Real-Time Communication Analysis with a Priority Share Policy 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, the hardware implementation cost is expensive and hence constrains its practical deployment. To address this problem, we propose a new solution by utilizing a priority share policy to reduce the resource overhead while still achieving the hard real-time service guarantees. The composite model-based schedulability analysis technique and relevant priority allocation scheme are represented in this paper. Experiment results show that significant resource saving can be achieved with no performance degradation in terms of missed deadlines. By using this approach, 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) [8], [3], have emerged as a new design paradigm to overcome the limitation of current bus-based communication infrastructure [9], and are increasingly important 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 [6]: a better wire utilization 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-core based designs 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 allowable arrival order of the packet set.

As a popular switching control technique, wormhole switching [16] has been widely applied for on-chip networks due to its greater throughput and smaller buffering requirement [11]. 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 [2], [10], [18] that allows the serving of traffic-flows with higher priorities before traffic-flows with lower priorities. The hard timing bound is delivered by this approach with the support of a priority arbitration infrastructure [18]. But the major problem of this priority-based approach is precisely that it requires distinct priorities and an exclusive virtual channel for each traffic-flow in a router port. This restricted implementation structure results in higher area and energy overhead and heavily limits its employment and development in real-time on-chip networks.

In this paper, we propose a solution that utilizes a *priority share policy* to reduce the resource overhead while still achieving hard real-time communication guarantee in wormhole switching on-chip networks. The priority share policy permits multiple traffic-flows to contend for a single virtual channel and share the same priority level. Considering the extra blocking delay introduced by this priority share, a novel schedulability analysis approach is presented which can effectively evaluate the timing properties of each real-time service. This approach models all the traffic-flows sharing the same priority as a single scheduling entity. Hence, the packet transmission is analyzed holistically and the computational complexity is kept low. Building on this static analysis, we furthermore propose a share priority allocation policy, in such a way that the timing requirements of all the traffic-flows are met with a reduced hardware overhead. By using

these approaches, we can flexibly explore, at design time, the quality of service (QoS) and system performance of a real-time SoC/NoC communication platform.

The rest of this paper is organized as follows: Section II reviews the related works and their limitations. In section III, the wormhole switching network with priority share policy and corresponding real-time communication model are introduced. A novel schedulability analysis technique and relevant priority allocation scheme are proposed in sections IV and V. By experiments based on enumeration, section VI presents the evaluation results of our approach. Finally, section VII concludes the paper.

II. Related works and their limitations

Real-time communication in wormhole switching networks has been studied widely. The early work to explore the packet's timing property was presented by Li and Mutka [13] in 1996. In that paper, the trade-off between the hardware costs and the performance under several different priority assignment and priority adjustment polices was addressed. However the upper bound of network latency for each packet in the network is not delivered by Li's method. Balakrishnan *et al* [2] proposed using *distinct priority per traffic-flow* to solve the real-time communication problem. Static priority preemptive policy assures at any time only the highest priority traffic-flow can access the link resources. Based on this priority arbitration model, a group of schedulability analysis approaches have been explored. Balakrishnan *et al* [2] and Hary *et al* [10] introduced the *lumped link* scheme which lumps all the links that the traffic-flow travels as one shared resource to analyze the schedulability problem. In [12], Kim *et al* used a blocking dependency graph to express the contentions a flow may meet and derived the packet delivery upper bound. The analysis by Lu *et al* [15] utilized a contention tree to take account of the parallel interference in disjoint traffic-flows and tried to improve the accuracy of analysis. Recently, one novel real-time schedulability analysis approach [18] has been proposed which successfully emulates wormhole switching as a classic single processor scheduling model [14].

However, all these works have the basis assumption of a distinct priority and exclusive virtual channel per traffic-flow. This should be reasonable for a general network in terms of cheap storage and computation resources. But the realistic concern in implementation cost and energy consumption of on-chip networks always means that the number of virtual channels supported by an interconnection infrastructure is kept as small as possible. To address this problem, we propose a priority share scheme by mapping multiple traffic-flows onto a single virtual channel. This optimization is simple but effective because the complexity of the crossbar and input buffers will be significantly reduced.

III. Real-time communication model in wormhole switching

A. Wormhole switching with priority share

Wormhole switching is a very popular *cut-through* switching strategy for on-chip networks [11]. The packets arriving at an intermediate node are immediately forwarded to the next node without significant buffering. Each packet in a wormhole network is divided into a number of fixed size flits [16]. 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 routers along the path and only a small flits buffer is required in each router.

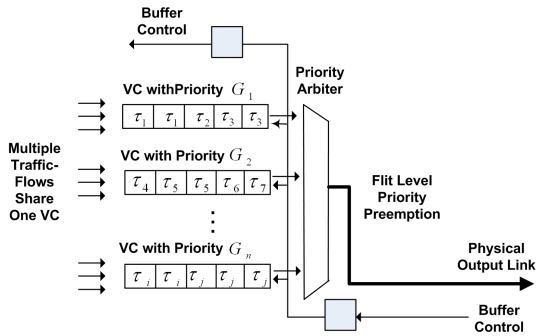

Fig. 1. Output Arbitration with Priority Share

In order to ensure hard real-time service guarantees with limited resource, we introduce a flit-level priority arbitration scheme, Figure 1 shows such a structure. There are a number of prioritized virtual channels [7] available at each router output port. The virtual channels (VCs) is a resource allocation technique which provides multiple independent buffers for each physical link. Each of these buffers is considered as a virtual channel and can hold one or more flits of a packet. A priority based arbitrator controls the access to the shared physical link for all the virtual channels. Since VCs are not mutually dependent on each other, the transmitting packet can bypass a blocked one through the different VCs. This strategy efficiently utilizes the network resource (link bandwidth) and improves the performance with a very small buffer overhead [5].

Differing from previous works [10], [12], [18], the distinctive characteristic of our scheme is that multiple traffic-flows per virtual channel [7] are supported. Assume each output port of a network router only supports limited distinct virtual channels and priority levels in terms of the restrictions of back-plane hardware. Multiple traffic-flows loaded in the wormhole network are allocated with the same virtual channel and thus are mapped to the same priority. Each packet generated by the traffic-flow inherits this priority. A packet with priority G_i can only request the virtual channel associated with priority G_i. With priority arbitration, 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 (flit level preemption). If the highest priority packet can not send data because it is blocked elsewhere in the network, the next highest priority packet wins the output arbitration.

B. Blocking Problem

The priority share scheme can dramatically reduce the hardware resource overhead compared with the original distinct priority per traffic-flow scheme [10], [12], [18], but on the other side, it may lead to significantly blocking and unpredictable network latency [19]. Consider the fact that traffic-flows within the same virtual channel are served in first-in-first-out (FIFO) order because priority preemption is only available between the different virtual channels. When a traffic-flow has to wait for the transmission of another traffic-flow in the same buffer due to priority share, blocking occurs. The blocking appears in real-time communication where the physical resource is not enough to handle the practical service requests. Therefore, a packet can be blocked by every packet with the same priority which arrives before it. Once a packet is blocked by another packet with the same priority which holds a virtual channel for a prolonged duration, it can block other packets, which can in turn block other packets, and so on.

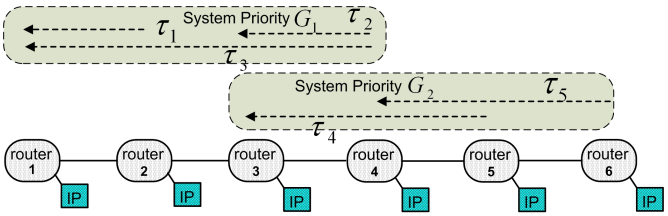

Fig. 2. A Case of Traffic-flows with Priority Share

As a simple example to motivate the blocking problem, consider Figure 2, which illustrates a number of traffic-flows loaded on a NoC platform. Flows τ_1, τ_2 and τ_3 share the same priority G_1, τ_4 and τ_5 share the same priority G_2 and G_1 is higher priority than G_2. We assume that a packet from τ_5 is released, because of priority share τ_5 can be blocked by τ_4 if it arrives just after τ_4. During τ_4's transmission, it can be preempted by the packet releases from higher priority flows τ_2 and τ_3. Note that when τ_2 or τ_3 is active, τ_4's packet service will be suspended but still occupy the link resources. In this situation, only after τ_4's completion, can the packet from τ_5 resume its transmission service. So the interference suffered by τ_4 actually extends the possible completion time of τ_5. Besides that, flow τ_1 in this case also can introduce some interference (see section IV-B) which delays the network latency of τ_5 further. Eventhough flows τ_1, τ_2 and τ_3 never share any physical link with τ_5, they still can play the major role in determining τ_5's transmission latency. This phenomenon does not exist in the distinct priority per flow policy. The exact analysis in this situation is very hard due to the complicated blocking inter-relationship between the flows.

To address schedulability under priority share, in this paper, we change the traditional analysis view point from per traffic-flow to per priority. The composite scheme is successfully proposed which collapses all the flows sharing the same priority level as a single scheduling entity. So a complicated blocking analysis is effectively avoided. The detailed issues will be discussed in the next section. First we will introduce the relevant model and assumptions.

C. Traffic-flow model

A wormhole switching real-time network Γ comprises n real-time traffic-flows $\Gamma = \{\tau_1, \tau_2, \ldots \tau_n\}$. Each traffic-flow τ_i is characterized by seven-tuple attributes $\tau_i = (P_i, G_i, C_i, T_i, D_i, J_i^R, J_i^I)$. 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 duration of transmission latency when no traffic-flow contention exists. Each real-time traffic-flow has relative deadline D_i 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 bound even in the worst case situation. Our model has the same restriction as [2], [12], [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 [1] and denotes the maximum deviation of successive packet 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 [18] which denotes the maximum deviation of successive packets start service time (see section III-E).

In contrast to previous models [2], [10], [12], [18], here we introduce two kinds of priority levels. Each traffic-flow τ_i has a *natural priority* P_i. The natural priority is produced relying on a distinct priority per traffic-flow policy. The value 1 denotes the highest priority and larger integers denote lower priorities. When the network supports priority share, besides natural priority, each traffic-flow is also assigned a *system priority* G_i which is also ordered from 1. All the traffic-flows competing for the same virtual channel will be mapped to the same system priority. Therefore, each system priority level G_i can be regarded as a flow set denoted by $S(i)$. In order to avoid run time complexity, here we require any flow if it is assigned a system priority at a router's port, it must have been assigned the same system priority at each router's port through its path. The natural priority and system priority are assigned off-line and remain constant at run-time. We also define two functions $P(\tau_i)$ and $G(\tau_i)$ to obtain the corresponding natural priority and system priority for a traffic-flow τ_i respectively. It follows that $\tau_i \in S(G(\tau_i))$.

D. Inter-relationships between traffic-flows

To capture the relations between traffic-flows and the physical links of the network, we formalize the mesh network

topology defined as a directed graph $\mathbb{G} : V \times E$. V is a set, whose elements are called nodes, each node v_i denotes one router in the mesh network. E is a set of ordered pairs of vertices, called edges. An edge $e_{x,y} = \{v_x \rightarrow v_y\}$ is considered to be a real physical link from router v_x to router v_y; v_x is called the source and v_y is called the destination. We define a mapping space from the traffic-flow set to the physical links $\Gamma \rightarrow E$. Given a set of n traffic-flows Γ, we can map them to the target network. The routing $\Re_i$ of each traffic-flows τ_i is denoted by the ordered pairs of edges, $\Re_i = \{e_{1,2}, e_{2,3}, \ldots, e_{n-1,n}\}$. If a traffic-flow τ_i shares at least one link with τ_j, the intersection set between them is $\Re_i \cap \Re_j$. If $\Re_i \cap \Re_j = \emptyset$, τ_i and τ_j are disjoint.

Based on whether they share the same physical links, we introduce two different relationships between the traffic-flows: *direct competing relationship*, and *indirect competing relationship*. The direct competing relationship means a traffic-flow has at least one physical link in common with the observed traffic-flow. For any two traffic-flows τ_i and τ_j, if direct competing relationship exists between them, then $\Re_i \cap \Re_j \neq \emptyset$. With the indirect competing relationship, by comparison, the two traffic-flows do not share any physical link but there is (are) intervening traffic-flow(s) between the given two traffic-flows. In another words, if there is an indirect competing relationship between τ_i and τ_k, then $\Re_k \cap \Re_i = \emptyset$ but $\Re_k \cap \Re_j \neq \emptyset$ and $\Re_i \cap \Re_j \neq \emptyset$; τ_j is the intervening flow in this case.

Assuming that natural and system priorities have been assigned to each traffic-flow, if a packet is released, the possible delays it suffered before completion consist of all the interferences from higher priority traffic-flows and the blocking from the traffic-flows with the same system priority. Based on different priority levels and the competing relationships, we categorize the delays into four different types:

- **Direct interference from traffic-flow with higher system priority**
  When two traffic-flows τ_i and τ_j have a direct competing relationship and meet the condition $G(\tau_j) > G(\tau_i)$, τ_j will force a direct contention with the observed traffic-flow τ_i. For flow τ_i, we define a set S_i^D which includes all the traffic-flows meeting the above conditions, S_i^D = $\{\tau_j | \Re_j \cap \Re_i \neq \emptyset$ *and* $G(\tau_j) > G(\tau_i)$ *for all* $\tau_j \in \Gamma\}$.
- **Indirect interference from traffic-flow with higher system priority**
  When two traffic-flows τ_i and τ_k have a indirect competing relationship and meet the condition $G(\tau_k) \geq G(\tau_j) > G(\tau_i)$, τ_j is the intervening traffic-flow, τ_i may suffer an indirect interference from τ_k even when they do not share any physical link, see [18] for detailed description. For flow τ_i, we define a set S_i^I which includes all the traffic-flows meeting the above conditions, S_i^I = $\{\tau_k | \Re_k \cap \Re_i = \emptyset$, $\Re_k \cap \Re_j \neq \emptyset$,$\Re_i \cap \Re_j \neq \emptyset$ *and* $G(\tau_k) \geq G(\tau_j) > G(\tau_i)$ *for all* $\tau_k \in \Gamma\}$.
- **Direct blocking from traffic-flow with same system priority**
  When two traffic-flows τ_i and τ_j have the direct competing relationship and meet the condition $G(\tau_j) = G(\tau_i)$, if the packet from τ_j is release just before τ_i, τ_j will force a blocking with τ_i. For flow τ_i, we define a set S_i^{SD} which includes all the traffic-flows meeting the above conditions, S_i^{SD} = $\{\tau_j | \Re_j \cap \Re_i \neq \emptyset$ *and* $G(\tau_j) = G(\tau_i)$ *for all* $\tau_j \in \Gamma\}$.
- **Indirect blocking from traffic-flow with same system priority**
  When two traffic-flows τ_i and τ_k have an indirect competing relationship and meet the conditions $G(\tau_k) = G(\tau_j) = G(\tau_i)$, where τ_j is a intervening flow, τ_i may suffer an indirect blocking from τ_k even they do not share any physical link. An indirect blocking example has been shown in Figure 2 where τ_1 blocks τ_3 and further blocks τ_2. For flow τ_i, we define a set S_i^{SI} which includes all the traffic-flows meeting the above conditions, S_i^{SI} = $\{\tau_k | \Re_k \cap \Re_i = \emptyset$, $\Re_k \cap \Re_j \neq \emptyset$,$\Re_i \cap \Re_j \neq \emptyset$ *and* $G(\tau_k) = G(\tau_j) = G(\tau_i)$ *for all* $\tau_k \in \Gamma\}$.

Note that the first two in the category correspond to the direct and indirect interferences in the original distinct priority per traffic-flow policy which have been presented in [18]. The last two are the new blockings introduced by the priority share policy, the related analysis will be addressed in the next section.

Returning to the example in Figure 2, five traffic-flows $\tau_1, \tau_2, \tau_3, \tau_4$ and τ_5 are mapped into two sets, the set with priority G_1 includes τ_1, τ_2 and τ_3, $S(1) = \{\tau_1, \tau_2, \tau_3\}$, the set with priority G_2 includes τ_4 and τ_5, $S(2) = \{\tau_4, \tau_5\}$; $G_1 > G_2$. Traffic-flows τ_1, τ_2 and τ_3 have no shared links with any higher system priority flow so no direct or direct interference, Due to sharing the same system priority, the direct and indirect blocking set for τ_1, τ_2 and τ_3 are $S_1^{SD} = \{\tau_3\}$, $S_1^{SI} = \{\tau_2\}$, $S_2^{SD} = \{\tau_3\}$, $S_2^{SI} = \{\tau_1\}$, $S_3^{SD} = \{\tau_1, \tau_2\}$ and $S_3^{SI} = \emptyset$. Flow τ_5 does not have any higher priority flow, so $S_5^D = S_5^I = \emptyset$. Flow τ_4 directly competes with τ_3 and τ_2 and indirect competes with τ_1 and hence $S_4^D = \{\tau_2, \tau_3\}$, $S_4^I = \{\tau_1\}$. Besides that, τ_4 and τ_5 share the same system priority level, thus $S_4^{SD} = \{\tau_5\}$, $S_4^{SI} = \emptyset$, $S_5^{SD} = \{\tau_4\}$ and $S_5^{SI} = \emptyset$.

E. Schedulability analysis with distinct priority per flow

In real-time wormhole switching networks, the schedulability analysis for the distinct priority per flow policy has been discussed by Shi and Burns [18]. The related idea and analysis strategy will be reused and extended in this paper. So, we utilize some space to review several important conclusions. The schedulability test is based on the 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, $R \leq D$, 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.

Generally, the second condition is called the *critical instant*. 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 calculated. The maximum basic network latency can be calculated by static analysis of the traffic-flow pattern and the network features. The interference, under the fixed priority preemption policy, is determined by the resources used by the higher priority traffic-flows. Shi and Burns [18] gave 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 [18]:

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

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 preempted by more than one packet from each τ_j, $\tau_j \in S_i^D$, since the packet releases are repeated. The term $\lceil \frac{R_i+J_j^R}{T_j} \rceil$ is the maximum number of packets of a higher priority traffic-flow that can release before τ_i completes.

2) Indirect higher priority interference: The observed traffic-flow may suffer interference when indirect higher priority traffic-flows exist. The reason is that when the competition occurs between 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 implies that the practical minimum arrival interval between successive packet arrivals of a higher priority traffic-flow 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 [18] for this derivation):

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

Note that not all the traffic-flows suffer interference jitter, this only happens when the observed traffic-flow τ_i has indirect interference , namely, J_j^I exists if and only if $S_j^D \cap S_i^I \neq \emptyset$. This possible extra delay 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 given by:

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

IV. Network latency upper bound analysis

One of the main challenging issues of NoC is how to use the minimum available on-chip resource while achieving considerable transmission quality of service. The on-chip resource overhead can be effectively reduced with a priority share policy. But this scheme inevitably introduces blocking which heavily complicates the analysis process. None of current schedulability analysis techniques [10], [12], [18] can handle this case because all these approaches assume there are sufficient priority levels so that blocking due to sharing virtual channels does not occur.

A. Properties of priority share

When multiple traffic-flows share the same priority with the $D \leq T$ constraint, we notice some special properties. These properties can help us reason about the effect of blocking and interference.

Theorem 1. *When a set of traffic-flows with $D \leq T$ and the same system priority are all schedulable, one flow can not block any other flow more than once.*

Proof: Let us assume a set of traffic-flow $\Gamma = \{\tau_1, \ldots, \tau_n\}$ with same priority level G_i. Suppose τ_a and τ_b are any two flows in set $S(i)$, $T_a \geq T_b$, and both meet the timing bound, the theorem is proved by the following two separated statements,

1) τ_b can not be blocked by τ_a more than once while still meeting its deadline.
2) τ_a can not be blocked by τ_b more than once while still meeting its deadline.

With the priority share policy, any flow can be blocked by other flows with the same priority just because of early arrival. Thus, the interference from higher priority and blocking from other same priority flows jointly determine the maximum latency of the current one. Suppose during any interval $[t_1, t_2]$, $t_1 \leq t_2$, the maximum interference suffered by a traffic-flow τ_i is $I_i(t_1, t_2)$. According to the property of priority share, if τ_b is released ahead of τ_a, the interference suffered by τ_b will be treated as blocking for τ_a, and vice versa.

With the condition $T_b \leq T_a$, statement (1) can be proved by contradiction. Let r_i denote the network latency of any flow τ_i. If τ_b is blocked by τ_a more than once, r_b must be greater than T_a, $r_b \geq T_a$. However, in order to meet the deadline, the relation $r_b \leq D_b \leq T_b \leq T_a$ must be hold, which contradicts with the above.

Statement (2) is proved by a similar contradiction. For τ_b, during the time interval [0, r_b], the total service requirement from τ_b, τ_a and all the other flows sharing the same system

priority is no more than $C_a+C_b+\sum_{\tau_i\in\{S(i)-\tau_a-\tau_b\}}\lceil\frac{r_b}{T_i}\rceil C_i$. The total service requirement from all the higher priority flows is no more than $\sum_{\tau_i\in S(i)} I_i(0,r_b)$. The network latency is hence derived by:

$$r_b = C_a+C_b+\sum_{\tau_i\in\{S(i)-\tau_a-\tau_b\}}\lceil\frac{r_b}{T_i}\rceil C_i+\sum_{\tau_i\in S(i)} I_i(0,r_b) \quad (4)$$

Since τ_b is schedulable with $r_b \le D_b \le T_b \le T_a$, and thus a value r_b where $r_b \le T_b$ must exist to fit Eq.(4).

On the other side, if τ_a is blocked by τ_b more than once, $r_a > T_b$ must be true. The time interval [0, r_a] can be subdivided into [0, T_b], and [T_b, r_a]. During [0, T_b], the service requirement must be greater than the network transmission capacity because until r_a, τ_a just completes its packet service. Thus, the following relation must be hold

$$C_a+C_b+\sum_{\tau_i\in(S(i)-\tau_a-\tau_b)}\lceil\frac{t'}{T_i}\rceil C_i+\sum_{\tau_i\in S(G_i)} I_i(0,t') > t' \quad (5)$$

where t' is any time instant during the time period [0, T_b]. But this contradicts Eq.(4) where a value no more than T_b can be found to fit this equation. Hence it is impossible that τ_b may block τ_a more than once while still meeting its deadline.

■

Theorem 2. *A necessary condition of schedulability for a set of traffic-flows with same priority is:* $\max(R_1, R_2, \ldots, R_n) \le \min(T_1, T_2, \ldots, T_n)$*, where* R_i *is the maximum network latency of* τ_i*.*

Proof: Suppose τ_i is a flow which has the maximum network latency in a set of traffic-flows sharing the same priority level, Theorem 1 proves that no traffic-flow can block another flow more than once while still meeting the deadline, hence, R_i must be no greater than any traffic-flow' period in that set to assure the schedulability, namely $\max(R_1, R_2, \ldots, R_n) \le \min(T_1, T_2, \ldots, T_n)$

■

B. Analysis with the composite model

The key issue of the schedulability analysis is how to efficiently calculate worst case network latency. The analysis technique [18] is based on per traffic-flow evaluation and tries to find the maximum delay for each observed flow. However, the analysis process becomes complicated when the priority share policy is supported. Both blocking and interference need to be taken into account. To solve this problem, here we propose a novel scheme - *composite traffic-flow model*. The composite model scheme tries to collapse all the flows sharing the same system priority as a single scheduling entity and hence, all the link resources required by the flows with the same priority level are treated as a single resource competing model. This new technique completely changes the analysis view point from "per-flow basis" to "per-priority basis". The motivation behind this approach is that by composite analysis, all the flows sharing the same priority will be scheduled as a holistic unit, so the complicated blocking analysis is effectively avoided and the computational complexity is kept sufficiently low.

Consider a set of traffic-flow $S(i)$ with the same system priority level G_i, the following steps are used to derive a composite task $\hat{\tau}_i$:

1) For the composite task $\hat{\tau}_i$, the maximum basic network latency is assigned equal to the summation of the all the flows in $S(i)$.
$$\hat{C}_i = \sum_{\forall\tau_j\in S(i)} C_j \quad (6)$$
2) Define the higher priority set $hp(i)$; all the flows in S_j^D are the members of set $hp(i)$,
$$hp(i) = \bigcup_{\forall\tau_j\in S(i)} S_j^D \quad (7)$$
where $\bigcup$ is the union operation of the flow sets.

In this composite model, all the original flows in $S(i)$ are integrated into a single transmission unit $\hat{C}_i$. Theorem 1 shows that no traffic-flow can be blocked more than once by the other flow with the same priority level. This actually implies the fact that the total service requirement from priority level G_i during [0, $max(R_1, R_2, \ldots, R_n)$] is no more than $\sum_{\tau_j\in S(i)} C_j$. We have showed in section III-B, due to blocking, any packet release from higher priority flow may extend lower priority flow completion time even they never share any link resource. Therefore all the original higher priority flows of each τ_j, $\tau_j \in S(i)$ are treated as the interference for this new composite flow. We assume the start service time for composite traffic-flow is released simultaneously with all the higher priority traffic-flows, this triggers the worst case network latency based on the condition of critical instant. Without loss of generality, we assume this time instant is at time 0. Until the time instant $\hat{R}_i$ when all the transmission service from priority level G_i complete, during the time interval [0, $\hat{R}_i$], the upper bound of interference produced by any direct higher system priority traffic-flow τ_j during this time interval is:

$$\lceil\frac{\hat{R}_i + J_j^R}{T_j}\rceil C_j \quad (8)$$

Note that, here we do not consider the interference jitter problem, but it does occur when indirect higher priority flows exist. This implies that the interference evaluated from Eq.(8) may be less than the real worst case scenario. The relative analysis and proof have been discussed in [18]. But the analysis in that paper only considers distinct priorities per traffic-flow. When the network supports priority share, the possible interference jitter comes from two potential situations.

Figure 3 illustrates these two situations which induce interference jitter. Flow τ_a shares the same system priority with τ_j in case 1 and higher system priority than τ_j in case 2. The transmission service from τ_a will force in τ_j an un-expected deferral. From the view of priority level-G_i flows, no matter what scenario occurs, the final result is similar: the indirect

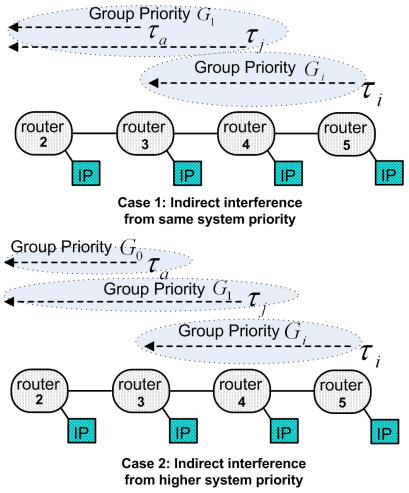

Fig. 3. Two possible situations for indirect interference

interference or blocking results in the real packet minimum arrival interval from direct higher priority traffic-flow τ_j is less than the original assumption T_j, this is modeled as interference jitter. So the upper bound of interference produced by τ_j when interference jitter [1] exists is:

$$\lceil \frac{\hat{R}_i + R_j - C_j + J_j^R}{T_j} \rceil C_j \tag{9}$$

where $R_j - C_j$ is the maximal possible jitter upper bound and R_j is the worst case latency of τ_j. Theorem 3 below will prove that the network latency of τ_j can be found by $\hat{R}_{G(\tau_j)}$. The interference jitter phenomenon only happens when a flow in $S(i)$ has indirect higher priority flow. Consider the fact that there are two possible indirect interferences scenarios when the network permits priority shares, either of which can result in the interference jitter, the corresponding conditions are $S_j^D \cap S_i^I \neq \emptyset$ or $S_j^{SD} \cap S_i^I \neq \emptyset$, where $\tau_i \in S(i)$ and $\tau_j \in S_i^D$.

As a result, an upper bound of network latency for the composite flow $\hat{\tau}_i$ in the case of interference jitter and release jitter is calculated as follows:

$$\hat{R}_i = \sum_{\forall \tau_j \in hp(i)} \lceil \frac{\hat{R}_i + R_j - C_j + J_j^R}{T_j} \rceil C_j + \hat{C}_i \tag{10}$$

The value of $\hat{R}_i$ can be found using the usual iterative technique [1]. The iteration starts with $\hat{R}_i = \hat{C}_i$ and terminates when $\hat{R}_i^{n+1} = \hat{R}_i^n$. By this iterative technique, the maximum latency can be calculated ($\hat{R}_i$=$\hat{R}_i^{n+1}$=$\hat{R}_i^n$).

Theorem 3. *The network latency upper bound R_i for τ_i in $S(i)$ is given by:*

$$R_i = \hat{R}_i + J_i^R \tag{11}$$

Proof: For any flow τ_i, $\tau_i \in S(i)$, the maximum delay occurs when it is released with all the higher priority flows simultaneously and all the other flows sharing the same priority as τ_i start their services just before τ_i, this will produce the maximum service requirements on the share resource. So the earliest starting time of τ_i is the same as $\hat{\tau}_i$. In the worst case, τ_i will be the last flow getting transmission opportunity in set $S(i)$. So the end of $\hat{\tau}_i$ is looked as the completion time of τ_i's packet instance. The network latency upper bound R_i for τ_i is hence given by Eq.(11). ■

Flow τ_i is schedulable, if and only if $R_i \leq D_i$. Notice that if each system priority G_i only contains one traffic-flow τ_i, $\hat{C}_i = C_i$ and $hp(i) = S_i^D$ in this case, Eq.(10) is degraded to Eq.(2). Actually, the distinct priority analysis approach [18] is only a specific case in our composite based analysis when each system priority is allocated to one traffic-flow only.

V. System Priority Assignment Policy

Wormhole switching with fixed priority arbitration has been proposed as a possible solution for real-time on-chip communication. However, the current approach emphasizing the independency between the traffic-flows introduces excessive virtual channel overheads, in such a way that area cost and energy consumption become unaffordable. Mapping more than one flow to a single virtual channel is a possible solution. The relevant blocking analysis and schedulability test have been represented in the previous section. However, this approach assumes that the system priorities have already been allocated to each traffic-flow. Consider that n traffic-flows are loaded on the network and there are only N system priorities available, where $N \leq n$, the total number of ways that n flows can be mapped into N priorities is $((n)!)/((N)!(n-N)!)$. This implies that it is impractical to determine the optimal solution by brute-force techniques, except for a very small size of traffic-flow set. Hence in this section, we design a greedy priority allocation policy which ensures schedulability with reduced time complexity.

Two important properties for priority based wormhole switching need to be clear before discussion of priority allocation:

Theorem 4. *[17] The network latency of a traffic-flow is not dependent on the lower priority traffic-flows.*

Theorem 5. *[17] The network latency of a traffic-flow is dependent on the higher priority traffic-flows and their relative priority ordering.*

It is obvious that the above two theorems still hold when a priority share policy is considered. Assuming that a schedulable priority ordering exists for a traffic-flow set under distinct priority per flow policy (how to find this priority ordering please reference [17]), now we give an assignment policy to find a corresponding flow ordering when the network supports priority share.

The intuition for the algorithm is as follow: at each system priority G_k, if any traffic-flow τ_i exists that when τ_i is mapped to priority G_k, all the flows which have been assigned system priority G_k or less are still schedulable, τ_i will be assigned

[1]The detailed analysis and calculation about interference jitter please reference to paper [18].

priority G_k. If no additional flow mapped to G_k can lead to a schedulable system, the system priority is increased.

Algorithm V.1: PRIORITY ASSIGNMENT ALGORITHM ()

```
procedure SCHEDULABILITYTEST(G)
 for G_i ← G to G_n
   do { R̂_i ← CompositeNetworkLatency(τ̂_i)
        for each τ_i ∈ S(G_i)
          do { R_i ← R̂_i + J_i^R
               if R_i ≥ D_i
                 then return ( false ) }
 return ( true )

procedure GETFLOWWITHOUTCHECKED(G, Γ')
 if (Policy 1)
   then return (LowestPriorityFlow(Γ'))
 if (Policy 2)
   then { if (S(G) is EMPTY)
            then return (LowestPriorityFlow(Γ'))
          else { MaxReFlow ← NULL
                 MaxRe ← 0
                 for each unchecked τ_j ∈ Γ'
                   do { if (MaxRe < ShareRe(τ_j, S(G)))
                          then MaxReFlow ← τ_j
                 return (MaxReFlow) }

main
 G ← G_n
 while Γ' is not EMPTY
   do { SetAllUnChecked(Γ')
        repeat
          { τ_i ← GetFlowWithoutChecked(G, Γ')
            TryPriorityAssignment(τ_i, G)
            if SCHEDULABILITYTEST(G)
              then { Assigned(τ_i, G)
                     Move(τ_i, Γ'') }
              else FreePriorityAssignment(τ_i)
            SetChecked(τ_i) }
        until AllChecked(Γ')
        G ← G − 1 }
```

Algorithm V.1 shows the process of system priority allocation. The algorithm maintains two flow sets: Γ' contains flows which are not yet assigned their system priority and Γ'' contains flows with priority assigned. Initially, all the flows are inserted in set Γ', and Γ'' is empty. The priority assignment algorithm starts from the lowest system priority. At each priority level, the algorithm firstly needs to find a 'promising' flow τ_i in Γ'. There we introduce two different selection policies to find the right flow in set Γ', the details of which are described below. Then τ_i is assigned with the current system priority G, $G = G_n$ at the beginning, and is checked with the schedulability test. Theorem 5 implies that all the flows with current priority G or lower need to be checked since the network latency may change due to the possible blocking and interference jitter. If τ_i and all the flows in Γ'' are all schedulable, we can assign the G to τ_i and move it to Γ''. If τ_i can not be assigned to G, we do the same check for the next candidate flow in Γ'. During the process of schedulability check, we use a check status to mark each traffic-flow, namely, *checked* or *unchecked*. This ensures that each flow only can be checked once during a specific priority allocation. We repeat the procedure for all the flows in Γ' to find that any flows can be mapped to G. If none of the flows in Γ' can be assigned to the current system priority G, then G is increased and the algorithm repeats the above processing.

When considering the next candidate flow to allocate system priority, all the flows in Γ' should be checked in turn. To make the priority assignment more efficient, here two greedy selection policies help find the most likely candidate when more than one flow is available in set Γ'.

- The first policy always returns the lowest natural priority flow in Γ' at each selection process.
- The second policy selects the flow which shares the maximum network resource with all the flows at current priority level in Γ''. If there is no flow at current priority level, the algorithm automatically returns the lowest natural priority one.

Theorem 6. *If a schedulable priority ordering exists under distinct priority per flow policy, there must exist a schedulable priority ordering under priority share.*

Proof: At each system priority assignment process, the algorithm V.1 ensures that all the flows moved in set Γ'' are schedulable. In addition, we find that all the flows in set Γ' still remain schedulable, because the interference on them has decreased or at least unchanged since some higher priority flows have been moved to set Γ''. So after each step of system priority assignment, the whole flow set remains schedulable. The priority assignment algorithm will stop when all the flows are allocated system priority. Thus, the algorithm guarantees that if a schedulable priority ordering exists under distinct priority per flow policy, a corresponding schedulable priority ordering under priority share can be found. ■

In the worst case, the number of system priority levels required is equal to the natural priority level which means that no traffic-flows can share the same priority. The greedy algorithm includes inner and outer loops and thus its time complexity is $O(N^2)$.

VI. EXPERIMENT EVALUATION

This section presents our experimental evaluation results concerning the resource saving improvement of the priority share approach. The traditional distinct priority per traffic-flow approach [18] is selected as the benchmark. We are concerned with evaluating the reduction in the number of priorities required and the subsequence reduction in the number of virtual channels. Considering the fact that a low dimensional mesh is quite common in current on-chip networks [8], [3], we conduct our evaluation based on a 4×4 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 architectures inter-connected by 2D mesh networks with X-Y routing schemes, our algorithm can be adapted to other regular architectures with different network topologies or different deterministic routing schemes.

Each traffic-flow is characterized by its period T, deadline D, basic communication latency C and transmitting path.

A random source/destination node pair is chosen for each traffic-flow. The basic network latency C is chosen from the range [16, 1024] time units with a uniform probability distribution function. Another important metric period T is calculated by a random utilization variable u_i. We employ the uniform distribution algorithm [4] to generate a set of uniform distributed random utilization variables. Employing 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 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.

The major implementation overhead in supporting priority based wormhole switching includes the total priority levels and the amount of virtual channels. So we use the ratio of required resource cost between priority share policy and the distinct priority per traffic-flow policy as a performance metric. Two groups of experiments are designed to compare the different approaches: first we make the evaluation while varying the overall link utilization. And then, we vary the number of traffic-flow sets from 40 to 100 with a fixed link load. All the experiment measures are taken under the maximum and average link load schemes separately. Except for the evaluation with the varied size of flow set, each generated test set contains 30 flows and each investigation level on the resulting diagrams is the average of 1000 randomly generated flow sets. In general, the less resources are used, the more effective for the implementation approach.

Figure 4 and Figure 5 show the ratio of resource cost (both priority levels and virtual channels) under each utilization level. As expected, the priority share policy significantly outperforms the traditional approach under all the link load situations. Especially at lower link utilization, the priority share scheme (policy 2) exhibits a remarkable hardware cost saving; consuming only 20.3% of priority levels and 38.4% of virtual channels compared with the original approach when the network maximum link load reaches 0.4 in Figure 4.

Figure 6 and Figure 7 show the variation of the rate of resource cost as a function of the size of the traffic-flow set for a fixed link utilization. With the increase of the flow set size, the difference between the priority share policy and the distinct priority per traffic-flow policy becomes progressively larger. Our approach shows its power and achieves better resource saving as the flow set size increases. This is because when

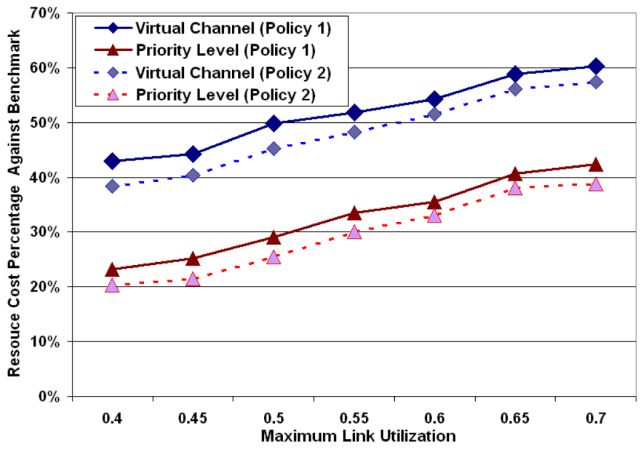

Fig. 4. The ratio of resource cost under varied maximum link utilization

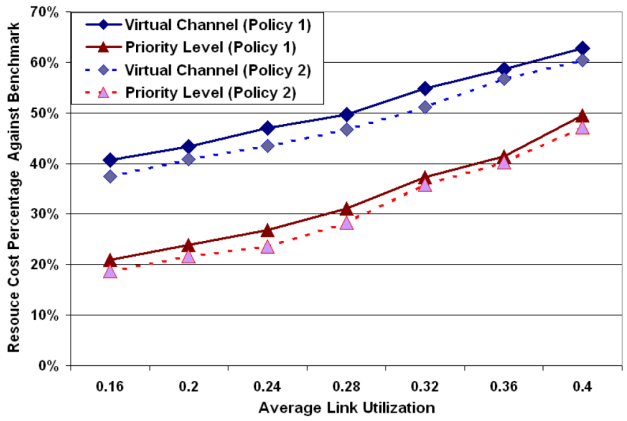

Fig. 5. The ratio of resource cost under varied average link utilization

more and more traffic-flows compete for the same physical link, there is more opportunity for multiple flows to share the virtual channel and priority level and hence decrease the hardware overhead. In all cases, the experiment shows the second selection policy which returns the flow with the maximum share resource performs better than the first policy.

VII. Conclusion

The new on-chip communication architectures need to provide different levels of service for various components on the same network. Wormhole switching with fixed priority preemption has been proposed as a possible solution for real-time on-chip communication. However, the hardware implementation costs always becomes the burden to restrain practical deployment. In this paper we explore real-time communication services with a priority share policy. A novel schedulability approach and relevant priority allocation policy are represented, in such a way that the deadlines of all the traffic-flows are still met with reduced resource overhead. The experimental results show that, on average, the number of virtual channels and priorities can be reduced by 50% and 70% respectively. Reducing the number of virtual channels also decreases the complexity of the switch because there needs to be fewer

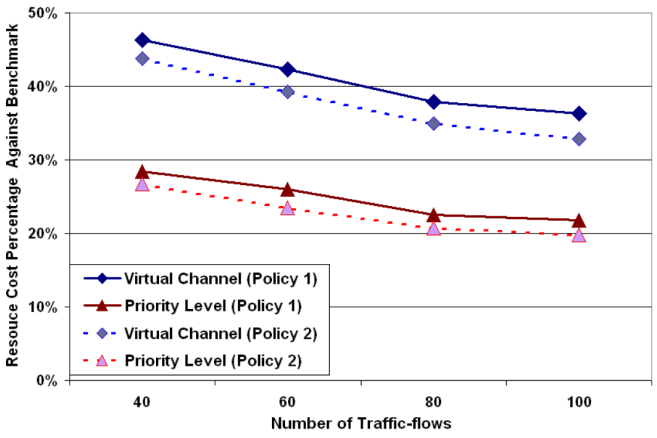

Fig. 6. The ratio of resource cost with varied number of traffic-flow sets under 0.55 maximum link utilization

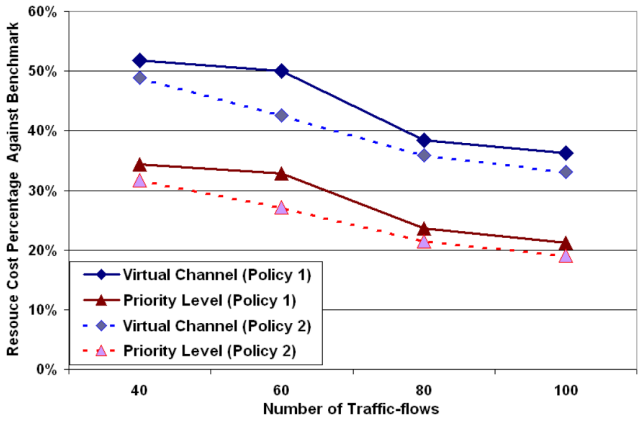

Fig. 7. The ratio of resource cost with varied number of traffic-flow sets under 0.32 average link utilization

crossbars between input and output buffers. By using these approaches, a broad class of real-time communication with different QoS requirements can be explored and developed in a SoC/NoC communication platform.

REFERENCES

[1] 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*, 8:284–292, 1993.

[2] S. Balakrishnan and F. Ozguner. A priority-driven flow control mechanism for real-time traffic in multiprocessor networks. *IEEE Trans. Parallel Distrib. Syst.*, 9(7):664–678, 1998.

[3] L. Benini and G. D. Micheli. Networks on Chips: A New SoC Paradigm. *Computer*, 35(1):70–78, 2002.

[4] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time System Journal*, 30(1-2):129–154, 2005.

[5] T. Bjerregaard and S. Mahadevan. A survey of research and practices of network-on-chip. *ACM Computer Survey*, 38(1):1, 2006.

[6] T. Bjerregaard and J. Spars. Implementation of guaranteed services in the mango clockless network-on-chip. *IEE Proceedings: Computing and Digital Techniques*, Vol. 153, (4)(217-229), 2006.

[7] W. J. Dally. Virtual-channel flow control. *IEEE Trans. Parallel Distrib. Syst.*, 3(2):194–205, 1992.

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

[9] S. Furber and J. Bainbridge. Future trends in SoC interconnect. In *IEEE International Symposium on VLSI Design, Automation and Test*, pages 183–186, 2005.

[10] S. L. Hary and F. Ozguner. Feasibility test for real-time communication using wormhole routing. *IEE Proceedings - Computers and Digital Techniques*, 144(5):273–278, 1997.

[11] N. Kavaldjiev and G.J.M. Smit. A survey of efficient on-chip communications for SoC. In *4th PROGRESS Symp. on Embedded Systems*, pages 129–140, 2003.

[12] 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*, pages 527–534, 1998.

[13] J.P. Li and M.W. Mutka. Real-time virtual channel flow control. *J. Parallel and Distributed Computing*, 32(1):49–65, 1996.

[14] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[15] 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*, pages 960–964, 2005.

[16] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, 1993.

[17] Z. Shi and A. Burns. Priority assignment for real-time wormhole communication in on-chip networks. In *Proceeding of the 29th IEEE Real Time System Symposium (RTSS)*, pages 421–430, 2008.

[18] 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)*, pages 161–170, 2008.

[19] H. Song, B. Kwon, and H. Yoon. Throttle and preempt: A new flow control for real-time communications in wormhole networks. In *ICPP '97: Proceedings of the international Conference on Parallel Processing*, pages 198–202, 1997.