Improvement of Schedulability 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

Priority-based wormhole switching with a priority share policy has been proposed as a possible solution for real-time on-chip communication. However, the blocking introduced by priority share complicates the analysis process. In this paper, we propose a new "per-priority" basis analysis scheme which computes the total time window at each priority level instead of each traffic-flow. By checking the release instance of each flow at the corresponding priority window, we can determine schedulability efficiently.

1 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 importance in today's System-on-Chip (SoC) designs. The typical architecture of on-chip networks consists of multiple *intellectual property (IP)* modules connected through an interconnection network. This architecture offers a general and fixed communication platform which can be reused for a large number of SoC designs.

Multiple IP-cores based design using NoC allows multiple applications to run at the same time. These applications execute data processing and exchange information through the underlying communication infrastructure. Some 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.

The on-chip network is a significant solution for complex communication of SoCs and outperforms the traditional busses or a point-to-point approach in many ways [8]. But it also introduces unpredictable network delay since the expensive hardware resources (e.g. link bandwidth and buffer space) are usually shared by a number of applications. When more than one packet tries to access the shared resource at the same time, contention occurs. The contention problem, which leads to packet delays and even missing deadlines, has become the major influence factor of network predictability. So how to solve contention problem is a key issue in implementing guaranteed performance service in NoC design.

Contention avoidance and contention acceptance are two basic approaches to address the contention problem. First approach considers that contention is avoidable by trying to pre-arrange and allocate resources before the start of the communication, so that two packets never access the same resource at the same time. Time division multiplexing (TDM) and circuit switching are two common contention avoidance schemes. In Ætheral [10] and Nostrum [15], the whole link transmission capacity is partitioned into fixed time-slots, each of which represents a unit of time when a single application can occupy this physical link exclusively for data transmission. But this scheme requires a global notion of time in the network. Besides that, the latency is coupled to bandwidth, preventing low latency from being provided to low rate requirement without over-allocating. A circuit-switching technique is used in [20, 19]. A dedicated connection is constructed between source and destination nodes by reserving a sequence of wiring resources. The major problem of this scheme is that the resources that have been reserved for a flow can not be used by any other flow which results in the under-utilized links. The contention avoidance policy requires the network resource to be configured before the communication which lacks flexibility and

wastes the links transmission capacity.

A contention acceptance policy normally utilizes an arbiter at running time. QNoC [6] divides network services into four levels and utilizes a priority arbiter to implement the differentiated services. But this scheme only offers the coarse-granularity service and does not seem to be suitable for hard real time application. Kavaldjiev *et al* [12] presented a simple round-robin arbiter to cater for the real time services. In Mango NoC project [5], a new arbiter is designed which combine round-robin and priority to bound latency and bandwidth. But both of them suffer the same problem as TDM that the latency is coupled with bandwidth. As a new solution, a priority based wormhole switching technique [17] is introduced to provide communication service guarantees. The hard timing bound is delivered by this approach with the support of a priority based router infrastructure which allocates each traffic-flow with a distinct priority and virtual channel independently. This scheme successfully overcomes the problem of latency coupled with bandwidth and thus supports a wide range of traffic types. Latency analysis and validation have been discussed in a number of papers [2, 11, 13, 14, 17]. But the drawback of the priority-based wormhole switching approach is precisely that the distinct priority per traffic-flow implementation policy results in higher area and energy overhead and hence limits its employment and development in on-chip networks. To solve this problem, Shi and Burns [18] proposed a priority share policy to reduce the resource overhead while still achieving hard real-time communication guarantees. The priority share policy permits multiple traffic-flows to contend for a single virtual channel and share the same priority level. In that paper [18], the authors also presented a composite model analysis scheme. But this approach requires that all the traffic-flows must meet the constrain that network latency is no more than period. This is a strong restriction. The more complex the system, with long communication delays over several hops, the greater the global delays will become.

In this paper, we propose a new analysis approach which can efficiently handle wormhole switching with a priority share policy. The new analysis is based on "per-priority basis", that is, it computes the total time window at each priority level instead of each traffic-flow. By checking the packet release instance of each traffic-flow at the corresponding priority window, we can verify the timing semantics of real time traffic-flow with a simple yet efficient mechanism. The deadline no more than period constraint is successfully removed by this approach with a low computational complexity. In addition, we also find that the previous result proposed in [18] is just a special case covered by this new analysis.

The rest of this paper is organized as follows: Section 2 introduces wormhole switching networks with a priority share policy. Section 3 describes the real-time communication model and notations used in this paper. A novel schedulability analysis technique and related example are presented in sections 4 and 5. Finally, section 6 concludes the paper.

2 Wormhole switching with priority share

2.1 Wormhole switching structure

Wormhole switching [16] is an increasingly common interconnect scheme for NoC as it minimizes communication latencies, requires small buffer and is simple to implement. 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.

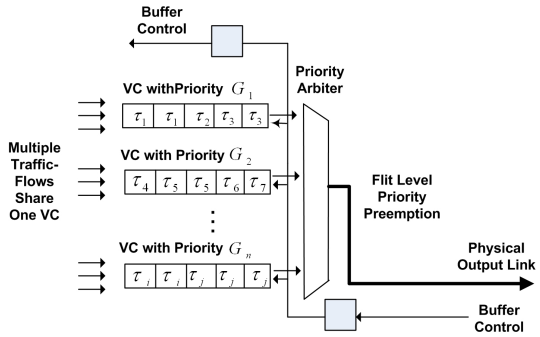

Figure 1. Output Arbitration with Priority Share

In order to ensure hard real-time service guarantees with limited resources, a priority share based flit-level arbitration structure is introduced [18], 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) area 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. The credit-based flow control provides each virtual channel of each router with some credit, which is equal to the buffer size of that virtual channel of the subsequent router. The credit is decremented upon transmitting a flit and incremented upon receiving a buffer-free notification from the next router. 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 per-

formance with a very small buffer overhead [4].

Differing from previous works [11, 13, 17], the distinctive characteristic of the priority share scheme is that multiple traffic-flows per virtual channel are supported. These traffic-flows sharing the same virtual channel will be 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 channels associated with priority G_i. At any time, a flit of a given packet will be sent out through its respective output port if it has the highest priority and it has credit(s). In addition, a higher priority packet can also preempt a lower priority packet during its transmission. As a hybrid solution, best-effort traffic-flows also can be multiplexed on the same links with lowest priority (any real time flow has higher priority than best-effort flows). In the case where no real time flow is available, best-effort flows make use of spare bandwidth.

2.2 The problem of blocking

By sharing priority, the hardware resource overhead can be reduced dramatically compared with the traditional distinct priority per traffic-flow scheme [11, 13, 17]. But on the other side, it may lead to significantly blocking and unpredictable network latency. Consider the fact that traffic-flows within the same virtual channel are served in first-in-first-out (FIFO) order because the priority preemption is only available between the different virtual channels. When a packet has to wait for the transmission of another packet (this packet can be released from the same flow or other flow) in the same buffer due to priority share, blocking occurs. Therefore, a packet can be blocked by every packet with the same priority which arrives just 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.

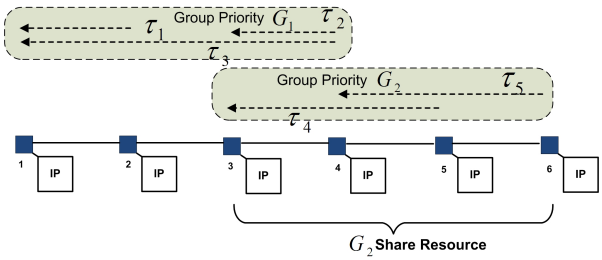

Figure 2. A Case of Traffic-flows with Priority Share

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 a 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 will still occupy 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 [18]) 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 a major role in determining τ_5's transmission latency. This phenomenon only exists when the network supports priority share. The latency analysis in this situation is very hard due to the complicated blocking inter-relationship between the flows.

To simplify the blocking problem, the analysis in [18] imposes a deadline no more than period restriction so that one traffic-flow can not be blocked by another flow with the same priority more than once; this is termed **single blocking**. With this property, all the flows sharing the same priority are transformed into a single scheduling unit and the maximum network latency is addressed by this new model. However, without this enforced constraint, we find two additional blocking phenomena may appear which also need to be considered.

Multiple blocking : A set of traffic-flows sharing the same priority; one could block another more than once. Figure 3(A) shows such a situation. The solid up arrow indicates the packet release instance. The packet's latency is depicted as horizontal arrow line. τ_a shares the same priority as τ_i, if the transmission latency of τ_i is bigger than the period of τ_a, τ_i could be blocked by τ_a more than once.

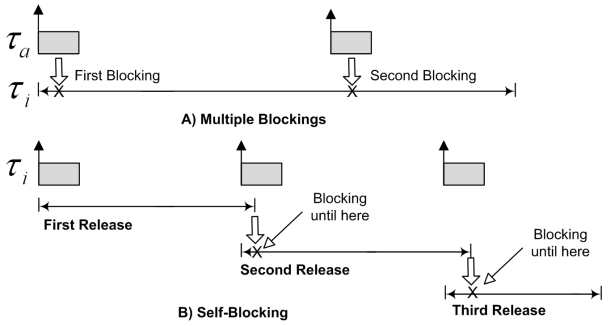

Figure 3. The Blocking Problem

Self-blocking : In a situation while the end flits from a previous packet are being delivered, the start flits of the next packet from the same flow are already introduced. Therefore, the possible blocking delay suffered by the new arrival packet

comes from not only the other flows with the same priority but also the flow itself, we refer to this as *self-blocking*. Figure 3(B) shows such a situation. In this example, the second packet released from τ_i is blocked by the first one until its completion. Similarly, the third packet is also blocked by the second one, etc.

3 System model and definition

3.1 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 has a set of properties and timing requirements which are characterized by six-tuple attributes $\tau_i = (G_i, C_i, T_i, D_i, J_i^R, J_i^I)$. 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 [17]. Each real-time traffic-flow has relative deadline D_i which is the upper bound restriction of network latency. There is no restriction on the relationship between deadline D_i and period T_i. Any flow's deadline can be less than, equal to or greater than its period. 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 [17] which denotes the maximum deviation of successive packets start transmission time. Besides these, each traffic-flow has a priority G_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, e.g. a greedy priority allocation algorithm has been proposed in [18]. All the traffic-flows competing for the same virtual channel will be allocated to the same priority. Therefore, each priority level G_i can be regarded as a flow set denoted by $S(i)$. The priority should be assigned off-line and remain constant at run-time. We also define a functions $G(\tau_i)$ to obtain the corresponding priority for a traffic-flow τ_i. It follows that $\tau_i \in S(G(\tau_i))$.

3.2 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 $\Re_i \cap \Re_j \neq \emptyset$. In the indirect competing relationship, 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. For example, if there are three flows τ_i, τ_j, τ_k meeting the following situation, $\Re_k \cap \Re_i = \emptyset$, $\Re_k \cap \Re_j \neq \emptyset$ and $\Re_i \cap \Re_j \neq \emptyset$; τ_j is the intervening flow in this case, then there is an indirect competing relationship between τ_i and τ_k. Notice that indirect competing has transitivity. If more than one intermediate flow exists, the indirect competing relationship still holds. Following the above case, if there is a new flow τ_a which has an indirect competing relationship with τ_j (τ_k is intermediate flow between τ_a and τ_j), and τ_a does not share any physical link with τ_i, then there is an indirect competing relationship between τ_a and τ_i (both τ_k and τ_j are intermediate flows). Figure 4 shows the situation.

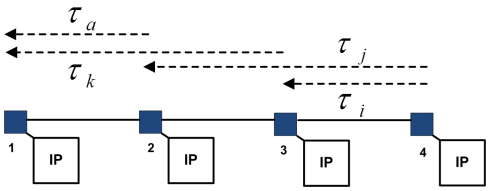

Figure 4. Transitivity in indirect competing relationship

Assuming that the 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 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 priority**
  When two traffic-flows τ_i and τ_j have a direct compet-

ing relationship and meet the condition $G(\tau_j) > G(\tau_i)$, τ_j will force a direct interference with the observed traffic-flow τ_i. For flow τ_i, we define a direct interference 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 priority**
  When two traffic-flows τ_i and τ_k have an indirect competing relationship and meet the condition $G(\tau_k) \geq G(\tau_j) > G(\tau_i)$, τ_j is the intervening flow, τ_i may suffer an indirect interference from τ_k even when they do not share any physical link, see [17] for detailed description. For flow τ_i, we define an indirect interference set S_i^I which includes all the traffic-flows meeting the above conditions, S_i^I = $\{\tau_k | \tau_k$ has indirect competing relationship with τ_i *and* $G(\tau_k) \geq G(\tau_j) > G(\tau_i)$ *for all* $\tau_k \in \Gamma$, where τ_j is any intermediate flow$\}$.

- **Direct blocking from traffic-flow with same 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 direct blocking 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 priority**
  When two traffic-flows τ_i and τ_k have an indirect competing relationship and meet the conditions $G(\tau_k) = G(\tau_j) = G(\tau_i)$, τ_j is the 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 an indirect blocking set S_i^{SI} which includes all the traffic-flows meeting the above conditions, S_i^{SI} = $\{\tau_k | \tau_k$ has indirect competing relationship with τ_i *and* $G(\tau_k) = G(\tau_j) = G(\tau_i)$ *for all* $\tau_k \in \Gamma$, where τ_j is any intermediate flow$\}$.

Note that the affect of the direct and indirect interferences has been presented in [17]. The priority share policy introduces the new direct and indirect blockings. Especially without constraint of D $\leq$ T, there are three different blocking relationships which severely complicate the analysis progress. So in this paper, we propose a new scheme which changes the analysis view from per flow to per priority. The detailed issues will be discussed in the next section.

Return to the example in Figure 2. Five traffic-flows τ_1, τ_2, τ_3, τ_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, $S(2) = \{\tau_4, \tau_5\}$; $G_1 > G_2$ in this case. Traffic-flows τ_1, τ_2 and τ_3 have no shared links with any higher priority flow so no direct or direct interference. Due to sharing the same 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_4 directly competes with higher priority flows τ_2 and τ_3 and indirect suffers interference from τ_1, $S_4^D = \{\tau_2, \tau_3\}$, $S_4^I = \{\tau_1\}$. Flow τ_5 does not have any higher priority flow, so $S_5^D =$, $S_5^I = \emptyset$. Besides that, τ_4 and τ_5 share the same priority level, thus $S_4^{SD} = \{\tau_5\}$, $S_4^{SI} = \emptyset$, $S_5^{SD} = \{\tau_4\}$ and $S_5^{SI} = \emptyset$.

4 Network latency upper bound analysis

4.1 Priority window model

The correctness of the design and development of practical real-time applications in priority-based wormhole switching relies on efficient schedulability analysis. The schedulability test in this paper 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 term *priority level-G_i shared resource* is introduced which denotes all the link resources required by the flows with the same priority G_i. This shared resource is modeled as a single competing unit. A *priority window* $W(i)$ is used to define a contiguous time interval during which this priority level-G_i shared resource keeps the network busy and serves all the traffic-flows of priority higher than or equal to the priority G_i. The priority window will continue until the time when the shared resource becomes idle, ready for the next transmission and yet there is no service requirement from priority level G_i or higher waiting to be transmitted. As shown in Figure 2, for the priority level G_2 shared resource, the links between router 3 and 6, the corresponding priority window is the contiguous time duration where the shared resource keeps serving all the queueing packets with priority G_2 or higher (G_1 in this case). Figure 5 illustrates a priority level-G_2 window. The bold circle denotes the time the packet is received completely at the destination node. In this example, the total window $W(2)$ at priority level-G_2 shared resource is the time span from the first release of τ_2 to the completion time of the second instance of τ_5.

For a set of traffic-flow $S(i)$ with the same system priority level G_i, next, we show how to compute the corresponding priority window $W(i)$.

Lemma 1. *The priority level-G_i window $W(i)$ upper bound can be calculated by the following relation:*

$$W(i) = E(i) + I(i) \tag{1}$$

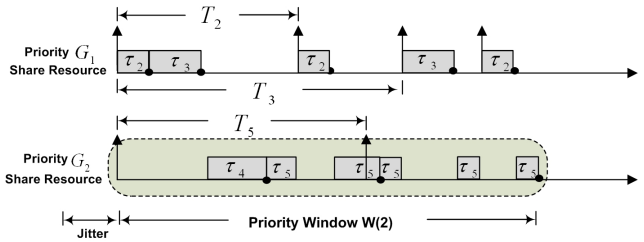

Figure 5. Priority Level-G_2 Window

where $E(i)$ denotes the summation of service requirements generated by all the traffic-flows with the priority G_i and $I(i)$ accounts all the interferences from the higher priority traffic-flows which contend the level-G_i share link resource during this window.

Proof. According to the definition of priority window, all the arrival packets of priority G_i or higher before the end of the priority window must be transmitted during the window. Besides that, any packet with priority lower than G_i is unable to delay current window. Therefore, the width of the priority window is equal to the time interval taken to serve the transmission requirements, $E(i)$, made by all the traffic-flows with the priority G_i and all the interferences, $I(i)$, from the higher priority traffic-flows which contend the level-G_i share link resource during this window.

□

The value $E(i)$ and $I(i)$ determine the priority window for the level-G_i shared resource. So if we can find an upper bound of $E(i)$ and $I(i)$, the maximum priority window $W(i)$ is then trivially computed. Note that, when we explain how to calculate the priority window for G_i priority level, we assume that analysis for all the higher priority $G_1, G_2, \ldots, G_{i-1}$ has been completed.

Theorem 1. *The maximum priority window $W(i)$ for priority level G_i share resource can be found by:*

$$W(i) = \sum_{\forall \tau_n \in S(i)} \lceil \frac{W(i) + J_n^R}{T_n} \rceil C_n + \sum_{\forall \tau_j \in hp(i)} \lceil \frac{W(i) + J_j^R + R_j - C_j}{T_j} \rceil C_j \tag{2}$$

where $hp(i)$ is the higher priority set, all the flows in S_i^D, $\tau_i \in S(i)$, are the members of the set $hp(i)$,

$$hp(i) = \bigcup_{\forall \tau_i \in S(i)} S_i^D \tag{3}$$

where $\bigcup$ is the union operation of the flow sets.

Proof. Supposing that we can find an upper bound of the total priority window $W(i)$, the maximum number of instances of a flow τ_n with priority G_i that can delay this window is computed as follows:

$$\lceil \frac{W(i) + J_n^R}{T_n} \rceil \tag{4}$$

assuming the worst case release scenario of τ_n: the first packet release starts J_n^R later than the first arrival, and the subsequent releases are maximum packet size with the maximum release rate of $1/T_n$. Consequently, the service requirement summation from all the flows in $S(i)$ is thus:

$$E(i) = \sum_{\forall \tau_n \in S(i)} \lceil \frac{W(i) + J_n^R}{T_n} \rceil C_n \tag{5}$$

On the other hand, the interferences produced by all the higher priority flows which compete the level-G_i shared resource also delay the corresponding priority window. The maximum interference analysis has been discussed in [18]. During any time interval, an upper bound of interference produced by any higher priority traffic-flow τ_j when interference jitter exists is:

$$\lceil \frac{W(i) + J_j^R + R_j - C_j}{T_j} \rceil C_j \tag{6}$$

where $R_j - C_j$ is the maximal possible jitter upper bound and R_j is the worst case latency of τ_j. Theorems 2 and 3 below will show that the network latency of τ_j can be found by calculating the corresponding priority window. The interference jitter phenomenon only happens when indirect higher priority flow exists. The paper [18] has discussed two possible conditions, $S_j^D \cap S_i^I \neq \emptyset$ or $S_j^{SD} \cap S_i^I \neq \emptyset$, where $\tau_i \in S(i)$ and $\tau_j \in S_i^D$, either of which can result in the interference jitter.

Any packet release from the higher priority flows which compete the priority level-G_i shared resource will finally extend the corresponding priority window $W(i)$. For convenience, we define a higher priority set $hp(i)$, all the flows in S_i^D, $\tau_i \in S(i)$ are inserted into set $hp(i)$. The maximum interferences produced by these higher priority flows can be computed as follows:

$$I(i) = \sum_{\tau_j \in hp(i)} \lceil \frac{W(i) + J_j^R + R_j - C_j}{T_j} \rceil C_j \tag{7}$$

Combining Eq.(1), Eq.(5) and Eq.(7) , an upper bound of priority level-G_i window in case of interference jitter and

release jitter is given by:

$$W(i) = \sum_{\forall \tau_n \in S(i)} \lceil \frac{W(i) + J_n^R}{T_n} \rceil C_n + \sum_{\forall \tau_j \in hp(i)} \lceil \frac{W(i) + J_j^R + R_j - C_j}{T_j} \rceil C_j \quad (8)$$

□

The result of $W(i)$ can be solved using the iterative technique [1]. The iteration starts with $W(i)^0 = \sum_{\forall \tau_n \in S(i)} C_n$ and terminates when $W(i)^n$ no longer increases, it has converged. By this iterative technique, the maximum priority window can be calculated ($W(i) = W(i)^{n+1} = W(i)^n$).

4.2 Maximum network latency

Based on the maximum priority window of G_i, the next step is how to find the maximum network latency for each flow in $S(i)$. For any observed 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 the priority level G_i window beginning. To calculate the worst case network latency, we need to find the latest completion time. Due to the multiple blocking and self-blocking problems, if more than one packet instance is released from the same flow during a priority level G_i window, then it is necessary to check these instances in order to find the overall worst case network latency of traffic-flow.

Motivated by the observation of the relation between priority window and period, we check the priority window at three different situations: $W(i) \leq min(T_n - J_n^R)$, $min(T_n - J_n^R) < W(i) \leq T_i - J_i^R$ and $W(i) > T_i - J_i^R$, (τ_n is any flow in $S(i)$) as showed in Figure 6.

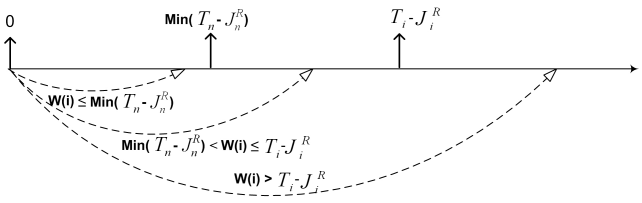

Figure 6. Three possible relations between priority window and period

Theorem 2. *The maximum network latency R_i for τ_i is given by:*

$$R_i = W(i) + J_i^R \quad (9)$$

when $W(i) \leq T_i - J_i^R$

Proof. The interval [0, $T_i - J_i^R$] is divided into two subintervals [0, $min(T_n - J_n^R)$] and ($min(T_n - J_n^R)$, $T_i - J_i^R$].

If condition $W(i) \leq min(T_n - J_n^R)$ for $\forall \tau_n \in S(i)$ is true, then the priority window ends at or before any repeated release from flow with priority G_i. This means that no traffic-flow can be blocked by any other flow sharing the same priority more than once. The multiple blocking and self blocking discussed in section 2.2 do not occur. In the worst case, τ_i will be the last flow getting transmission opportunity in this priority window. So the ending of the priority window is the completion time of τ_i's packet instance. The maximum network latency R_i for τ_i is hence given by:

$$R_i = W(i) + J_i^R \quad (10)$$

We also find that for $\forall \tau_n \in S(i)$, the relation $\lceil \frac{W(i)+J_n^R}{T_n} \rceil = 1$ is always true. Eq.(5) in this situation is simplified as $\sum_{\forall \tau_n \in S(i)} C_n$ and the priority window analysis scheme is degraded to the composite analysis presented in [18]. Actually, the composite model analysis is only a specific case in the priority window analysis when $W(i) \leq min(T_n - J_n^R)$.

If $min(T_n - J_n^R) < W(i) \leq T_i - J_i^R$ is true, this implies that only the first packet instance of τ_i is served during this window and no self-blocking occurs. But multiple packet instances from any other flow in $S(i)$ may fall into the current window because of $\exists \tau_n \in S(i)$, $\lceil \frac{W(i)+J_n^R}{T_n} \rceil > 1$. These multiple blockings will delay the completion time of the current packet, but the worst case latency still can be found by checking the priority window. In this case, only one packet instance is released by τ_i, hence the existing relation showed by Eq.(10) is still valid and hence provides the maximum network latency.

From the above discussion, the maximum network latency is calculated by Eq.(10) when $W(i) \leq T_i - J_i^R$.

□

If $W(i) > T_i - J_i^R$, then more than one packet instance of τ_i is generated during a priority level-G_i window. Some successive generated packets from τ_i might be blocked by previous ones. In this situation, the delay from self-blocking also needs to be taken into account.

Theorem 3. *The maximum network latency R_i for τ_i is given by:*

$$R_i = \max_{q=1,\ldots,\lceil \frac{W(i)+J_i^R}{T_i} \rceil} (w_q(i) - (q-1)T_i + J_i^R) \quad (11)$$

where q is the index of packet instance, and $w_i(q)$ is given

by:

$$w_q(i) = qC_i + \sum_{\forall \tau_n \in S(i), \tau_n \neq \tau_i} \lceil \frac{w_q(i) + J_n^R}{T_n} \rceil C_n + \sum_{\forall \tau_j \in hp(i)} \lceil \frac{w_q(i) + R_j - C_j + J_j^R}{T_j} \rceil C_j \quad (12)$$

when $W(i) > T_i - J_i^R$.

Proof. The number of packets that could be released from τ_i before the end of the priority window is given by:

$$\lceil \frac{W(i) + J_i^R}{T_i} \rceil \quad (13)$$

To determine the worst case network latency, we must check all the packet instances during the priority window. The maximum of these values gives the worst case network latency.

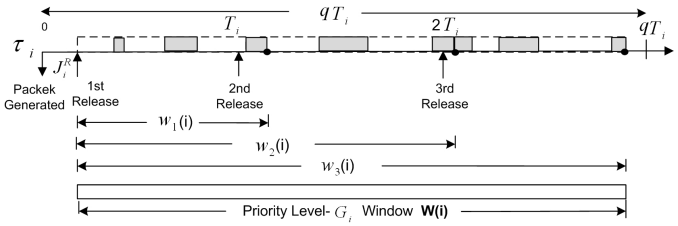

Figure 7. Priority Level-G_i window with Self-blocking

Figure 7 shows self-blocking during a priority level-G_i window. We use the index variable q to denote a packet instance of τ_i. The first packet in the window corresponds to $q = 1$ and the final one is $q = \lceil \frac{W(i)+J_i^R}{T_i} \rceil$. Therefore, the time from the first release of τ_i until achieving the q^{th} transmission is given as a collection of service requirements from all the flows which compete the priority-G_i shared resource. We assume a new time phase $w_q(i)$ which denotes the time interval from the beginning of the priority window until the completion of the q^{th} packet transmission. The time phase for the 1^{st}, the 2^{nd} and the 3^{rd} packet in the window are shown in Figure 7 as $w_1(i)$, $w_2(i)$ and $w_3(i)$ respectively. The time phase $w_q(i)$ is given by:

$$w_q(i) = qC_i + \sum_{\forall \tau_n \in S(i), \tau_n \neq \tau_i} \lceil \frac{w_q(i) + J_n^R}{T_n} \rceil C_n + \sum_{\forall \tau_j \in hp(i)} \lceil \frac{w_q(i) + R_j - C_j + J_j^R}{T_j} \rceil C_j \quad (14)$$

The variable qC_i accounts for the transmission service time of the first q packet instances of τ_i during the priority window. The final part of the right hand side of this equation includes all the service requirements from priority level G_i or higher flows which fall in this time windows $w_q(i)$. The value of $w_q(i)$ can be found by the similar iteration policy while starting with $w_q(i)^0 = C_i + qT_i$ and ending when $w_q(i)^{n+1} = w_q(i)^n$. The generation time of the q^{th} packet happens at instant $(q-1)T_i$ relative to the start of priority window so the network latency of the q^{th} instance is given by:

$$R_i(q) = w_q(i) - (q-1)T_i + J_i^R \quad (15)$$

The maximum network latency can occur at any one of these packet releases during the priority window. We will consecutively analyze each release until τ_i stops blocking itself; which means the packet transmission service finishes within the same period as it is released, $W(i) \leq qT_i - J_i^R$. Thus maximum network latency is given by:

$$R_i = \max_{q=1,\ldots,\lceil \frac{W(i)+J_i^R}{T_i} \rceil} (w_q(i) - (q-1)T_i + J_i^R) \quad (16)$$

□

Finally, flow τ_i is schedulable, if and only if $R_i \leq D_i$.

5 A case example

Revisiting the example given in Figure 2. The interrelations between these traffic-flows have been examined in section 3.2. The attributes of the traffic-flows are showed in Table 1. 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	G	T	D	J^R
τ_1	2	1	8	8	0
τ_2	2	1	11	11	0
τ_3	4	1	13	13	0
τ_4	3	2	8	12	0
τ_5	1	2	30	30	0

Table 1. Traffic-flows Description

Flows τ_1, τ_2 and τ_3 share the same priority G_1. Utilizing priority window analysis, first we calculate the $W(1)$ according to Eq.(2):

$$W(1) = \lceil \frac{W(1)}{T_1} \rceil C_1 + \lceil \frac{W(1)}{T_2} \rceil C_2 + \lceil \frac{W(1)}{T_3} \rceil C_3$$

Utilizing the iterative technique,

$$W(1)^0 = 2 + 2 + 4 = 8$$
$$W(1)^1 = \lceil \frac{8}{8} \rceil 2 + \lceil \frac{8}{11} \rceil 2 + \lceil \frac{8}{13} \rceil 4 = 2 + 2 + 4 = 8$$

The recurrence stops at $W(1)$ = 8 which is less than $min(T_n - J_n^R)$ for $\forall \tau_n \in S(1)$. So R_1=R_2=R_3 = $W(1)$ =8 less than D.

Flows τ_4 and τ_5 share the priority level G_2. The maximum window for G_2 not only considers the blocking but also the interference from higher priority flow. In this case, τ_2 and τ_3 contend for the priority level G_2 shared resources and hence contribute direct interference. Besides that, the activity of indirect higher priority flow τ_1 also can introduce some extra interference which is treated as interference jitter. So the window for G_2 is given by:

$$W(2) = \lceil \frac{W(2)}{T_2} \rceil C_2 + \lceil \frac{W(2)+R_3-C_3}{T_3} \rceil C_3 + \lceil \frac{W(2)}{T_4} \rceil C_4 + \lceil \frac{W(2)}{T_5} \rceil C_5$$

which iteratively results in $W(2) = 22$.

For τ_5, $W(2) < T_5$, so only one instance is released during this window. According to Theorem 2, $R_5 = W(2) = 22$.

For τ_4, since $\lceil \frac{W(2)}{T_4} \rceil = 3$, there are three packet instances released during the priority window. We need check all these instances to determine the worst case network latency. Utilizing Theorem 3, the window phases for the first packet, first two packets and the first three packets are $w_1(2)$, $w_2(2)$ and $w_3(2)$ respectively. Using Eq.(14) and Eq.(15), we get:

$w_1(2) = 10$ and $R_4(1) = 10 - 0 = 10$
$w_2(2) = 19$ and $R_4(2) = 19 - 8 = 11$
$w_3(2) = 22$ and $R_4(3) = 22 - 8 * 2 = 6$

The maximum network latency is thus $max(R_4(1), R_4(2), R_4(3)) = 11$. All the flows meet their deadlines and the set is schedulable.

6 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. Utilizing a priority share policy, the resource overhead can be reduced effectively. However, in order to simplify the analysis process, an existing technique imposes a deadline no more than period restriction which will bring inflexibility to network exploration and design. In this paper, we relax this constraint and present a novel analysis approach to cover all possible situations. Utilizing this new analysis scheme, we can flexibly evaluate at design time the schedulability of traffic-flow sets with different QoS requirements in a real-time 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] T. Bjerregaard and S. Mahadevan. A survey of research and practices of network-on-chip. *ACM Computer Survey*, 38(1):1, 2006.

[5] T. Bjerregaard and J. Spars. A scheduling discipline for latency and bandwidth guarantees in asynchronous network-on-chip. In *ASYNC '05: Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 34–43, 2005.

[6] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. QNoC: QoS architecture and design process for network on chip. *Journal of Systems Architecture*, 50(105-128), 2004.

[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] K. Goossens, J. Dielissen, and A. Radulescu. Aethereal network on chip: Concepts, architectures, and implementations. *IEEE Des. Test*, 22(5):414–421, 2005.

[11] 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.

[12] N. Kavaldjiev, Gerard J. M. Smith, P. G. Jansen, and P. T. Wolkotte. A virtual channel network-on-chip for GT and BE traffic. In *ISVLSI '06: Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, page 211, Washington, DC, USA, 2006. IEEE Computer Society.

[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 International Conference on Parallel Processing*, pages 527–534, 1998.

[14] 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.

[15] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In *Proceedings of the Design Automation and Test Europe Conference (DATE)*, page 20890, February 2004.

[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. 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.

[18] Z. Shi and A. Burns. Real-time communication analysis with a priority share policy in on-chip networks. In *21st Euromicro Conference on Real-Time Systems (ECRTS)*, pages 3–12, 2009.

[19] D. Wiklund and D. Liu. Socbus: Switched network on chip for hard real time embedded systems. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 78.1, Washington, DC, USA, 2003. IEEE Computer Society.

[20] P. T. Wolkotte, G. J. M. Smit, G. K. Rauwerda, and L. T. Smit. An energy-efficient reconfigurable circuit-switched network-on-chip. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 3*, page 155.1, Washington, DC, USA, 2005. IEEE Computer Society.