Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching

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

Abstract—**In this paper, we discuss a real-time on-chip communication service with a priority-based wormhole switching policy. A novel off-line schedulability analysis approach is presented. By evaluating diverse inter-relationships among the traffic-flows, this approach can predict the packet network latency based on two quantifiable different delays: direct interference from higher priority traffic-flows and indirect interference from other higher priority traffic-flows. Due to the inevitable existence of parallel interference, we prove that the general problem of determining the exact schedulability of real-time traffic-flow over the on-chip network is NP-hard. However the results presented do form an upper bound. In addition, an error in a previous published scheduling approach is illustrated and remedied. Utilizing this analysis scheme, we can flexibly evaluate at design time the schedulability of a set of traffic-flows with different QoS requirements on a real-time SoC/NoC communication platform.**

I. Introduction

With the development of semiconductor technology over the last fifteen year, it is possible to offer more than many tens of million of transistors on a single chip. Under this condition, designers are developing ICs integrating complex heterogeneous functional elements into a single device, known as a System-on-Chip (SoC).

Generally, SoC is an integrated circuit that implements most or all of the functions of a complete electronic system. In such a system, different components need a standard approach to support on-chip communication. Early SoCs employed busses or a point-to-point approach to fulfil the information exchange demands. However, with the rapid increase in the number of blocks to be connected and the increase in performance demands, busses and point-to-point based platforms suffer from limited scalability and quickly become a communication bottleneck [9], [11]. On-chip packet-switched networks have recently been proposed as a significant solution for complex communication of SoCs. Network on Chip (NoC) [7], [4] is an architectural paradigm for scalable on-chip interconnection architectures. This architecture offers a general and fixed communication platform which can be reused for a large number of SoC designs.

Networks as a subject has been studied for decades. However, the situation for NoC is different from off-chip networks meaning that we can not deploy general networks on SoC platforms directly. NoCs differ from off-chip networks mainly in that they are more constrained and less non-deterministic [3]. The structure of an off-chip network or general network is in principle unknown, ie. topology is not fixed and behaviours of nodes are not predictable. So the protocols need enough adaptability to meet various requirements. Some results used in traditional networks can't be employed directly and must be re-evaluated. A few distinctive limitations are unique for on-chip networks, namely, minimal energy consumption, and small size [12] (both computation and storage functions implemented in a small silicon area). Therefore many network design choices need to be modified so that the implementation cost as well as speed/throughput performance is acceptable.

The new on-chip communication architecture needs to provide different levels of service for various application components on the same network. One kind of communication, namely real-time communication, has very stringent requirements, the correctness relies on not only the communication result but also the completion time bound. For a packet transmitted over the network, this time bound is denoted by the *packet network latency.* A data packet received by a destination too late could be useless. For instance, the signal message packet or control message packet of an application requires timely delivery. The worst case acceptable time metric 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 [19] has been widely applied for on-chip networks due to its greater throughput and smaller buffering requirement [12]. However, few works have been done to analyze the real-time packet schedulablility for wormhole switching networks. In order to support real-time requirements, particularly satisfying its deadline bound, predictable behaviour of the network service is essential. But the situation for on-chip wormhole networks is partially non-deterministic due to the contentions in communication. In on-chip networks, several tasks running on different nodes exchange information periodically. During a transmission period, one transmitted packet shares the resources, such as buffers or physical links, with other packets. When several packets try to access the same resource at the same time, contention occurs and the network only can serve one packet and suspend the others based on some arbitration policy. Once a packet becomes blocked, it can block other packets, which can in turn block other packets, and so on. The exact analysis of congestion in this situation is hard [2] due to the possibility of a packet becoming blocked at several routers during its journey from source to destination. The contention problem leads to packet delays and even missed deadlines. Therefore, it is necessary to give a scheduling strategy and analysis approach to predict whether all the real-time packets can meet their timing properties.

In this paper we explore real-time communication in wormhole switching for on-chip networks. We assume the priority-based transmission preemption method [2], [10], [13]. A novel analyzable approach, the worst case network latency evaluation, is presented, within which a broad class of real-time communication services can be explored and developed. By evaluating diverse inter-relationships and service attributes among the traffic-flows, our model 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. Due to the inevitable existence of parallel interference, we prove that the general problem of determining the exact schedulability of real-time traffic-flows over the on-chip network is NP-hard. We also prove that the real maximum network latency is bounded by the theoretical calculation in our model. By using this approach, we can flexibly evaluate at design time the schedulability of a traffic-flow set with different quality of service (QoS) requirements in a real-time SoC/NoC communication platform.

The rest of this paper is organized as follows: section II introduces the major features of wormhole switching. A preemptive arbitration structure is deployed to implement priority-driven transmission scheduling. A novel real-time communication model and associated analysis are represented in sections III and IV. Section V discusses the limitation of our model when there exists parallel interference. Finally, section VI concludes the paper.

II. Wormhole switching in real-time communication

A. Wormhole switching

It is a major challenge for NoCs to provide real-time support. The area and energy constraints determine that the cut-through switching approach (wormhole switching) is the more practical deployment strategy than the store-and-forward switching (packet switching) policy [8], [12]. In a wormhole switching network, data is encapsulated into a packet format for network transmission. So for convenience in discussing, a packet is treated as the basic information unit throughout this paper. Each packet in a wormhole network is divided into a number of fixed size flits [19]. 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. But this blocking will decrease the available resource for other communication traffic-flows and reduce the network resource efficiency.

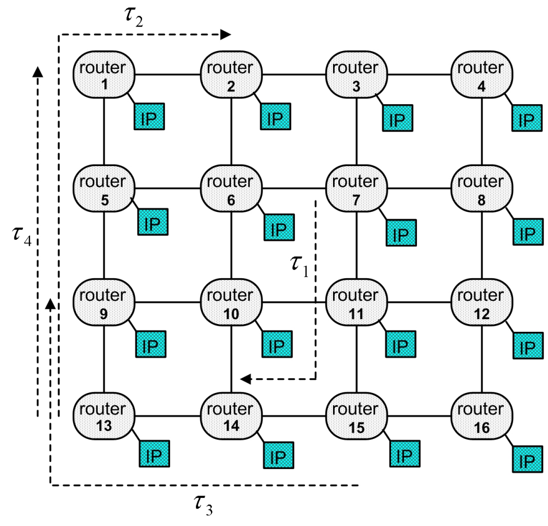

Fig. 1. A General NoC Platform

A general communication infrastructure for the wormhole on-chip network is illustrated in Figure 1. A set of routers and point-to-point links interconnecting the routers are organized in a mesh structure. Each router has one or several *intellectual property (IP)* modules which hold tasks for execution. These tasks, executing on different IPs, communicate with each other by transmitting packets through the on-chip interconnection network. Two unidirectional links, one for each direction, connecting two routers realize the full-duplex transmission media. The network uses dimension-order X-Y routing, which is simple and easy to be implemented in the regular topology. The virtual channels (VCs) technique [6] is deployed which decouples resource allocation by providing multiple buffers for each physical link in the network. Each of these buffers is considered as a virtual channel and can hold one or more flits of a packet. By combining with the virtual channels technique, the transmitting packet can bypass a blocked one. This strategy efficiently utilizes the network resource (link bandwidth) and improves the performance with a very small buffer overhead [5].

Figure 1 also illustrates a number of traffic-flows loaded on this NoC platform. For example, τ_1 starts in router 7 and passes through routers 11 and 15 before terminating in router 14.

B. Priority preemptive arbitration

In conventional wormhole routers, data flits held by VCs access the output link based on first-come first-service arbitration. This scheme is suitable for non-real-time networks since it is fair and produces good average performance. But in real-time communication, the network must ensure each real-time packet meets its deadline bound. Priority arbitration is proposed to resolve this problem [2], [10], [13]. We employ the flit-level preemption method implemented by using virtual channels, which have similar structure as in [10], [13]. 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 output port. Each virtual channel is assigned a different priority. An output port structure of a router is shown in Figure 2. 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. Since the real-time traffic-flows between different routers in a specific on-chip network is known a prior, a global priority assignment policy should be ameanable to off-line analysis [2].

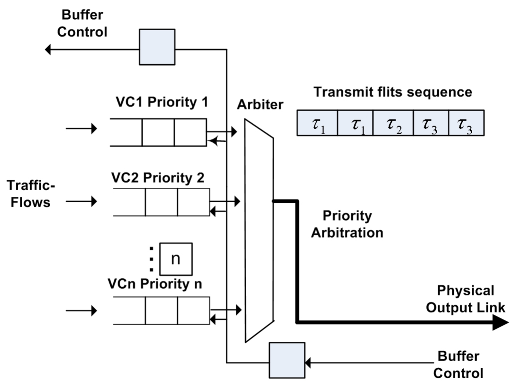

Fig. 2. Priority Arbitration Output Port

Consider n traffic-flows, each one with an associated virtual channel containing flits. The arbiter fetches flits from these queues according to priority arbitration and forwards them over a shared output link. 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. The allowable service time for a traffic-flow is all the time intervals at which no higher priority traffic-flow competes for the same physical link.

C. Current works

Wormhole switching achieves high throughput performance with less buffer requirement comparing with packet-switched technique [8], [12]. But it also introduces unpredictable network delay [8]. Hard real-time communication, on the contrary, requires the timing to be predictable even under the worst case situation. Besides this, the network resource allocation and scheduling for the real-time traffic-flow should be analysable during the design phase. Predictability of performance is essential for NoCs design to take early decisions before actual implementation. Fortunately, the communication pattern of a SoC is determined during a pre-configuration period; interconnection topology and characteristics of traffic pattern are foreseeable. Therefore, we need an off-line static evaluation approach to ensure the packet network latency never violates its timing bound. Utilizing this approach, we also can plan and explore the distribution of the real-time applications over the network to produce a very effective mapping.

The first work to explore the packet's timing property in wormhole switching was published by Li and Mutka [16] in 1994. Utilizing the priority strategy, for a wormhole network with the same number of virtual channels as the number of priority levels, a packet can request only a virtual channel which is numbered lower than or equal to its priority [16]. Song *et al* [20] proposed a flow control approach to avoid the priority inversion problem. By flit-level preemption, the different priority traffic-flows can be catered for by a very small number of virtual channels. However the upper bound of network latency for each packet in the network are not delivered by Song and Li's methods. Balakrishnan *et al* [2] proposed a quite naive and simple approach - *lumped link* to address this problem. All the links the traffic-flow travels are lumped as one shared resource - like a bus structure. Static priority preemptive policy is adopted to assure at any time only the highest priority traffic-flow can access the link resources. However, due to lumping, direct and indirect contentions are treated in the same way, Balakrishnan's result is sufficient but pessimistic [18]. Hary *et al* [10] utilized the same model proposed in [2] but ignored indirect competition, the result of [10] is optimistic. In this paper we treat the indirect contentions as interference jitter and get an upper bound on network latency. The analysis and relative example are represented in section IV. The analysis by Lu *et al* [18] takes account of the parallel interference in disjoint traffic-flows and tries to minimize the direct interferences of higher priority traffic-flows. But this parallel consideration is not reasonable when worst case network latency is desired.

III. Real-time communication model and assumptions

A. System model and attributes

The packet level analysis approach of real-time communication in general networks [22] in the absence of buffer restrictions is not suitable for wormhole networks. We need a new analysis model for real-time wormhole switching communication. Some conditions, assumptions and explanations are essential.

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 $\tau_i = (P_i, C_i, T_i, D_i, J_i^R)$. We assume that all the traffic-flows which require timely delivery are periodic[1]. The length of time between releases of successive packets of τ_i is a constant, which is called the period T_i for this traffic-flow. Each traffic-flow τ_i has a priority value P_i. All the packets that belong to the τ_i inherits the

[1]This periodic restriction is for presentation reason, the analysis can be extended to sporadic traffic-flows where T is the minimum inter-arrival interval.

same priority P_i. The value 1 denotes the highest priority and larger integers denote lower priorities. We assume the traffic-flow is prioritized by any possible priority assignment policy. The issue of priority assignment is beyond the scope of this paper. Each real-time traffic-flow has deadline D_i constraint which means all the packets belonging to this traffic-flow have the restriction that it should be delivered from a source router to a destination router within a certain delay bound even in the worst case situation. Our model has the same restriction as [2], [13], [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] denotes the maximum deviation of successive packets released 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 $a + D_i$.

The *basic network latency* happens when no traffic-flow contention exists. The basic network latency is determined by its source/destination routing distance, packet size, link bandwidth and some additional protocol control overheads. For real-time evaluation purpose, we use the term *maximum basic network latency*. Let H denote the hops between source and destination nodes and S indicate the constant processing delay in each router. Let the maximum packet size belonging to τ_i be L_i^{max} and the flit size for wormhole switching be f_{size}, the maximum basic network latency C_i [8] is given by:

$$C_i = \lceil \frac{L_i^{max} + L^{add}}{f_{size}} \rceil \cdot f_{size}/B_{link} + H \cdot S \qquad (1)$$

where L^{add} is the additional data for wormhole switching such as header and tail flit information. Here we only consider the longest possible, or maximum, basic network latency for evaluation. If the real-time traffic-flow can meet its deadline with the maximum basic network latency scenario, it will meet the deadline for any other basic network latency scenario. Note that the network latency here does not consider contention. The competing interventions can disturb and extend the packet network latency. The competing interventions and related worst case evaluation are discussed in section IV.

B. 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 $G : V \times E$. 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. For the case in Figure 1, the routing of τ_1, τ_2, τ_3 and τ_4 are $\Re_1 = \{e_{7,11}, e_{11,15}, e_{15,14}\}$, $\Re_2 = \{e_{13,9}, e_{9,5}, e_{5,1}, e_{1,2}\}$, $\Re_3 = \{e_{15,14}, e_{14,13}, e_{13,9}\}$ and $\Re_4 = \{e_{13,9}, e_{9,5}, e_{5,1}\}$. The intersection sets between them are $\Re_1 \cap \Re_2 = \Re_1 \cap \Re_4 = \emptyset$, $\Re_1 \cap \Re_3 = \{e_{15,14}\}$, $\Re_2 \cap \Re_3 = \{e_{13,9}\}$, $\Re_2 \cap \Re_4 = \{e_{13,9}, e_{9,5}, e_{5,1}\}$ and $\Re_3 \cap \Re_4 = \{e_{13,9}\}$.

The packet advances when it receives the bandwidth of all the links along the path. 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 measured. The maximum basic network latency can be calculated by Eq.(1). Therefore the important factor dominating the latency upper bound is interference. Under the priority arbitration policy, only traffic-flows with higher priority than the current one can cause interference. So we should find all the higher priority traffic-flows which could affect the observed one and calculate the interference upper bound by analyzing the characteristics of these higher priority traffic-flows.

Kim *et al* [13] 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. 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. S_i^D includes all the traffic-flows which meet the following condition: $S_i^D = \{\tau_k | \Re_k \cap \Re_i \neq \emptyset \textit{ and } P_k > P_j > P_i \textit{ for all } \tau_k \in \Gamma\}$. 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 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, $S_i^I = \{\tau_k | \Re_k \cap \Re_i = \emptyset \textit{ and } \Re_k \cap \Re_j \neq \emptyset, \Re_j \in S_i^D \textit{ and } P_k > P_i \textit{ for all } \tau_k \in \Gamma\}$. For each traffic-flow from higher priority to lower priority, the set S_i consisting of all traffic-flows with direct/indirect interference is constructed: $S_i = S_i^I + S_i^D$.

An example is shown in Figure 1. Four prioritized traffic-flows sorted from high to low priority are τ_1, τ_2, τ_3 and τ_4. Flows τ_1 and τ_2 have no shared links with any other higher priority traffic-flow so no direct or indirect interference, $S_1 = S_2 = S_1^I = S_1^D = S_2^I = S_2^D = \emptyset$. Flow τ_3 competes with τ_1 and τ_2 and gets $S_3^I = \emptyset$ and $S_3 = S_3^D = \{\tau_1, \tau_2\}$. Flow τ_4 directly contends with τ_2 and τ_3 and indirectly suffers interference from τ_1, $S_4^D = \{\tau_2, \tau_3\}$, $S_4^I = \{\tau_1\}$ and $S_4 = \{\tau_1, \tau_2, \tau_3\}$. Note that if one traffic-flow is both contending directly and indirectly with an observed one, then this traffic-flow will be regarded as generating direct contention only.

With the assumption and definition of the communication model, we will give, in the next section, a determinant upper bound on the schedulability of real-time traffic-flows.

IV. Network latency upper bound analysis

An efficient approach is necessary in order to evaluate all the possible competing interferences imposed by all the higher priority traffic-flows. We find that the observed traffic-flow has a relationship of resource competition with its relevant higher priority traffic-flows. This is similar to the processor resource model [17], [1], [21] in real-time scheduling, in which all the tasks contends for the shared processor resource. In wormhole switching networks, the shared physical communication links are also contended for by the associated traffic-flows. Utilizing the preemptive fixed priority scheduling policy, we can analyze this model following the real-time scheduling approach in single-processors.

We introduce the concept of *worst case network latency* which is inspired by *worst case response time* (WCRT) [14] proposed in real-time system scheduling. A traffic-flow is schedulable if and only if the network latency of all the packets belonging to this traffic-flow is no more than its deadline. If we can find the worst case network latency of this traffic-flow, we can judge whether this traffic-flow is schedulable. Generally, we need to find the condition which can trigger the traffic-flow's worst case network latency. Liu and Layland in their seminal paper [17] identified two major conditions to achieve the worst case response:

- All the tasks execute for their worst-case execution time and all tasks are subsequently released at their maximum rate.
- The task is released at the *critical instant*.

The critical instant is the time that the task is requested simultaneously with requests of all higher priority tasks. We borrow the concept of critical instant to apply it for real-time scheduling in wormhole switching. The worst case network latency is assumed to occur when the packet from the observed traffic-flow is fired simultaneously with all the packets from higher priority traffic-flows with their maximal release rates.

We have discussed that the worst case network latency is primarily determined by the interference after computing the maximum basic network latency. Next, we need to quantify the analysis based on two distinguishing interferences:

- Direct interference from higher priority traffic-flows.
- Indirect interference from other higher priority traffic-flows.

A. Interference from direct higher priority

For an observed traffic-flow τ_i, the network latency R_i of a packet released from τ_i is:

$$R_i = B + C_i + I_i \tag{2}$$

where I_i is the interference summation from the higher priority traffic-flow(s). The maximum basic network latency C_i is constant and known a prior by static analysis (Eq.(1)). B is the maximum blocking time by any lower priority traffic-flow which has already begun transmission. The maximum blocking happens when a higher priority packet arrives just after a lower priority packet starts its service. Consider our flit-level preemptive scheduling strategy, the higher priority packet waits at most one flit time and then starts its transmission at each hop output port. The maximum blocking time is represented by $B = f_{size} \times H/B_{link}$. The flit size and link bandwidth is constant after on-chip network configuration. Thus, the blocking time could be regarded as a constant parameter and incorporated in the basic network latency C_i. The network latency equation Eq.(2) is simplified into

$$R_i = C_i + I_i \tag{3}$$

We assume the packet from the observed traffic-flow is released simultaneously with all the packets from 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 R_i when the packet is accepted completely by the receiver, during the time interval [0, R_i], the maximum possible direct competition interference from higher priority traffic-flows in S_i^D to a packet from τ_i when release jitter is considered is:

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

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 traffic-flow can release during the time interval [0, R_i]. Using Eq.(4) to substitute I_i in Eq.(2), we can produce:

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

We find the variable R_i appears on both sides of the Eq.(5). This equation can be solve using an iterative technique [1]. Let r_i^{n+1} be the $(n+1)^{th}$ iterative value generated from the equation:

$$r_i^{n+1} = \sum_{\forall \tau_j \in S_i^D} \lceil \frac{r_i^n + J_j^R}{T_j} \rceil C_j + C_i \tag{6}$$

The iteration starts with $r_i^0 = C_i$ and terminates when $r_i^{n+1} = r_i^n$. This iteration also should halt if $r_i^{n+1} > D_i$, which denotes the deadline miss for this packet. By this iterative technique, the worst case network latency with direct interference can be calculated (R_i=r_i^{n+1}=r_i^n).

B. Interference from indirect higher priority

The model indicated by Eq.(5) only considers the direct interference from higher priority traffic-flow. Indirect interference also needs to be taken into account.

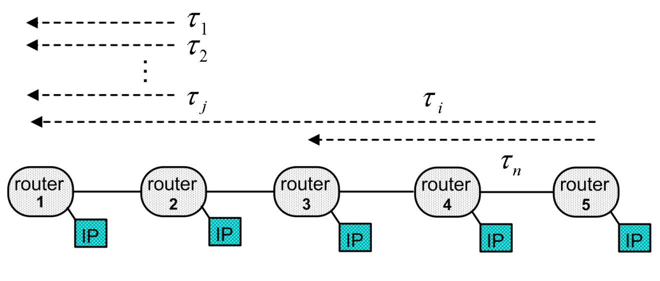

Fig. 3. A Case of Indirect Interference

Consider the following scenario: $\tau_1, \tau_2, \ldots, \tau_j$, τ_i and τ_n are loaded on the network with the inter-relations of traffic-flows in Figure 3. Traffic-flows are sorted with priority descending, τ_1 with highest priority, τ_n with lowest priority. We examine the competing relation of τ_n and τ_i and get $S_n^D = \{\tau_i\}$, $S_n^I = \{\tau_1, \ldots, \tau_j\}$, $S_i^D = \{\tau_1, \ldots, \tau_j\}$ and $S_i^I = \emptyset$. Without considering the possible indirect interference from $\tau_1, \ldots, \tau_j$, the worst case network latency of τ_n occurs when τ_i and τ_n are released at the same time.

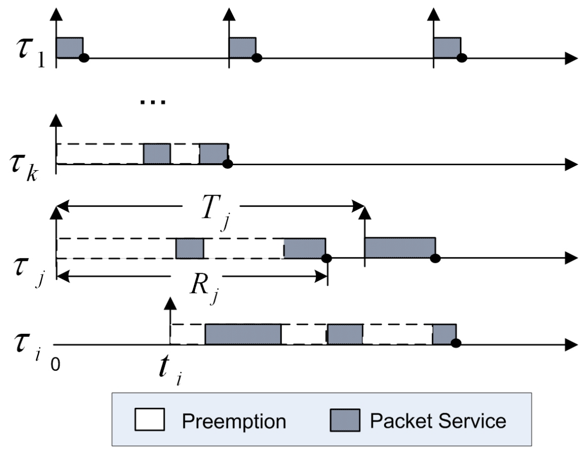

Fig. 4. The Problem of Indirect Interference

When $\tau_1, \ldots, \tau_j$ as indirect higher priority traffic-flows are taken into account, even though they do not share any physical link with τ_n, we find their services still can impose an extra interference on τ_n. The time-line graph in Figure 4 shows such a situation. The solid up arrow in the graph indicates the release time of a packet from a traffic-flow. A packet served by the network for some time is depicted as a shaded rectangle. The preemption of a packet is depicted as a white rectangle. The bold circle denotes the complete packet received by the destination. We assume τ_i is released with the other higher priority packets at the same time 0, the packets from $\tau_1, \ldots, \tau_j$ will contend with τ_i. This contention delays the start time of τ_i until time t_i, t_i is the start time of τ_i first transmission service. At the time $0 + t_i$, τ_n is released, the packet from τ_i immediately preempts τ_n. It is easy to find that τ_i imposes the interference C_i upon the τ_n during the time interval $[t, R_i]$. At the time $0 + T_i$, τ_i is released again but this time all the higher priority traffic-flows $\tau_1, \ldots, \tau_j$ only send a very small packet or even do not send any packet. Flow τ_i in this situation does not suffer any interference from them and gets network service immediately. From the view of τ_n, the time interval between two successive releases from τ_i is only (T_i - t_i). Consider our original assumption, the worst case network latency occurs when all the packets from the higher priority traffic-flows are released periodically with the minimum packet release interval T. The maximum interference a packet from τ_n suffered from a higher priority traffic-flow is calculated by $\lceil \frac{R_n + J_i^R}{T_i} \rceil$. But in this case, the minimum interval between subsequent preemptions from τ_i is only (T_i - t_i) which is less than the original minimum interval assumption T_i. Note that this phenomenon can only occur when considering indirect interferences.

Theorem 1: The upper bound of interference suffered by τ_n from direct higher priority traffic-flow τ_i is:

$$\lceil \frac{R_n + R_i - C_i + J_i^R}{T_i} \rceil C_i \tag{7}$$

when the indirect interference is considered.

Proof: Let s_i denote the packet release time from τ_i. In this analysis assumption, without loss of generality, the first packet is released at time 0. Therefore, each packet from τ_i is generated periodically at the time instant 0, T_i, $2T_i$, ..., kT_i. $s_{i,k} = (k-1)T_i$, where k is the sequential number of the packets. But a real application does not always meet this constraint and has application release jitter. More specifically, the release time $s_{i,k}$ satisfies:

$$(k-1)T_i \leq s_{i,k} \leq (k-1)T_i + J_i^R \tag{8}$$

In addition, in Figure 4, we observe that the possible interference from higher priority packets also defers its starting service time. If the worst case network latency for τ_i is R_i, the upper bound of start service time is $R_i - C_i$. The real service start time for each packet satisfies:

$$(k-1)T_i \leq s_{i,k} \leq (k-1)T_i + J_i^R + R_i - C_i \tag{9}$$

Now we evaluate the maximum interference suffered by τ_n from τ_i in a given time interval. Here we assume the start service time of τ_i is $a = R_i - C_i + 0$, this is the upper bound of the start service time since it is released. A packet from τ_n is released simultaneously with τ_i and b is the corresponding completion time of this release. The worst case interference occurs when most packets from τ_i are released since τ_n is released. Figure 5 illustrates this situation:

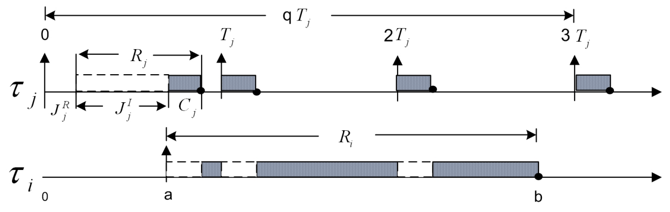

Fig. 5. Upper Bound Analysis of Indirect Interference

The number of preemptions by τ_i is given by the positive integer number g between the interval $[a, b]$, $g \in \mathbb{N}$. The last release of τ_i should fall into the interval before the completion of τ_n, g is the largest value that satisfies:

$$a - (R_i - C_i) - J_i^R + (g-1)T_i < b \tag{10}$$

or, equivalently,

$$g < \frac{J_i^R + R_i - C_i + b - a}{T_i} + 1 \tag{11}$$

The largest positive integer number satisfying this inequality is given by

$$g = \lceil \frac{J_j^R + R_i - C_i + b - a}{T_i} \rceil \tag{12}$$

The interval $[a, b]$ marks the worst case network latency of τ_n, $b - a = R_n$. Therefore, the interference upper bound from τ_i is:

$$\lceil \frac{J_i^R + R_i - C_i + R_n}{T_i} \rceil C_i \tag{13}$$

■

The packet from τ_n will experience worst latency than what predicted by Eq.(5) on account of the indirect interference from $\tau_1, \ldots, \tau_j$ which delays τ_i and further force more hits on τ_n. Therefore, the worst case network latency does occur not when the packet is released simultaneously with higher priority packets but at the point when the packet from the observed traffic-flow is released at the same time as the higher priority packets finish waiting and start to receive service.

This deviation induced by higher priority interference between consequtive releases is called *interference jitter*, using symbol J^I to denote the interference jitter of traffic-flows. The interference jitter of a traffic-flow is the maximum deviation between two successive packet start service times which can be obtained by computing the difference between the maximum and minimum value of packet start service times. Consider the situation that no higher priority packet is sent in a period, the minimum packet start service time becomes zero. Accordingly, the interference jitter of the traffic-flow is the maximum number of start service time which can be given by an upper bound:

$$J_i^I = R_i - C_i \tag{14}$$

Note that not all the traffic-flows suffer interference jitter, this only happens when the observed traffic-flow τ_n has indirect interference, namely, J_i^I exists if and only if $S_i^D \cap S_n^I \neq \emptyset$. As a result, the worst case network latency in case of interference jitter and release jitter are calculated as follows:

$$R_n = \sum_{\forall \tau_i \in S_n^D} \lceil \frac{R_n + J_i^R + J_i^I}{T_i} \rceil C_i + C_n \tag{15}$$

We modify our traffic-flow attributes with the six-tuple $(C_i, P_i, T_i, D_i, J_i^R, J_i^I)$ for τ_i. Differing from the conclusion of [2], [10], here we treat the indirect interference as interference jitter of direct higher priority traffic-flow and obtain an tighter upper bound for worst case network latency.

C. A case example

Let us revisit the example given in Figure 1. The inter-relations between these traffic-flows have been examined in section III-B. The attributes of the traffic-flows are shown in Table I. The time units are not necessary in this analysis as long as all the traffic-flows use the same base.

Real-Time Traffic-flow	C	P	T	D	J^R
τ_1	2	1	6	6	0
τ_2	1	2	5	5	0
τ_3	3	3	10	10	0
τ_4	4	4	15	15	0

TABLE I
TRAFFIC-FLOWS DESCRIPTION

Since the higher priority traffic-flow always forces interference to the lower priority traffic-flow and extends the latter's network latency, we sort the traffic-flows based on their priority and proceed to analyze them from the highest priority one by one. τ_1, τ_2 do not suffer any contention and receives the worst case network latency equal to their maximum basic latency, R_1=C_1= 2, R_2= C_2=1. Thus τ_1 and τ_2 are schedulable. τ_3 shares the physical link with the higher priority traffic-flows τ_1 and τ_2, S_3^D= $\{\tau_1, \tau_2\}$, S_3^I = $\emptyset$. The network latency for τ_3 according to Eq.(5):

$$R_3^0 = 3$$
$$R_3^1 = 3 + \lceil \frac{3}{6} \rceil 2 + \lceil \frac{3}{5} \rceil 1 = 3 + 2 + 1 = 6$$
$$R_3^2 = 3 + \lceil \frac{6}{6} \rceil 2 + \lceil \frac{6}{5} \rceil 1 = 3 + 2 + 2 = 7$$
$$R_3^3 = 3 + \lceil \frac{7}{6} \rceil 2 + \lceil \frac{7}{5} \rceil 1 = 3 + 4 + 2 = 9$$
$$R_3^4 = 3 + \lceil \frac{9}{6} \rceil 2 + \lceil \frac{9}{5} \rceil 1 = 3 + 4 + 2 = 9$$

The recurrence stops at R_3 = 9 which is less than the deadline 10. The worst case network latency of τ_3 is 9.

Flow τ_4 suffers both direct and indirect interferences with S_4^D=$\{\tau_2, \tau_3\}$, S_4^I=$\{\tau_1\}$. Based on the principle proposed in Section IV-B when indirect interference exists, we treat indirect interference as interference jitter and therefore update the attributes of our example. The interference jitter of traffic-flow τ_3 referred to τ_4 equals $R_3 - C_3$ = 6. Eq.(15) becomes

$$R_4 = C_4 + \lceil \frac{R_4}{T_2} \rceil C_2 + \lceil \frac{R_4 + J_3^I}{T_3} \rceil C_3$$

which stops at R_4 = 13. So, the worst case network latency of τ_4 is 13 with both direct and indirect interference. Again, the traffic-flow meets its deadline. Note that if τ_1 is considered as direct interference (Balakrishnan's analysis approach [2]), the worst case network latency calculated above is 19 which means it misses the deadline.

V. TIGHTNESS OF THE ANALYSIS

By abstracting the communication resources and finding the inter-relations among the traffic-flows, we have succeeded in transforming the real-time wormhole scheduling approach into an analyzable model. In this section, we show that our analysis approach can produce an upper bound of network latency for all situations. We also show that the exact worst case network latency evaluation in the parallel interference situation, as a general problem, is NP-hard. A case example is used to illustrate that the latency upper bound is not tight when where is parallel interference.

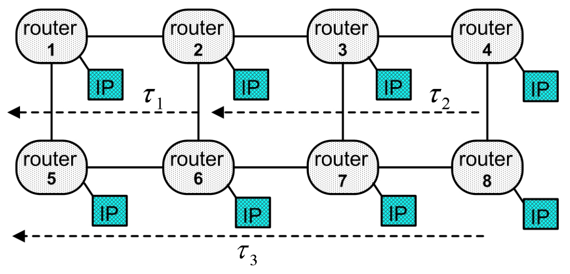

Fig. 6. Parallel Interference Case

Suppose the relations of traffic-flows shown in Figure 6 with the attributes in Table II. Relying on the critical instant assumption, flow τ_3 experiences its maximum network latency when released simultaneously with τ_1 and τ_2. The worst case network latency for τ_1, τ_2, τ_3 are 1, 3, 9 according to Eq.(15). But we find that during the analysis of τ_3, τ_2 is forced to

Real-Time Traffic-flow	C	P	T	D	J^R
τ_1	1	1	5	5	0
τ_2	3	2	10	10	0
τ_3	4	3	15	15	0

TABLE II
TRAFFIC-FLOWS DESCRIPTION

compete with τ_1 in this model even though they do not share any physical link; and this will force an extra delay on τ_3. The problem is when we schedule all the traffic-flows with the resource competing relationship, we always ideally assume at any time instant only one traffic-flow can be served by the subset of the network which hosts the interfering flows service.

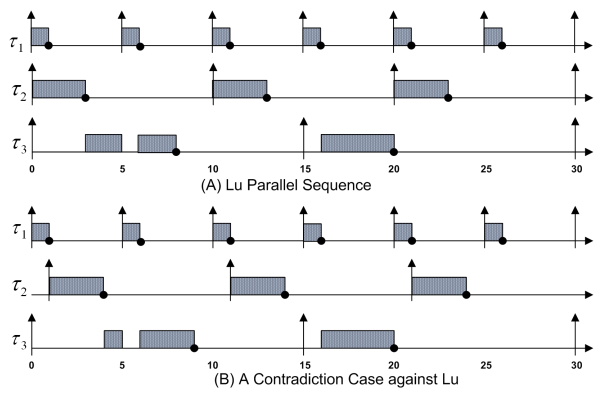

Fig. 7. Scheduling Sequence With Parallel Interference

Lu *et al* in his paper [18] found this phenomenon that real-time transmission scheduling can be parallel for disjoint concurrent contentions. Figure 7(A) illustrates this concurrency. If all traffic-flows release the packets simultaneously, τ_1 and τ_2 are executed at the same time in this scheduling sequence. This parallel interference reflects the fact that no real link resource is shared between the direct higher priority traffic-flows of the observed one. For τ_i, this parallel interference phenomenon exists when the following condition is met: $\tau_j, \tau_k \in S_i^D$ *and* $\Re_j \cap \Re_k = \emptyset$. Lu *et al* [18] also gave an approach to analyze this phenomenon. The worst case network latency is assumed to occur when all the traffic-flows are released simultaneously and the disjoint traffic-flows are scheduled in parallel. The corresponding network latency of packet in τ_1, τ_2 and τ_3 are 1, 3, 8 in Lu's model which are smaller than previous results. However we must point out the analysis by Lu *et al* [18] is defective. Even though this parallel interference can possibly reduce the intervention to the lower priority traffic-flow since many of them are not likely to occur at the same time. But the worst case latency does not always occur when all the traffic-flows are released at the same time. Figure 7(B) is a contradictory case against this assumption. The network releases τ_1 and τ_3 at the same time, τ_2 is ready just before τ_1 accomplishes its transmission. τ_2 consequently holds the link resource until service completion. After that τ_3 starts its packet transferring. In this situation, the lower priority traffic-flow τ_3 suffers more interference. The network latency of τ_3 is 9 more than the upper bound value produced in Lu's analysis.

The parallel interference phenomenon does not appear in all the traffic-flow set loaded on the network. However, this possible parallelism clearly complicates the analysis progress. In general scheduling, when the worst case release conditions are not easily determined, the analysis is usually intractable. Let us re-visit our analysis model, for τ_i, the total interferences generated by all direct higher priority traffic-flows are the sequence summation of each traffic-flow, $I_i = \sum_{j \in S_i^D} \lceil \frac{R_i + J_j^R}{T_j} \rceil$. This assumption can produce a tight result for each traffic-flow when there is no parallel interference from direct higher priority traffic-flows; the traffic-flow set in Figure 1 is a case. However this assumption ignores the possible simultaneous communication service. The following proof shows that this approach is not tight and only an upper bound of network latency can be produced when parallel interference exists.

Theorem 2: The real worst case network latency is no more than the calculation result accomplished by Eq.(5) and Eq.(15) when parallel interference exists.

Proof: Similar to the scheduling in the single processor model, this model also implies that at any time only one traffic-flow can win the access right of the shared resource and execute the service. Let t_1, t_2 indicates the time instant after network startup. During any time interval $[t_1, t_2]$ where $t_1 < t_2$, the maximum required service time of all the higher priority traffic-flows (namely, interference to the observed traffic) is no more than $\sum_{\forall j \in S_i^D} \lceil \frac{t_2 - t_1 + J_j^R + J_j^I}{T_j} \rceil C_j$. Since parallel service exist, more service opportunity can be supported by the network in any time interval. This may accelerate consumption of the required service from higher priority traffic-flows and finally shortens the whole interference imposing on the observed one. The network latency is exactly equal to the interference of all the higher priority traffic-flows plus service time of itself, Eq.(2). Consequently, the worst case network latency in the real network is no more than the calculation result under the assumption of no parallel interference. ■

Theorem 2 implies our model and analysis approach is only sufficient but not necessary when parallel interference exists. In another words, if a traffic-flow can pass this schedulability test then it can meet its deadline. But if it fails this test, no similar conclusion can be made. Next, we give a proof that it is not possible to produce a polynomial-time necessary and sufficient test unless P=NP.

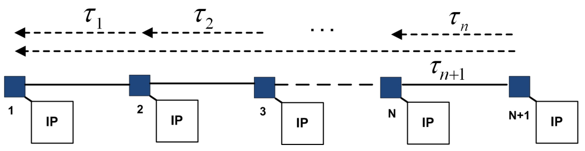

Fig. 8. A General Parallel Interference Model

Suppose a parallel communication instance in Figure 8, the set Γ includes $N+1$ traffic-flows distributing across the network with attributes τ_1=$(C_1, P_1, T_1, D_1,\ J_1^R, J_1^I, s_1)$, τ_2= $(C_2, P_2, T_2,\ D_2, J_2^R, J_2^I, s_2)$, ..., τ_{n+1}=$(C_{n+1}, P_{n+1}, T_{n+1}, D_{n+1}, J_{n+1}^R, J_{n+1}^I, s_{n+1})$. s_i is the release time of τ_i and τ_1 with highest priority, τ_{n+1} with lowest priority. First N

traffic-flows are disjoint with each other and communication in parallel. The observed one τ_{n+1} with lower priority contends with first N higher priority traffic-flows, so S_{n+1}^{D} = $\{\tau_1, \tau_2, \ldots, \tau_n\}$. In order to determine the network latency, we need to take account all the possible free gap intervals for the observed traffic-flow. The case in Figure 7(B) implies the worst case network latency no longer occurs when all the traffic-flows are released simultaneously. Thus, we need to examine all the possible packet release sequences to achieve the worst case network latency calculation. We now shown that to solve this problem when parallel interference exists is NP-hard.

Lemma 1: For traffic-flows set Γ meeting the conditions shown in Figure 8, the observed traffic-flow τ_{n+1} is schedulable on the network if and only if all the deadlines are met during time interval $[0, t_1]$, where $t_1 = s + P$, $s = max\{s_1, \ldots, s_{n+1}\}$ and $P = lcm\{P_1, \ldots, P_{n+1}\}$.

Proof: $\Rightarrow$: If the observed traffic-flow is schedulable on the network, relying on the schedulable condition, all the deadlines are met since it was released. Thus, all deadlines in the interval [0, t_1] are met.

$\Leftarrow$: At any time instant t_i, the network state is denotes by $E(e_1, \ldots, e_n)_{t_i}$, where e_i is the amount of time for τ_i has finished transmission service since last release. The time interval [0, $s + P$] is separated into two parts [0, s) and [s, s+P]. After s, all the traffic-flows are released and executed in parallel except τ_{n+1}. Let $t_i \in [s, s + P]$ and t_j = $t_i + j \times P$ for j is positive integer and $j > 0$, it is not difficult to see the network state $E_{t_i} = E_{t_j}$ is always true. This means the schedule progress repeats itself every P units of time after s. Since all deadline of τ_{n+1} in [s, s+P] are met, all the deadline after $s+P$ should also be met. As the assumption, the deadline of τ_{n+1} in [0, s] are met. Thus τ_{n+1} is schedulable in this network. ■

Strictly speaking, we need to check all the infinite scheduling sequences after packet releases of the traffic-flow set to ensure the deadline is always met. Lemma 1 gives us an easy method to estimate whether the traffic-flow set is schedulable in finite schedule time interval $[0, s+P]$. If all the deadlines in $[0, s+P]$ are met, then this traffic-flow set is a valid schedule. If we know period, maximum basic network latency and release time of all packets in a traffic-flow set in advance, we can find all the available gap intervals in [0, s+P] for observed traffic-flow with a polynomial time algorithm. Determining the schedulability of observed traffic-flow, namely finding the worst case network latency when knowing all the available gap intervals in advance is also taking polynomial time. Now we prove that if the traffic-flow set with arbitrary release time, finding the worst case network latency is intractable. Utilizing the K Simultaneous Congruences [15] which has been shown to be the NP-complete, our problem can be proved to be intractable by reducing the current problem to a known NP-complete problem in polynomial time.

K Simultaneous Congruences: Given N ordered pairs of positive integers (a_1,b_1), ..., (a_n, b_n) and a positive integer K ($2 \leq K \leq N$). Is there a subset of $\ell \geq K$ ordered pairs $(a_{i,1}, b_{i,1}), \ldots, (a_{i,\ell}, b_{i,\ell})$ such that there is a positive integer x with the property that $x = a_{i,j} + p_j \times b_{i,j}$ for each $1 \leq j \leq \ell$?

Theorem 3: For a traffic-flow set with arbitrary release time, the problem of determining schedulability when parallel interference exists is NP-hard.

Proof: The proof includes two steps, first, we try to prove finding all the gap intervals for the observed traffic-flow is NP-hard. Suppose n sorted pairs of positive integers (a_1,b_1), ..., (a_n,b_n) with constraint $a_i \geq b_i - 1$ and a positive integer K, we construct n parallel traffic-flows communications $(\tau_1, \ldots, \tau_n)$ with attributes $T_i = D_i = b_i$, $C_i = b_i - 1$, and $s_i + C_i = a_i$. In our construction, each traffic-flow in each period only exports one free time unit. For this traffic-flow set, finding the gap interval on the network only and only if all the N traffic-flows output free time units simultaneously. Thus, the K Simultaneous Congruences problem has a solution if and only if the gap intervals can be found in the constructed model. Since this construction progress can be done in polynomial time, the problem of finding all the gap intervals is as hard as the K Simultaneous Congruences problem. We assume a traffic-flow τ_{n+1} which has the lowest priority comparing with $\tau_1, \ldots, \tau_n$. Determining the schedulability of τ_{n+1} when knowing all the gap intervals is polynomial time complexity, so the problem of determining schedulability in the parallel interference situation is NP-hard. ■

VI. Conclusion

The new on-chip communication architectures need to provide different levels of service for various components on the same network. The requirement of real-time applications needs a scheduling strategy and analysis approach to predict whether all the real-time packets can meet their timing bounds. In this paper, we introduce an analysis approach for real-time on-chip communication with wormhole switching and fixed priority scheduling. The worst case network latency upper bound for each traffic-flow can be achieved for all situations. When parallel interference exists in a real network, we show that the exact determinant of schedulable for traffic-flow sets is NP-hard. Utilizing this analysis scheme, we can flexibly evaluate the schedulability of traffic-flow sets with different QoS requirements in a real-time communication platform at the design phase.

The future work will involve the issues of priority assignment and the practical consideration of having less virtual channel and priority levels.

References

[1] N. C. Audsley, A. Burns, M. Richardson, K. 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. Powering networks on chips. In *ISSS*, pages 33–38, 2001.

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

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

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

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

[8] J. Duato and S. Yalamanchili. *Interconnection Networks: An Engineering Approach*. Institute of Electrical & Electronics Enginee; 1st edition, 1997.

[9] S. Furber and J. Bainbridge. Future trends in SoC interconnect. In *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] J. Henkel, W. Wolf, and S. Chakradhar. On-chip networks: A scalable, communication-centric embedded system design paradigm. In *VLSID '04: Proceedings of the 17th International Conference on VLSI Design*, page 845, Washington, DC, USA, 2004. IEEE Computer Society.

[12] N. Kavaldjiev and G.J.M. Smit. A survey of efficient on-chip communications for soc. *CTIT*, 2003.

[13] 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 1998 International Conference on Parallel Processing*, pages 527–534, Washington, DC, USA, 1998. IEEE Computer Society.

[14] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *IEEE Real-Time Systems Symposium*, pages 201–213, 1990.

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

[16] J. P. Li and M. W. Mutka. Priority based real-time communication for large scale wormhole networks. In *Proceedings of the 8th International Symposium on Parallel Processing*, pages 433–438, Washington, DC, USA, 1994. IEEE Computer Society.

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

[18] 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, New York, NY, USA, 2005. ACM Press.

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

[20] 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, Washington, DC, USA, 1997. IEEE Computer Society.

[21] K. W. Tindell, A. Burns, and A. J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time System.*, 6(2):133–151, 1994.

[22] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. In *Proceedings of the IEEE*, volume 83, pages 1374–1396, 1995.