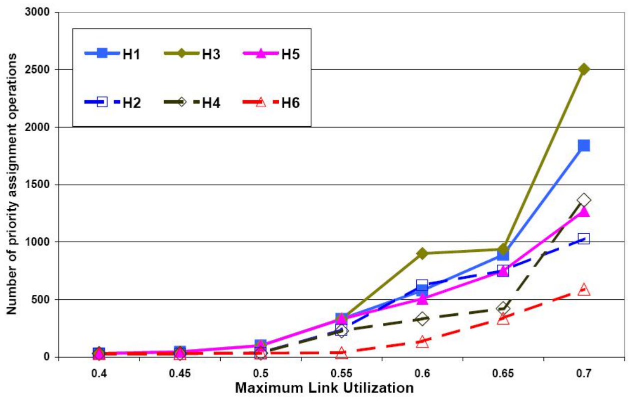

Fig. 4. The number of priority assignment operations under varied maximum link utilization

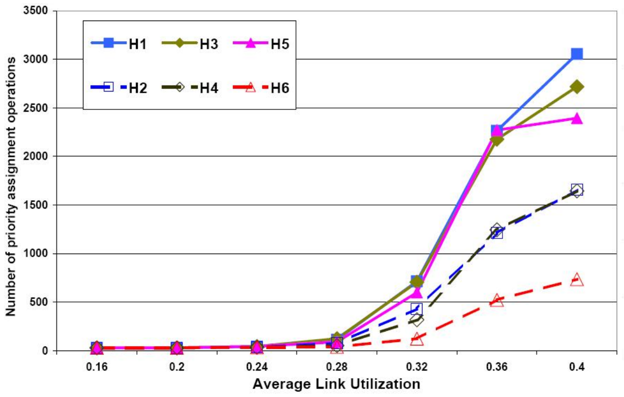

Fig. 5. The number of priority assignment operations under varied average link utilization

Figure 4 and Figure 5 show the average number of priority assignment operations for several heuristic functions under varied link load. As shown in both figures, all of the heuristic policies perform comparatively well and find a schedulable priority ordering quickly at the lower network load (no more than 0.28 in average load or 0.5 in maximum load). This is because in many cases, no matter what the priority ordering, the traffic-flow set is always schedulable. However, with the increase of network load, the iteration operation increases rapidly. The difference among these heuristic functions becomes clear. As can be seen in both figures, there is a significant reduction in average number of assignment operation for heuristic algorithms with the sensitivity analysis technique (H2, H4 and H6) than the intuitive ones (H1, H3, H5) which consider the absolute distance between deadline and worst case latency. The average operation numbers from H2, H4 and H6 sometimes are even no more than the half of the corresponding H1, H3 and H5. Two considered heuristic factors, distance and network service capacity, give comparable results. From the observation, the operation numbers of H5 and H6 are fairly reduced comparing with the baseline functions H1 and H2 when the network service capacity is considered. But the distance as heuristic factor (H3 and H4) does not present the more advantage over H1 and H2.

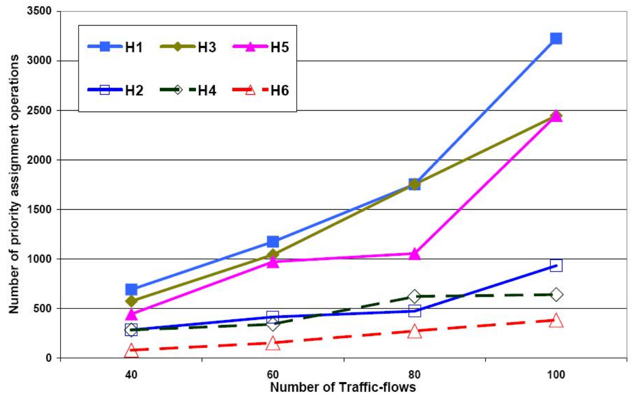

Fig. 6. The number of priority assignment operations with varied number of traffic-flows set under 55% maximum link utilization

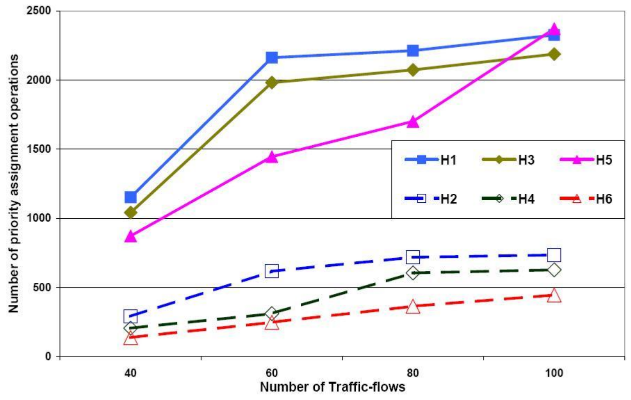

Fig. 7. The number of priority assignment operations with varied number of traffic-flows set under 32% average link utilization

Figure 6 and Figure 7 show the variation of priority assignment operations as the function of the size of traffic-flows set for fixed link utilization scenarios. With the increase of the flows set size, the gap between H1, H3, H5 and H2, H4, H6 becomes progressively larger. It is evident that the sensitivity analysis technique can achieve the better performance under the flow set with bigger size. Finally, in all cases, the experiment result shows that H6 scheme which combines with sensitivity analysis technique and network load capacity performs equally well and is highly effective in reducing the number of heuristic search operations.

B. Evaluation of different priority assignment schemes

This experiment investigated the performance of different priority assignment policies. Three existing schemes that are compared with HSA are : RM(T) [16], improvements of RM (RM(T/H) and RM(T/ln(e+H-1))) [8], [16]. The evaluation is aimed at the question "is it worth undertaking the heuristic scheme". The comparison measure used in this part is *pass ratio*, namely, the percentage of the number of traffic-flows set that pass the schedulability test versus the number of original traffic-flows set with different priority assignment policies. Only if all the traffic-flows in a generated set pass the schedulability test, do we say that the set is schedulable. H6 heuristic function is choosed as the default for the HSA. For evaluation reasons, we do not try to check all the possible branches but set a stoping criteria for the HSA with the maximum number of iterations. When the algorithm reaches the maximum iterations without finding a schedulable priority ordering for a set, this set is treated as unschedulable. In general, the more traffic-flow sets meet their timing constraints, the higher performance for the assignment policy.

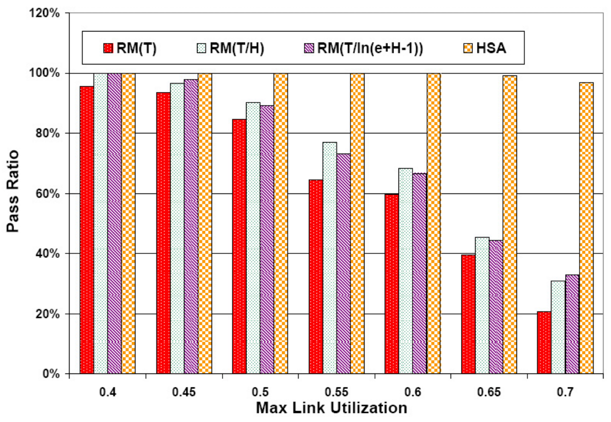

Fig. 8. The pass ratio under varied maximum link utilization

Due to limited space, only the evaluation results under varied maximum link load are shown. Similar trends can also be observed under average link load. Figure 8 plots, for each utilization level, the percentage of traffic-flow sets that were schedulable according to each priority assignment schemes. We find, as the link load increases, pass ratio decreases as expected. The heuristic search algorithm significantly outperforms the RM and its related schemes under higher network load. It still can find a schedulable priority assignment for a set even when the network maximum link load reaches 0.6, comparing with RM which only gets 59.8% pass ratio in Figure 8.

Figure 9 shows the pass ratio variation with the increase in the traffic-flows set size. HSA shows its robustness and achieves the highest pass ratio. In addition, the number of flow sets does not greatly affect HSA performance, and the pass ratio remains nearly constant. By comparison, RM and related policies perform poorly and their performances are sensitive to the size of flow set. When the number of flow set increases from 40 to 100, the average pass ratio falls almost 35% for RM related policies.

VI. Conclusion

Wormhole switching with fixed priority preemption has been proposed as a significant solution for real-time on-chip communications. One of the problems involved in the development

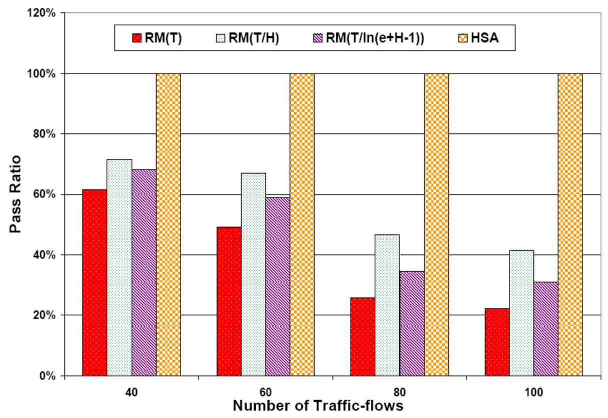

Fig. 9. The pass ratio with varied number of traffic-flows set under 55% maximum link utilization

of priority-based communication is: how to assign priority to each traffic-flow so that all the traffic-flows loaded on the network will meet their time constraints. In this paper, we introduce the heuristic branch and bound search algorithm to find the possible priority ordering. Differing from the traditional exhaust algorithm which costs exponential complexity, our algorithm can effectively reduce the search space. In addition, our algorithm can assure that if a priority ordering makes a set of traffic-flows schedulability, this priority ordering will be found by the search algorithm. By combining with schedulability analysis, a broad class of real-time communications with different QoS requirements can be explored and developed in a SoC/NoC communication platform.

References

[1] W. J. Dally, "Route packets, not wires: On-chip interconnection networks," *Proceedings of the 38th Design Automation Conference (DAC)*, pp. 684–689, 2001.

[2] L. Benini and G. D. Micheli, "Networks on Chips: A New SoC Paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[3] S. Furber and J. Bainbridge, "Future trends in SoC interconnect," in *IEEE International Symposium on VLSI Design, Automation and Test*, 2005, pp. 183– 186. [Online]. Available: http://www.imit.kth.se/ãxel/papers/2002/rvk-2002.pdf

[4] T. Bjerregaard and J. Spars, "Implementation of guaranteed services in the mango clockless network-on-chip," *IEE Proceedings: Computing and Digital Techniques*, vol. Vol. 153, (4), no. 217-229, 2006.

[5] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *Computer*, vol. 26, no. 2, pp. 62–76, 1993.

[6] N. Kavaldjiev and G. Smit, "A survey of efficient on-chip communications for SoC," in *4th PROGRESS Symp. on Embedded Systems*, 2003, p. 129140.

[7] S. Balakrishnan and F. Ozguner, "A priority-driven flow control mechanism for real-time traffic in multiprocessor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 7, pp. 664–678, 1998.

[8] S. L. Hary and F. Ozguner, "Feasibility test for real-time communication using wormhole routing," *IEE Proceedings - Computers and Digital Techniques*, vol. 144, no. 5, pp. 273–278, 1997. [Online]. Available: http://link.aip.org/link/?ICE/144/273/1

[9] Z. Lu, A. Jantsch, and I. Sander, "Feasibility analysis of messages for on-chip networks using wormhole routing," in *ASP-DAC '05: Proceedings of the 2005 conference on Asia South Pacific design automation*, 2005, pp. 960–964.

[10] Z. Shi and A. Burns, "Real-time communication analysis for on-chip networks with wormhole switching," in *Proceeding of the 2nd ACM/IEEE International Symposium on Networks-on-Chip(NoCS)*, 2008, pp. 161–170.

[11] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[12] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic real-time tasks," *Performance Evaluation*, vol. 2, pp. 237–250, 1982.

[13] N. C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," in *YCS164, Dept. Computer Science, University of York*, 1991. [Online]. Available: citeseer.ist.psu.edu/audsley91optimal.html

[14] N. C. Audsley, "On priority asignment in fixed priority scheduling," *Inf. Process. Lett.*, vol. 79, no. 1, pp. 39–44, 2001.

[15] A. Zuhily and A. Burns, "Optimality of (dj)-monotonic priority assignment," *Information Processing Letters*, vol. 103, no. 1, pp. 247–250, 2007.

[16] M. Mutka, "Using rate monotonic scheduling technology for real-timecommunications in a wormhole network," in *Proceeding of the Second Workshop on Parallel and Distributed Real-Time Systems*, 1994, pp. 194–199.

[17] J. Li and M. Mutka, "Real-time virtual channel flow control," *J. Parallel and Distributed Computing*, vol. 32, no. 1, pp. 49–65, 1996.

[18] B. Kim, J. Kim, S. J. Hong, and S. Lee, "A real-time communication method for wormhole switching networks," in *ICPP '98: Proceedings of the International Conference on Parallel Processing*, 1998, pp. 527–534.

[19] N. C. Audsley, A. Burns, M. Richardson, K. W. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, pp. 284–292, 1993. [Online]. Available: citeseer.ist.psu.edu/audsley93applying.html

[20] R. Racu, M. Jersak, and R. Ernst, "Applying sensitivity analysis in real-time distributed systems," in *RTAS '05: Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 160–169.

[21] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time System Journal*, vol. 30, no. 1-2, pp. 129–154, 2005.