On-time Network On-Chip: Analysis and Architecture

Dai Bui
Alessandro Pinto
Edward A. Lee

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2009-32
http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-32.html

February 19, 2009

On-time Network On-Chip: Analysis and Architecture

Dai Bui, Edward A. Lee
Electrical Engineering & Computer Sciences
University of California, Berkeley
{daib,eal}@eecs.berkeley.edu

Alessandro Pinto
United Technologies Research Center
{pintoa}@utrc.utc.com

***Abstract*—Game, multimedia, consumer and control applications today often demand high performance computing platforms that are able to deliver real-time services while satisfying tight power constraints. Multi-core architectures, supported by on-chip networks, are emerging as scalable solutions to fulfill these requirements. However, the increasing number of concurrent applications running on these platforms and the time-varying nature of the communication requirements give rise to communication delays that are difficult to predict.**

We propose a simple and flexible instrumentation of the on-chip network that provides services to the application software for establishing end-to-end communication flows with timing guarantees. We provide a detailed analysis of the delay in a NoC using worm-hole flow control and an algorithm that, given the state of the NoC and a request to route a new real-time flow, establishes a suitable path for the route, if one exists, or rejects the request. We implemented the new protocol in a cycle-accurate simulator to validate the accuracy of the models and the correctness of the path establishment algorithm.

I. Introduction

Multi-core and possibly heterogeneous architectures are emerging as promising solution to the memory and power walls. The communication mechanism that allows these cores to communicate efficiently on the chip is a critical parameter that contributes to the optimization of the performance of these complex architectures. The Network-on-Chip (NoC) [1], [2] design paradigm addresses the communication aspect related to multi-core architectures by defining the criteria to provide a high performance, scalable and silicon-optimized interconnection fabric.

The class of software applications that could potentially leverage the availability of multiple cores on a chip is much wider than the set of applications that can be executed by a single processor. This class spans those applications that require high computation throughput, low power consumption, and, most importantly, concurrency. In fact, the resources offered by such distributed architectures are best utilized by applications that are concurrent and in general with a dynamic workload. Many such applications can be found in the embedded system domain that encompasses safety critical systems as well as consumer electronics products such as gaming consoles and mobile devices.

Common to these applications, especially for safety critical systems, is the need for real-time execution and determinism. Because in multi-core architectures communication accounts for a considerable part of the system-level delays, real-time requirements have a strong impact on the design of the NoC and on the interaction between the software application and the communication primitives. The attempt to deliver real-time guarantees by increasing the network throughput, striving to provide a zero-time communication abstraction to the application software, is not a scalable solution. Moreover, congestion on a large packet switching network can still occur resulting in unpredictable communication delays between network nodes. This non-deterministic behavior makes it very difficult to verify the timing semantics of a NoC, and as a result, the NoC design paradigm becomes unsuitable for real-time applications.

Inspired by the Precision Timed Architecture (PRET) [3] philosophy to favor timing predictability over average performance, we propose to instrument the NoC with the right mechanisms such that the application layer of the protocol stack is equipped with real-time communication primitives. This mechanisms need to be simple yet efficient and flexible enough for most of applications. This paper is organized as follows. In Section II we discuss previous results in the area of quality-of-services for NoC designs. In Section III, we argue that, despite the interesting solutions proposed in the related works, none of them seems to be optimal and we discuss the promises and the challenges of our solution. In Section IV, we present a mathematical formulation of the real-time connection establishment problem. In Section VI we show the implementation of the real-time NoC and in Section VII we evaluate its performance.

II. Related Work

Timing has always been important in a synchronous design paradigm, which is still the de-facto standard in System-on-Chip design. The introduction of a NoC as the interconnection mechanism of multiple cores on a chip, while promising from a throughput and scalability perspective, gives rise to the issue of determining the time required by a packet to reach its destination. The interesting question of whether this time could be guaranteed by the network sparked the interest of the research community in Quality-of-Service for NoC [4], [5], [6], [7].

The Æthereal network [5], developed by NXP, uses a Time Division Multi Access (TDMA) scheme for packets that

are subject to guaranteed throughput constraints. Contention is avoided in two ways: by *globally* scheduling the packet injection [8] from network nodes at network interfaces, and by using contention-free routing that solves contention by *delaying* packets at source nodes. As the number of components in a NoC grows larger, it is very difficult to find a suitable and efficient global scheduling algorithm, thereby limiting the flexibility and scalability of NoCs based on this mechanism. Furthermore, to optimize the system, the TDMA slot allocation is currently solved at *design time* [8] , resulting in a number of configurations to be loaded on the routers of the NoC. An optimization is carried out on the set of configurations, but still the tables indicating the TDMA slot assignment could be very large for large NoC. Moreover, the router optimization results in a static allocation that makes this architecture unable to exploit multiple paths from source to destination as an alternative way to avoid conflicts between guaranteed service packets. Æthereal also has some known issues with bursty traffic [9], which is typical for applications running on multi-core architectures. The fixed size of real-time packets also makes Æthereal less flexible.

In the SoCBUS architecture [7], exclusive communication paths can be reserved between a source and a destination core to provide real-time guarantees. A core wishing to reserve a path sends a set up packet first. The path will be reserved until released by the sending core, and its use is exclusive meaning that the resources that have been "blocked" for the real-time traffic cannot be used by any other packet. This approach has the following drawbacks. Consider two flows of packets with real time requirements but low throughput. If the scheduling of packets at the intermediate nodes of the path is known, it would be possible to check whether the two flows can share the same set of resources. If the analysis gives a positive answer, the two flows may be multiplexed on the same path without extra use of resources. This is not possible in SoCBUS. In additions, the resources that have been reserved for a flow cannot be used by any other flow, not even best-effort traffic that could use inactivity periods of the real-time one.

The QNoC architecture [4] supports multiple service levels (from lower priority to highest priority): Signaling, real-time, short memory read and write operations and block transfers. The QNoC architecture seems to be a good approach for soft real-time applications such as video streaming but it does not seem to be suitable for hard real-time applications. For instance, signaling packets have the highest priority and could block real-time packets. Because the number of signaling packets travelling the network is applications dependent, fixing a bound on the maximum interference between signaling packets and real-time packets becomes difficult.

The drawbacks identified in these approaches can be addressed by an architecture that keeps track of the number of real-time flows together with their real-time constraints, and that guarantees that a link is neither overloaded nor underutilized.

III. Summary of the On-time NoC Technology

The On-time NoC solution is the result of the synergy of four techniques: *admission control*, *real-time packet scheduling*, *run-time configuration* and *spatial diversity*. The configuration of the NoC is defined by the set of concurrent flows using the network resources together with their traffic characteristics and routing paths. The configuration can be stored either locally on each network nodes, or it can be managed in a centralized way by one of the processors attached to the network. Obviously, for each given network, there exists an upper bound on the number of concurrent real-time flows that can share network resources.

The first technique that we use is admission control. Borrowing from the resource reservation idea [10] commonly use in the Internet, a real-time communication between two nodes in the NoC must first reserve network resource. The admission protocol checks the configuration of the network and determines whether the request can be satisfied or not. If the request is accepted, then packets are delivered with the required maximum delay between source and destination. This mechanism avoids unpredictable behaviors when multiple real-time flows generated by a dynamic application suddenly appear on the same link exceeding its total capacity.

The second technique deals with the scheduling of packets. We want to avoid the exclusive use of network resources by one flow, preventing those resources to be shared by other real-time flows. It has been shown [11], [12] that real-time flows can be multiplexed on the same links without violating real-time requirements. In the On-time NoC, we use a fixed priority scheduling policy [13], [14] to achieve this goal. Moreover, in those cases where spare bandwidth is available, we ship best-effort flows on the reserved links.

The last two techniques that we adopt deal with the dynamic nature of the traffic that we expect from the type of applications that we target. Requests for real-time flows can be made at run-time and may require the establishment of new paths for the incoming requests. The path establishment algorithm, once executed, reconfigures the components of the NoC by updating the local state of the nodes involved in the transport of the new flow. In this way, we avoid configuring the network at design time. Because a path establishment algorithm is needed to dynamically handle real-time flow requests, we can leverage path diversity to improve the chances for a request to be accepted.

In this work, we will we assume that packets are divided into *flits* [15]. The position of a flit in a packet determines if the flit is *head, body* or *tail*. Only head flits contain routing information for packets, while body and tail flits follow head flits on the same path from sources to destinations. We also assume that the network uses worm-hole flow control. Further, in the experimental results we will use a bi-dimensional mesh topology for the NoC. However, as it will become clear from the definitions in Section IV, the results are general and apply to any topology.

IV. Problem Setting

We capture the topology of the NoC as a directed graph $G(V, E)$ where the set of vertexes V is partitioned in the set of cores C and the set of routers R, and $E \subseteq V \times V$ is the set of links.

In On-time NoC, a real-time flow is associated to a set of parameters that capture the traffic pattern and the maximum end-to-end latency that can be tolerated. A formal definition of a real-time flow is the following:

Definition 1 (Flow): A flow is a 7-tuple $(C_s, C_d, \nu, \tau, t, l, d)$ where $C_s \in C$ is the source core, $C_d \in C$ is the destination core, ν is an identifier associated with the flow and called *virtual channel*, and τ denotes the type of the packets transported by the flow. The flow is associated with the following real-time properties: $t \in \mathbb{N}$ is the minimum packet inter-arrival time in clock cycles, $l \in \mathbb{N}$ is the maximum length of a packet in number of flits, and $d \in \mathbb{N}$ is the required maximum end-to-end delay.

To avoid introducing the projection operator on tuples, for a real-time flow $f = (C_s, C_d, \nu, \tau, t, l, d)$ we refer to the elements of the tuple directly as $C_{s,f}$, $C_{s,f}$, ν_f, τ_f, t_f, l_f, d_f. Let $F = \{f_1, \ldots, f_n\}$ denote a set of flow.

We now introduce the concept of network *configuration*. Briefly, a network configuration comprises a set of flow F together with the routing information for each one of the flows. Not all configurations are valid. The first condition that a configuration must satisfy is the *connectivity constraint*. A configuration is valid if the routing path of each flow f is such that packets originating at $C_{s,f}$ arrive at destination $C_{d,f}$ along a unique path[1]. Let I be the node-edge incidence matrix of the given NoC topology. The generic element $I[v, e]$ of the incidence matrix is defined as follows:

$$I[v,e] = \begin{cases} 1 & \text{if } e \in E \text{ arrives at } v \in V \\ -1 & \text{if } e \in E \text{ depart from } v \in V \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

A routing path for a flow f is defined by a vector $\mathbf{y}_f \in \{0,1\}^{|E|}$. An entry of the vector $\mathbf{y}_f(e)$ corresponding to an edge $e \in E$, also denoted $y_{f,e}$, is defined as follows:

$$y_{f,e} = \begin{cases} 1 & \text{if flow } f \in F \text{ uses link } e \in E \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Vector $\mathbf{y}_f$ tells us what are the links in the NoC that belong to a path. To be a path from $C_{s,f}$ to $C_{d,f}$, the vector $\mathbf{y}_f$ must satisfy the following linear constraint (also known as balance equation):

$$I\mathbf{y}_f = \mathbf{b}_f, \quad \mathbf{b}_f(v) = \begin{cases} 1 & \text{if } v = C_{s,f} \\ -1 & \text{if } v = C_{d,f} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

The second constraint that must be satisfied is the *capacity constraint*. To define this constraint, we first need to make an assumption on the service time of a packet at a node of the network. We refer to the service time as to the time required by a router to send a packet on an output link. Thus, given a node u sending a packet of a flow f on a link $e = (u, v)$, we denote the service time for that packet $s_{f,e}$, which includes header processing, transmission time, and any other operations. The service time is often a function of the packet length. We assume that it takes one cycle for a router to process and send one flit over a link[2], therefore $s_{f,e} = l_f$. When multiple real time flows share the same link, we must make sure that the total link capacity is not exceeded. Thus, a valid configuration must satisfy the following constraints [11], [16]:

$$\sum_{f \in F} \frac{s_{f,e}}{t_f} y_{f,e} = \sum_{f \in F} \frac{l_f}{t_f} y_{f,e} \leq 1, \ \forall e \in E \quad (4)$$

where the multiplication by the binary variables $y_{f,e}$ makes sure that only those flows sharing the link are taken into account in determining its utilization.

The last constraint is the *end-to-end delay constraint*. Notice that the service time is different from the delay incurred by a packet when traversing a router. Depending on the scheduling policy implemented by the router, packets can wait in the input queues for a certain number of cycles. Assume we know the total delay experienced by the packets of a flow f at a node u to be sent on the output link $e = (u, v)$. As for the service time, we associate the delay with the output link of the node and denote it by $d_{f,e}$, which is the *maximum* delay of a packet of flow f on link $e \in E, e = (u, v)$ with $u, v \in V$ measured in cycles. The computation of $d_{f,e}$ is in general non-trivial. In Section V we will develop accurate models for the delay incurrend by a packet at a node.

The end-to-end delay constraint can be written in a very general form as follows:

$$\sum_{e \in E} d_{f,e} y_{f,e} \leq d_f, \ \forall f \in F \quad (5)$$

Again, the multiplication by the binary variables $y_{f,e}$ will make sure that only the delay of the links belonging to the routing path of flow f are accounted for.

Definition 2: A valid configuration of the On-Time NoC is pair $\mathcal{C}(F, Y)$, where F is a set of flow, $Y = \{\mathbf{y}_f \in \{0,1\}^{|E|} : f \in F\}$ is the set of routing paths for the flows in F, and such that constraints(3),(4) and (5) are satisfied.

The set of valid configuration of an On-Time NoC is not empty. The empty configuration, i.e. the configuration containing no flows and no paths, is trivially valid. The admission control protocol and the path establishment algorithm, which will be presented later in this paper, aim at solving the following problem:

Problem 1: Given a valid configuration (F, Y) for an On-Time NoC and a new real-time flow f, find a routing path $\mathbf{y}_f$ such that $(F \cup f\}, Y \cup \{y_f\})$ is a valid configuration.

V. System Analysis

In this section we lay down the models that we use to characterize the delay $d_{f,e}$ of a flow to traverse a router. We

[1] We do not allow multi-path routing in this work. However, similar constraints can be derived for multi-path routing.

[2] This value can vary depending on the transmission scheme. We make this assumption to simplify the explanation of our idea.

develop a cycle accurate model to precisely estimate the end-to-end delay of each flow for a given configuration. Our model is based on very few assumptions. We assume that routers are driven by a clock and we measure delays in number of clock cycles. Further, the processing of a flit is an atomic operation. We also specialize our model to the case where $s_{f,e} = l_f$. Our study follows the one in [11], [16]. However, this model is for store-and-forward flow control while we adopt wormhole flow control, which has better throughput characteristics. Therefore we will modify the computation of the delay bound accordingly.

A. Delay Bound Analysis

Several packet scheduling algorithms have been proposed in literature [17], [18], [19], [20], [12]. We employ a fixed priority non-preemptive scheduling policy for real-time packets. Priorities among packets of different flows are determined based on an ordering function $O_e : F \rightarrow \mathbb{N}$ defined for each outgoing edge $e(u, v)$ of a router u. Consider two packets p_1 and p_2, belonging to two real-time flows f_1 and f_2, respectively, and waiting to be forwarded from node u to node v (i.e. competing for the same output link $e(u, v)$). Then, packet p_i will be forwarded before p_j if and only if $O_e(f_i) < O_e(f_j)$, for $i, j = 1, 2$. The ordering function $O_e(f)$ associated with each edge allows us to assign the *priority* of each flow at a node, and decide upon the delay incurred by the packets. This mechanism can be considered as a *hybrid fixed priority* scheduling since priorities are not *globally* defined.

Consider a flow f using an edge $e(u, v)$. The delay experienced by a packet belonging to this flows is the sum of two delays: propagation delay on e, denoted by $p_{f,e}$, and the queuing delay at u, denoted $q_{f,e}$, both measured in cycles. The propagation delay is the time that elapses from when the a packet is selected to be sent at u until it arrives at v. Notice that we do not allow real-time packets to be preempted, therefore the propagation delay of a flit is also the propagation delay of a packet. In out NoC we assume that $p_{f,e} = 1 \forall e \in E$[3]. The queuing delay is the time spent by a packet in the queue of a node. Similarly to [11], we assume that all real-time flows going through a link $e(u, v)$ satisfy the following condition:

$$q_{g,e} + q_{f,e} < t_f, \ \forall f, \ g \in F \ s.t. \ y_{f,e} y_{g,e} = 1 \qquad (6)$$

This condition helps in simplifying the delay bound computation since the interval between two successive packets of each real-time flow is large enough that adding the service time of multiple packets one real-time flow is not necessary.

The delay on edge $e(u, v)$, $d_{f,e}$, is the sum of the maximum queueing delay at the sending node and the propagation delay on that edge, i.e.:

$$d_{f,e} = q_{f,e} + p_{f,e} = q_{f,e} + 1 \qquad (7)$$

We can now compute an upper bound for the queuing delay $q_{f,e}$. When a packet arrives at a node u, the worst case queuing delay that it may experience is the time required to send all higher priority packets (one packet for each higher priority flows as for condition 6), plus the maximum time to send the flits of a lower priority packet of a real-time flow that may be already in service at node u (since real-time packets are not preemptible). Therefore:

$$q_{f,e} \leq \sum_{\substack{g \in F : y_{g,e} = 1 \\ O_e(g) < O_e(f)}} s_{g,e} + \sup_{\substack{h \in F : y_{h,e} = 1 \\ O_e(h) > O_e(f)}} \left\{ s_{h,e} \frac{l_h - 1}{l_h} \right\} = \hat{q}_{f,e} \qquad (8)$$

Where we define $\sup\{\emptyset\} = 0$. The time $s_{h,e} \frac{l_h - 1}{l_h}$ is the time required to send a packet that is already in service (and therefore at least one flit must have been already sent). For example, in Figure 1, if we suppose p_i is a packet of flow f_i, and $O_e(f_3) < O_e(f_2) < O_e(f_1)$, then we can easily see that the queueing delay for packets p_1, p_2 and p_3 are 5, 6, 4, respectively, as computed by equation (8).

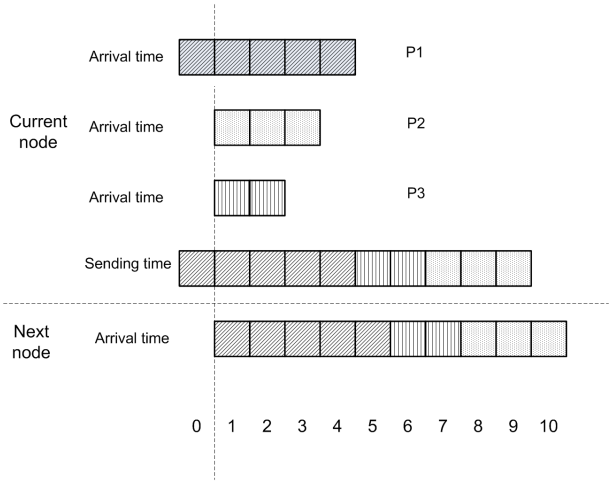


Figure 1. An example of queuing delay experienced by the packet of three flows arriving at a node and competed for the same output edge

The total delay on a path can now be computed by summing up the delays on each edge. To make our model precise, we also consider the time that is needed by the destination node to read an entire packet. Let e be the last edge of a path of a flow f, the service time at destination is $s_{f,r} = s_{f,e} \frac{l_f - 1}{l_f}$ which corresponds to reading all the remaining flits of the packet. The end-to-end delay of flow f, that must be smaller that the maximum delay admissible by a flow, can be computed as follows:

$$\sum_{\substack{e \in E \\ y_{f,e} = 1}} (\hat{q}_{f,e} + 1) + s_{f,r} \leq d_f \qquad (9)$$

Using the assumption that $s_{f,e} = l_f, \forall e \in E$, Equation 9 becomes:

$$\sum_{e \in E} \left[\sum_{\substack{g \in F : y_{g,e} = 1 \\ O_e(g) < O_e(f)}} l_g + \sup_{\substack{h \in F : y_{h,e} \\ O_e(h) > O_e(f)}} \{(l_h - 1)\} + 1 \right] y_{f,e} \leq$$
$$\leq d_f - l_f + 1, \ \forall f \in F \qquad (10)$$

[3]In pipeline router as in [21], [22], [23], some pipeline state cycles should be added to this delay

which substitutes Equation 5 with the new cycle accurate model for of the end-to-end delay.

B. Packet Scheduling Algorithm

We now describe a local scheduling algorithm that can guarantee the constraint expressed by Equation 10. We introduce the concept of *maturity* of a packet. The maturation time of a packet is the latest time a packet is expected to arrive at a node. For a given flow, let $\pi_f = (v_0, \ldots, v_{n_f})$ be the sequence of edges in the routing path of f, where $n_f = |\pi|$ is the path length. Notice that the path π_f is encoded by the binary vector $\mathbf{y}_f$. Also, let $\hat{d}_{f,e} = \hat{q}_{f,e} + 1$ be the least upper bound of the delay on a link of the path. This delay is a parameter that is stored locally in each router and that is used to estimate the maturation time of a packet. The maturation time of a packet p belonging to a flow f at a node v_n can be computed as follows:

$$m_{f,n}^{(p)} = t_p + \sum_{k=1}^{n-1} \hat{d}_{f,(v_{k-1},v_k)} \tag{11}$$

where t_p is the departure time of the header of packet p at the source of the flow v_0. A packet could arrive earlier than the maturation time due to the scheduling algorithm. Packets of real-time flows that have maturation time smaller or equal to the current time are called *mature packets*.

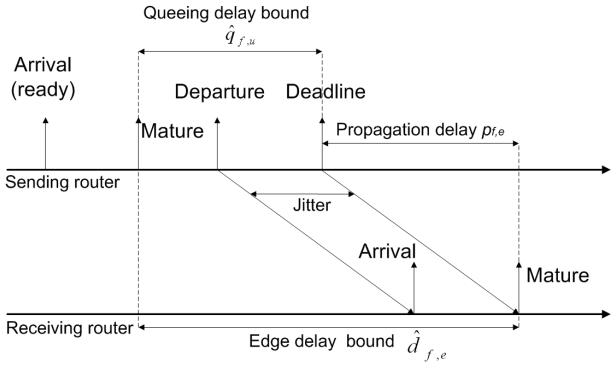


Figure 2. Timing diagram of packet forwarding

Figure 2 shows the timing diagram of the events involved in forwarding a packet. We say that a packet is *ready* when its header flit has been received. This time is also called *arrival time*. Algorithm 1 shows the algorithm that we use in each node to schedule packets. It simply forwards competing mature packets according to the ordering function. This algorithm will be implemented on each router. The reader may notice that this algorithm uses global information that are the delays on each link of the network. Later in this section we explain how the maturation time can be computed using local information only.

Theorem 1: If the constraints expressed by Equation 4 and 6) are satisfied, and all the sources of real-time flows obey their commitment on packet length and minimum packet period, the scheduling algorithm implemented by Algorithm 1 can guarantee the delay bound $\hat{q}_{f,e} + 1$ for each edge $e \in E$ and each flow $f \in F$.

Algorithm 1 Packet Scheduling Algorithm

```
1: if output edge e finishes sending a packet then
2:    Step 1:
3:    for all mature packets designated for e do
4:       find first mature packet of flow f with smallest O_e(f).
5:       forward it to edge e.
6:    end for
7:    Step 2 (optional):
8:    if no mature packet exists then
9:       select any available immature packet to forward.
10:   end if
11: end if
```

Proof: Suppose that packet p of flow f is the *first* packet that fails to be forwarded within its delay bound. Also, let m_0 be its maturation time at node with outgoing edge e. By definition of $\hat{q}_{f,e}$ the interval of time $[m_0, m_0 + \hat{q}_{f,e}]$ is sufficient to forward one packet of each flow with higher priority than f and one lower priority packet. Therefore, there must be some other flow with higher priority than f with two mature packets forwarded between m_0 and $m_0 + \hat{q}_{f,e}$. Let $g \in F$ be that flow and p_1 and p_2 are the two mature packets forwarded during in the interval $[m_0, m_0 + \hat{q}_{f,n}]$. Let m_1 and m_2 be the maturation time of p_1 and p_2, respectively, and t_1 and t_2 their departure time from the source of the flow, respectively. According to Equation 11, the following holds:

$$\begin{aligned} m_2 - m_1 = t_2 - t_1 \geq t_g \\ \implies m_2 \geq m_1 + t_g \end{aligned} \tag{12}$$

Since p_1 is forwarded within the interval $[m_0, m_0 + \hat{q}_{f,e}]$ and it does not miss its delay bound, it cannot become mature before $m_0 - \hat{q}_{g,e}$, thus:

$$m_0 - \hat{q}_{g,e} \leq m_1 \implies m_0 \leq m_1 + \hat{q}_{g,e} \tag{13}$$

Furthermore, p_2 is also mature in the interval $[m_0, m_0 + \hat{q}_{f,e}]$, meaning that the packet must have become mature before or at $m_0 + \hat{q}_{f,e}$, thus:

$$m_0 + \hat{q}_{f,e} \geq m_2 \tag{14}$$

From Equation 6, 12 and 14 we have: $m_0 + \hat{q}_{f,e} > m_1 + \hat{q}_{g,e} + \hat{q}_{f,e} \implies m_0 > m_1 + \hat{q}_{g,e}$, which contradicts the hypothesis expressed by Equation 13. □

If the optional step of the scheduling algorithm is eliminated, the delay of equal length packets of a real-time flow is the same as in Figure 5(b). This is because packets are all kept until their maturation time, thereby experiencing the same delay at each node, and consequently the same end-to-end delay. See Section VII for more details.

a) Jitter characterization.: Consider a flow f along a path π_f and let $e(v_n, v_{n+1})$ be an edge on the path. The deadline of a packet of flow f at node v_n is defined as follows:

$$T_{f,n}^{(p)} = m_{f,n}^{(p)} + \hat{q}_{f,e} \tag{15}$$

The actual sending time of the packet is denoted by $D_{f,n}^{(p)}$ and the jitter for each packet is computed as follows:

$$j_{f,n}^{(p)} = T_{f,n}^{(p)} - D_{f,n}^{(p)} \tag{16}$$

Then the maximum jitter, which is the maximum amount of time that a packet is forwarded before the deadline, is $\hat{j}_{f,n} = \sup_{p\ in\ f}\{T_{f,n}^{(p)} - D_{f,n}^{(p)}\}$ (see Figure 2 for an example). Notice that if the Step 2 in Algorithm 1 is not executed, then the following holds:

$$\hat{j}_{f,n} \leq \hat{q}_{f,n} \tag{17}$$

since we never forward immature packets.

As we anticipated in this section, we aim for a simple scheduling algorithm that can be implemented on a router by using only local information (i.e. parameter of the router or information embedded in the header of a packet). We use the jitter information as a proxy to achieve this goal. Consider the $n+1$-th node on path of a flow f. It can estimate the deadline of a packet as follows:

$$T_{f,n+1}^{(p)} = j_{f,n}^{(p)} + \hat{q}_{f,n+1} + r_{f,n+1}^{(p)} \tag{18}$$

where $r_{f,n+1}^{(p)}$ is the arrival time of packet p at the $n+1$-th node (see Figure 2 for the an intuitive explanation of this equation). We use, Equation 18 in stead of Equation 11 and 15 to compute the deadline since each router does not know the delays of other routers and links.

b) Buffer bound computation.: Because some packets belonging to a flow may be forwarded earlier than their deadlines, the buffer requirement at the receiving node is the maximum number of immature packets that may be forwarded earlier than the deadline plus the maximum number of mature packets. Under the same settings of the previous paragraph, the buffer requirement for a flow f at node v_n, denoted by $B_{f,n}$, can be lower bounded as follows:

$$B_{f,n} \geq \lceil \frac{\hat{j}_{f,n-1} + \hat{q}_{f,e}}{t_f} \rceil \tag{19}$$

Whe the option step of of Algorithm 1 is not executed, Equation 19 becomes:

$$B_{f,n} \geq \lceil \frac{\hat{q}_{f,(v_{n-1},v_n)} + \hat{q}_{f,e}}{t_f} \rceil \tag{20}$$

From Equation 6, $t_f > q_{f,u}$, therefore:

$$B_{f,n} \geq \lceil \frac{\hat{q}_{f,(v_{n-1},v_n)} + \hat{q}_{f,e}}{t_f} \rceil \leq \lceil \frac{t_f + t_f}{t_f} \rceil = 2 \tag{21}$$

meaning that the minimum buffer requirement is 2 packets or $2 \cdot l_f$ flits. The constraint expressed by Equation 6 helps in keeping the buffer requirement small, limiting the cost in terms of area and power for each real-time flow. In some cases, the buffer requirement for a flow f at one node can be even lowered to l_f as we will see in Section VII, which make the implementation of the router inexpensive.

C. New Path Establishment

In this section we derive a criterion to check if a pair $(f, \mathbf{y}_f)$ of a flow and a path is a solution to Problem 1. To arrive at a practical criterion that can be easily checked, we use a *shortest maximum packet length first* ordering function for each edge. This ordering function is defined as follows: $O_e(f) < O_e(g) \iff l_f < l_g$, meaning that shorter packets are forwarded first.

Consider a valid configuration C. According to Equation 9, we define the slack associated with a flow f as follows:

$$slack_f = d_f - s_{f,r} - \sum_{\substack{e \in E \\ y_{f,e}=1}} (\hat{q}_{f,e} + 1) \tag{22}$$

The slack of a flow is the difference between the maximum admissible delay and the actual delay resulting from the configuration. The slack is the additional amount of time that a flow can be delayed for. Consider a new flow f' and a path for it $\mathbf{y}_{f'}$ such that the connectivity constraint is satisfied. Because of the choice of shortest maximum packet length first ordering function, if flow f' were to be added to configuration C, it would only delay the packets of those flows that are longer than the packets of f'. Moreover, f' only interfere with other flows when its path uses links that are used by those flows. We can compute the new delay of a flow f with $l_f > l'_f$ as follows:

$$\sum_{\substack{e \in E \\ y_{f,e}=1}} (\hat{q}_{f,e} + 1) + s_{f,r} + \sum_{\substack{e \in E \\ y_{f,e}y_{f',e}=1}} s_{f',e} \leq d_f \tag{23}$$

The effect of adding the new flow is clear from this equation: the flow experiences an additional delay that reduces its available slack. The criteria for any routing algorithm to check if a new real-time flow can maintain the On-Time Noc configuration valid can be stated as follows:

$$\sum_{\forall e \in E} l_{f'} y_{f',e} y_{f,e} \leq slack_f,\ \forall f \in F \tag{24}$$

If this condition is verified, then the new flow can be safely added to the configuration. However, after the configuration is changed, the slacks and the delays values $\hat{d}_{f,e}$ must be updated. In Section VI we present a protocol and a network architecture to achieve this purpose.

VI. On-Time NoC Protocol and Architecture

c) Transport layer protocol.: The main focus of the definition of a protocol for NoC has typically been at the flow-control level. The network level of the protocol stack has also been considered in terms of routing that is usually static. We start the definition of our protocol at a higher level, namely the application and transport layer. In the On-Time NoC, the transport layer takes care of accepting or rejecting requests for a new real-time flow to be sent on the network. The application software is offered a `SETPATH` primitive whose specification is as follows:

From Application	From Transport
`SETPATH.request {`	`SETPATH.indication {`
`source,`	`flowID,`
`dest,`	`accept }`
`flowID`	
`t, l, d }`	

The application software issues a request to route a new real time flow to the transport layer. The request contains the addresses of the source (requesting) and destination node, an identifier associated with the flow[4], and the real-time properties of the flow, namely the minimum inter-arrival time `t`, the maximum packet length `l` and the maximum end-to-end delay `d` that the flow can tolerate. The transport layer, together with the dynamic path establishment service provided by the network layer, checks whether the flow can be supported by the network and responds with an indication that containing two fields: `accept` is a Boolean field that is true if the request has been accepted and false otherwise, and `flowID` is the identifier of the flow the indication refers to. In fact the same node may issue multiple requests at once, therefore the `flowID` field in the indication is necessary to check which request has been accepted and which one has been rejected. Table I and III show the structure of the data packets for the request and indication transport level messages while Table II shows the structure a data packet sent by the application after the real-time connection has been successfully established. In the tables, the head flit is tagged with CTRL for control packets, and HEAD for all the others.

Because resources are important when considering the protocol implementation of an NoC, we define the size of each packet field in number of bits, thereby fixing the structures of a message. The packet structure can be tailored to the specific architecture and application at hand. We consider a flit-width of 32 bits (which is a common size for the data-path of an NoC). For the packet shown in Table IV, we use 8 bits for each node ID which limits the number of nodes to 256. This number is consider sufficient for most of the current NoC. We use 2 bits for the CTRL and SETUP fields, and 5 bits for the identification of the gate, which allows for up to 32 gates for a router. Finally, we use 7 bits for the priority field, which allows for up to 128 priority levels at a router.

For the data packet shown in Table II, we use 10 bits for the jitter filed that allows for a maximum jitter value of 1024 cycles at each node. This number if sufficient for most real-time applications, considering that a higher value for the jitter would make the speed of a real-time connection no faster than a best-effort one. The jitter field is initialized to 0 at the source node.

The transport protocol that we have defined in this section is independent from the implementation of the network protocol. It is provided by the network interface that masks the interaction with the network protocol. However, the network protocol depends on the network architecture.

[4]In our implementation the `flowID` field is compute from the source and destination addresses and it is guaranteed to be unique

d) Network architecture and protocol.: The implementation of On-Time NoC could in principle be done in a distributed or centralized manner. In a distributed implementation, the configuration of the NoC is stored locally in each router and network interface. When a new request is issued by an application running on a core, a distributed routing algorithm starts flooding the network with messages to find a route for the incoming flow. The number of messages to be exchanged may be very large due to the necessity of updating the slacks. Moreover, if multiple requests are sent at the same time over the network, the concurrent execution of the distributed routing algorithm may give rise incorrect results, unless a synchronization among the requests is enforced.

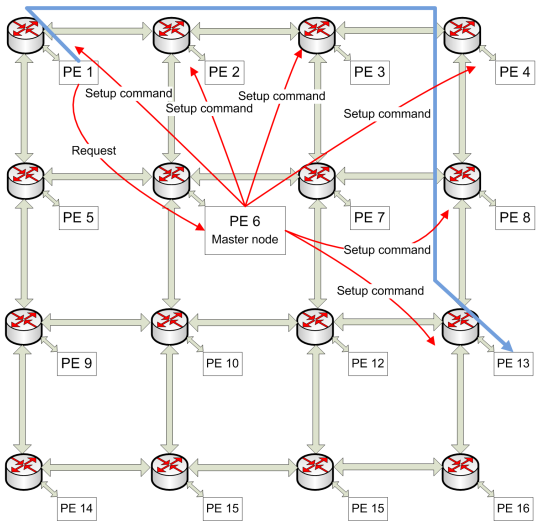


Figure 3. Example of implementation of the On-Time NoC on a 2D mesh topology: Master node and path setup procedure.

We choose to adopt a centralized architecture shown in Figure 3. Once of the node of the networks is appointed the special role of a *master node*. The master node is the only one receiving the requests of finding a routing path for an incoming flow, and responding to those requests, and therefore it also maintains an updated image of the configuration of the network. The requests are sent to the master node by the transport layer upon receiving the `SETPATH.request` call from the application software (as shown in Figure 3 by the read arrow from $PE1$ to $PE6$). Based on its knowledge about other real-time flows in the network and the demand for the new real-time flow, the master node will seek a routing path such that the new configuration is valid. If such path exists, the master node will send a *path setup* command message to each router on the path. The structure of the message is shows in Table IV. Each router will update its internal state with the following information: the input and output port of the flow, the priority associate with the flow and a delay field that is needed to compute the mature time of the packets belonging to the flow. After finishing setting up its internal configuration, each router sends back an acknowledgement message to the master node. The master node waits for the acknowledgements

CTRL	Req Node ID	Master Node ID	REQ	
BODY	Source ID	Destination ID	t_{min}	
TAIL	Flow ID		l_{max}	d_{max}

Table I
REQUEST FOR A NEW REAL-TIME FLOW

HEAD	Flow ID	Jitter	Data
BODY	Data flit		
BODY	...		
TAIL	Data flit		

Table II
REAL-TIME DATA MESSAGE

CTRL	Source ID	Destination ID	(ACK or ACCEPT)
TAIL	Flow ID		

Table III
INDICATION MESSAGE

CTRL	Master ID	Router ID	SETUP	Output gate	Priority
TAIL	Flow ID			Input gate	Delay

Table IV
PATH SETUP COMMANDS TO ROUTER

messages from all of the routers involved in the path setup procedure, and finally it sends an accept or reject message to the requesting node that issued the request. The requesting node can start sending data after receiving that indication from the transport layer that the path has been successfully established. When a path is not needed anymore, it can be released using a path-tearup protocol, which works similarly to the path-setup protocol and that we do not present for the sake of space.

The centralized architecture has several advantages. First, it can work with very little network overhead. In fact, the master node need only to communication with a limited number of routers that are the one needed by the new path (which can be computed by the node without any impact on the network traffic). Secondly, in the case where the master node is a processor, this architecture gives the flexibility of changing the routing algorithm easily and implementing complicated optimization procedure to balance the network load.

The micro-architecture of the router that we use is similarly to the one presented in [24], [12]. We also adopted some of the extensions presented in [21], [22], [23] to improve the router's performance.

e) Communication mechanism.: In Section V, we showed that the delay of the flows can be bounded given the real-time properties of the flow. This also means that the buffer requirement at each node is as discussed in Section V-B0b. Therefore, real-time packets can be sent *asynchronously*, without any need for a back-pressure acknowledgement signal, resulting is high bandwidth utilization. Conversely, best-effort packets require the use of acknowledgements from the receiving node. Figure 4 shows an example of mixed best-effort and real-time traffic communication. The `Ack` signal is needed for the best-effort traffic while a special signal is used to distinguish between best-effort and real-time. Moreover, notice that real-time packets are non-preemptible whereas best-effort ones can be preempted. Since real-time flits do not need acknowledgements they can be sent twice as fast as best-effort ones.

f) Routing algorithm.: Several routing algorithms can be implemented to optimize and balance a network on chip like this depending on application requirements. In this evaluation, we implemented an exhaustive depth-first search (DFS) routing algorithm to demonstrate the worst case delay estimation and our packet scheduling algorithm.

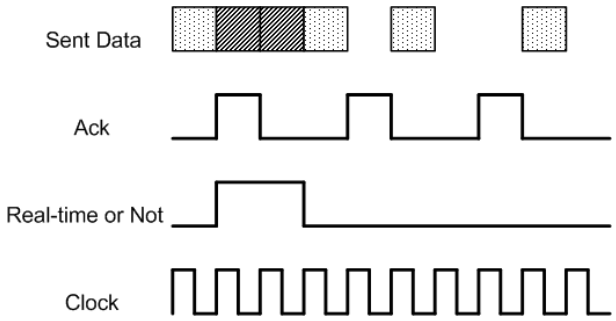


Figure 4. Example of mixed real-time and best-effort communication.

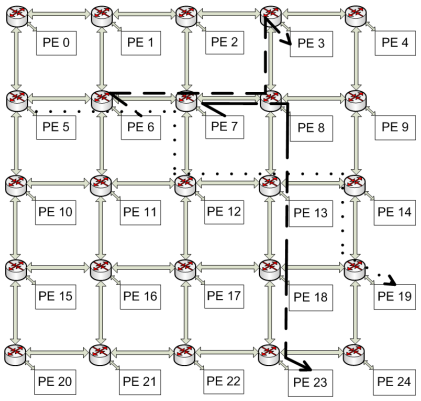
The DFS algorithm is listed in Algorithm 2. Its skeleton is the one of a DFS search with some extra conditions inserted in the code to check whether all the conditions of a valid configuration are satisfied. Also, line 7 does not specify the why in which the DFS algorithm selects a the next node to explore. Different criteria for selecting the candidate node can be used leading to different routing policy to optimize for performance, delay, power and so on. Each time node is selected, all the constraints derived in Section V are checked to verify that the new flow can be routed on that link without making the new configuration invalid.

Flow	From	To	Max Packet Length	Min Interval
1	PE 7	PE 23	5	11
2	PE 6	PE 3	3	10
3	PE 5	PE 19	4	9

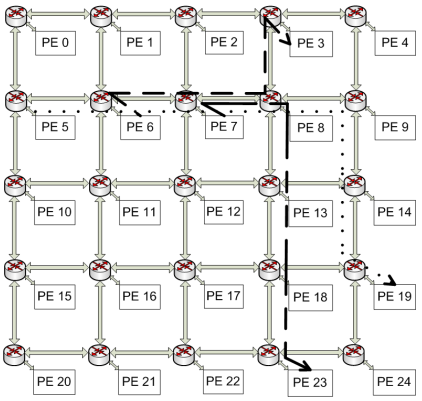
Table V
REAL-TIME FLOWS FOR THE FIRST TEST

Figure 5(a) shows an example of how the routing algorithm operates when XY routing (try X direction first) is adopted to select a candidate next node (line 7). Flow 1 request arrives first and is routed as indicated by the long-dashed path. Flow 2 arrives after and is routed as indicated by the short-dashed path. When flow 2 is routed through the link from node 7 to node 8, the delay bound of flow 1 on that edge is modified. However, the total delay bound of that flow still does not exceed the requested delay bound.

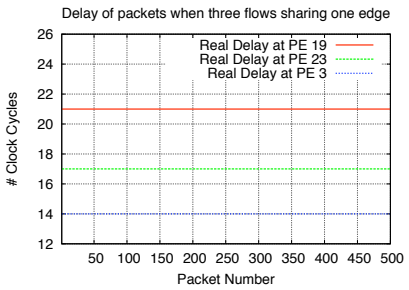
When flow 3 arrives last, it is routed to node 7. However, it cannot travel on the link from node 7 to 8 since the link



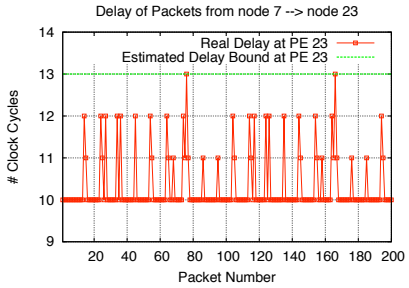
(a) Example of three flows scheduled to share links.



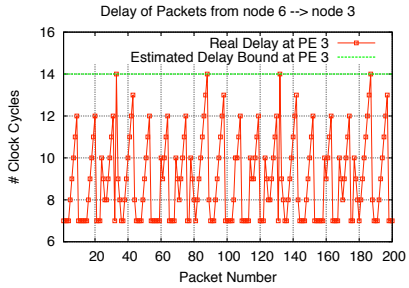
(b) Example of three flows scheduled to share one link.



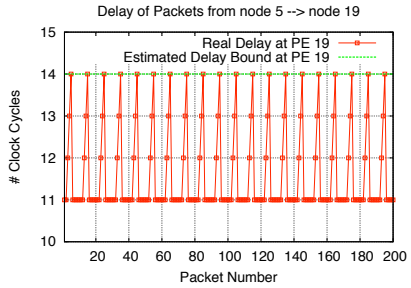
(c) Delays of packets when three flows share one link.



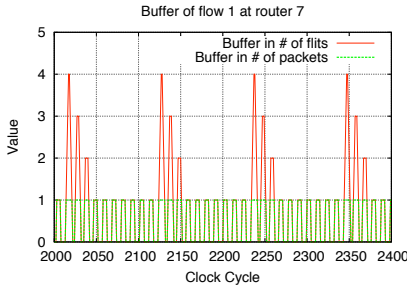
(d) Real delay and estimated delay of real-time flow from node 7 to node 23



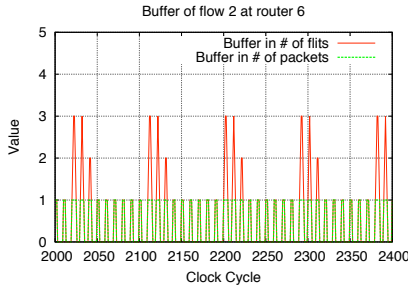
(e) Real delay and estimated delay of real-time flow from node 6 to node 3



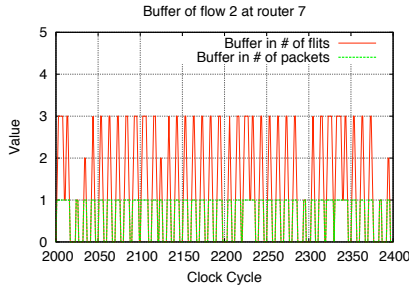
(f) Real delay and estimated delay of real-time flow from node 5 to node 19



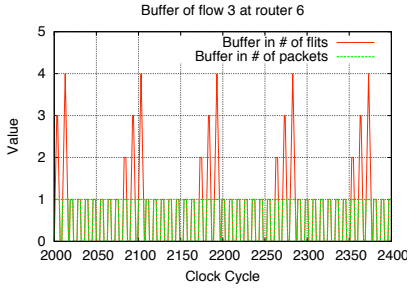
(g) Buffer usage of flow 1 at router 7



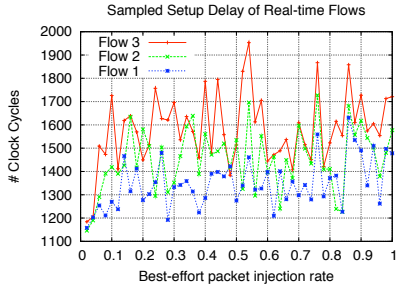
(h) Buffer usage of flow 2 at router 6



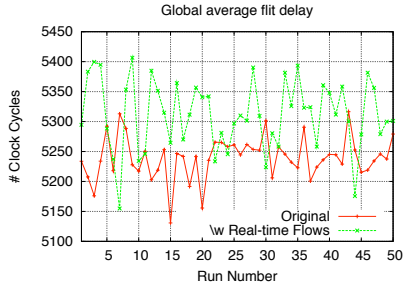
(i) Buffer usage of flow 2 at router 7



(j) Buffer usage of flow 3 at router 6



(k) Communication setup time for real-time flows



(l) Effect of real-time flows on global delay

Figure 5. Evaluation results

Algorithm 2 Routing algorithm

```
In: request (C_s, C_d, ν, τ, t, l, d)
Out: accept/reject
current_node = C_s
while current_node != C_d do
    Next[current_node] = all nodes adjacent to current_node
    if Next[current_node] ≠ ∅ then
        find a candidate next_node
        mark next_node as searched
        if (4)(20) are satisfied then
            compute the local delay for this flow
            if (6)(10)(24) are satisfied then
                for all impacted flows do
                    store the temporarily modified slacks
                    store the temporarily modified delays
                    current_node = next_node
                end for
                store local delay of this flow
            end if
        end if
        Next[current_node] = Next[current_node] \{ current_node}
    else if Next[current_node] = ∅ then
        return reject
    end if
end while
{destination node is reached}
commit slack and delays of each modified flow
commit the newly created flow's path, delays, slack
return accept
```

utilization would exceed 1. Therefore, flow 3 is routed to node 12 and finally reaches the destination node.

VII. Validation of the Models

To validate our analytical models and the implementation of the routing algorithm, we implemented the 5×5 mesh-based On-Time NoC shown in Figure 5(a). We used the Noxim simulator [25] that is based on SystemC. We stimulated the network with the scenarios shown in Figure 5(a) and 5(b), and defined in Table V and VI, respectively. This scenarios comprise three real-time flows sharing several network resources.

In this experiment, we choose a scheduling priority assignment that forwards packets of flows with smallest maximum packet length first. For two real-time flows f, g sharing an edge e, we assign priority as follows:

$$O_e(f) < O_e(g) = \begin{cases} true & \text{if } l_f < l_g \\ true & \text{if } l_f = l_g \text{ and } f \text{ is set up before } g \\ false & \text{otherwise} \end{cases} \quad (25)$$

The first set of results that we report are shown in Figure 5(d), 5(e) and 5(f). The results compare the delay of packets measured at the destinations of the flow with the delay estimated by our analysis (Section V). The measured delay is never greater than the estimated one. Moreover, since there are packets whose delay equals the estimated bound, the analysis is not too conservative. As an example, we will compute the delay upper bound for flow 2 from $PE6$ to $PE3$ using Equations 7 and 8 as follows: the maximum delay on the edge from $PE6$ to its router is 1, i.e. the propagation delay only, since there is no other flow competing for that link; the maximum delay on the edge from the router of node 6 to the router of node 7 is $(l_3-1)+1 = 4$ as computed by Equation 7 and 8, since flow 3 has lower priority; the maximum delay on the edge from node 7 to the router of node 8 is $(l_1-1)+1 = 5$; the maximum delay on the edge from the router of node 8 to the router of node 3 is 1; the maximum delay on the edge from router of node 3 to $PE3$ is 1. The time to receive the remaing flit of a packet of flow 2 is $l_2 - 1 = 2$. Summing up, we have that the delay bound is $1+4+5+1+1+2 = 14 = d_2$, which matches the required delay as shown by the simulation results in Figure 5(e). We also simulated the same scenario using the Noxim original implementation of a 5×5 mesh network, where only best-effort traffic is supported. The simulation showed that the delay of the Noxim original implementation (which turned out to be 5000 cycles in the average case) is much higher than the delay in the On-Time NoC.

Figure 5(k) shows the time required to establish a connection in different network loading condition. We randomly injected best-effort packets in the network at all routers at different rates, and we place the master node at the center of the NoC. The graph shows the *communication* time for setting up the path of the three flows defined as in Table V. The simulation shows that the best-effort packet insertion rate has low impact on the set up communication time, which seems to be strongly dependent on the number of hops from requesting nodes to the master node. This dependency also suggests that selecting the position of the master node in the network is very important and that, for large network, multiple master nodes should be used. The reason for this dependency is that best-effort control packets are treated as best-effort packets and forwarded using a Round-Robin scheduler, meaning that best-effort flows and control flows are allocated the same amount of bandwidth.

Figure 5(g), 5(h), 5(i) and 5(j) show the buffer utilization at the routers where the flows contend for the output links. In this simulations, the second step of Algorithm 1 is not executed. We observe that the number of packets in the buffers is never greater than 1. This is due to the constraint expressed by Equation 21 $q_{f,n-1} + q_{f,n} \leq t_f$ for all flows at these nodes. Therefore the maximum buffer utilization in flits is never greater than l_f for all the three tested flows as predicted by the Equation 20.

Now we change the configuration of the three flows so that they share the link from node 7 to node 8. We increase the interval between packets of each flows as in Table VI so that the total utilization of that link is not greater than 1. The three flows are then routed as in Figure 5(b). We run this configuration with the second step of the packet scheduling

Flow	From	To	Max Packet Length	Min Interval
1	PE 7	PE 23	5	21
2	PE 6	PE 3	3	19
3	PE 5	PE 19	4	17

Table VI
REAL-TIME FLOWS FOR THE SECOND TEST

Algorithm 1 removed. The result in Figure 5(c) shows that the packets of each of the three real-time flows reach their destinations at the exact estimated end-to-end delay bounds. These packets are sent at maximum packet lengths.

The last set of results show the impact of real-time connections on the global average delay of flits (Figure 5(l)). The global average delay is defined as the average of the delays of all flows measured at the destination cores. We run two configurations with and without the three real-time flows defined in Table V. The global average delay of flits is increased when real-time flows are set up on the network since real-time flows generally slow down best-effort flits sharing the same links.

VIII. Conclusion and Future Work

We presented the On-Time NoC, a Network-on-Chip for real-time applications suitable for multi-core and many-core on-chip platforms. The protocol and the network architecture of our On-Time NoC are based on a cycle accurate analytical model of the end-to-end delay of packets with priorities and worm-hole flow-control. The delay model is general with respect to packet scheduling and routing algorithm, and can be easily extended to pipeline routers. Because synchronization between processing cores can be enforced by the application software, we believe that the On-Time NoC is a good candidate to serve as the underlining communication infrastructure for some programming models such as MPI [26], [27] and PTIDES [28]. The communication time predictability offered by the On-Time NoC will allow the implementation of safety critical applications such as aircraft control, on multi-core platforms [29], [30], [31].

We also rise many research questions that remain to be answered. Firstly, acknowledgment messages may be blocked or delayed on the network, and probably should be given priority over best-effort traffic. However, their priority cannot be higher than real-time traffic. Secondly, because real-time flows have priority over best effort traffic, if the utilization of a link by real-time connections is equal or close to 1, the delay of best-effort traffic may grow exponentially. A re-routing strategy for best effort flow should be implemented to avoid this event to occur. Finally, the number and position of the master cores should be optimized.

Our next steps include identifying distributed routing algorithms that will optimize and balance a network of PRET processors [3], thus providing a real-time multi-core system. This work includes evaluating effective uses of the properties of PRET machines to enhance packet scheduling. We also plan to evaluate the buffer bound in detail for wormhole flow control. Finally we plan to design a hardware router and evaluate power and area cost.

References

[1] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *DAC '01: Proceedings of the 38th conference on Design automation*. New York, NY, USA: ACM, 2001, pp. 684–689.

[2] G. D. Micheli and L. Benini, *Networks on Chips: Technology and Tools*. Morgan Kaufmann, 2006.

[3] B. Lickly, I. Liu, S. Kim, H. D. Patel, S. A. Edwards, and E. A. Lee, "Predictable programming on a precision timed architecture," in *CASES '08: Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems*. New York, NY, USA: ACM, 2008, pp. 137–146.

[4] R. R. Dobkin, R. Ginosar, and I. Cidon, "Qnoc asynchronous router with dynamic virtual channel allocation," in *NOCS '07: Proceedings of the First International Symposium on Networks-on-Chip*. Washington, DC, USA: IEEE Computer Society, 2007, p. 218.

[5] K. Goossens, J. Dielissen, J. van Meerbergen, P. Poplavko, A. Rădulescu, E. Rijpkema, E. Waterlander, and P. Wielage, "Guaranteeing the quality of services in networks on chip," pp. 61–82, 2003.

[6] J. W. Lee, M. C. Ng, and K. Asanovic, "Globally-synchronized frames for guaranteed quality-of-service in on-chip networks," *SIGARCH Comput. Archit. News*, vol. 36, no. 3, pp. 89–100, 2008.

[7] 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*. Washington, DC, USA: IEEE Computer Society, 2003, p. 78.1.

[8] A. Moonen, C. Bartels, M. Bekooij, R. van den Berg, H. Bhullar, K. Goossens, P. Groeneveld, J. Huiskens, and J. van Meerbergen, "Comparison of an Aethereal network on chip and traditional interconnects - two case studies," in *VLSI-SoC: Research Trends in VLSI and Systems on Chip*, ser. IFIP International Federation for Information Processing, G. De Micheli, S. Mir, and R. Reis, Eds., no. 249. Springer, 2007.

[9] A. Hansson, M. Coenen, and K. Goossens, "Channel trees: Reducing latency by sharing time slots in time-multiplexed networks on chip," in *Int'l Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. ACM, Oct. 2007, pp. 149–154.

[10] L. Zhang, S. Deering, D. Estrin, and S. Shenker, "Rsvp: A new resource reservation protocol," *IEEE Network*, vol. 7, pp. 8–18, 1993.

[11] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE Journal on Selected Areas in Communications*, vol. 8, pp. 368–379, 1990.

[12] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," vol. 10, 1995, pp. 1374–1396.

[13] E. Bini, "Schedulability analysis of periodic fixed priority systems," *IEEE Trans. Comput.*, vol. 53, no. 11, pp. 1462–1473, 2004, member-Buttazzo,, Giorgio C.

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

[15] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann, 2004.

[16] D. C. Verma, H. Zhang, and D. Ferrari, "In proceedings of tricomm '91 delay jitter control for real-time communication in a packet switching network," 1991.

[17] J. C. R. Bennett and H. Zhang, "Wf 2 q: Worst-case fair weighted fair queueing," in *INFOCOM '96: Proceedings of IEEE Conference on Computer Communications*, 1996.

[18] D. D. Clark, S. Shenker, and L. Zhang, "Supporting real-time applications in an integrated services packet network: architecture and mechanism," in *SIGCOMM '92: Conference proceedings on Communications architectures & protocols*. New York, NY, USA: ACM, 1992, pp. 14–26.

[19] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *SIGCOMM '89: Symposium proceedings on Communications architectures & protocols*. New York, NY, USA: ACM, 1989, pp. 1–12.

[20] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 344–357, 1993.

[21] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha, "Express virtual channels: towards the ideal interconnection fabric," in *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*. New York, NY, USA: ACM, 2007, pp. 150–161.

[22] L.-S. Peh and W. J. Dally, "A delay model and speculative architecture for pipelined routers," in *HPCA '01: Proceedings of the 7th International Symposium on High-Performance Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2001, p. 255.

[23] ——, "A delay model for router microarchitectures," *IEEE Micro*, vol. 21, no. 1, pp. 26–34, 2001.

[24] J. Rexford, J. Hall, and K. G. Shin, "A router architecture for real-time communication in multicomputer networks," *IEEE Transactions on Computers*, vol. 47, pp. 1088–1101, 1998.

[25] F. Fazzino, M. Palesi, and D. Patti, "Noxim: Network-on-chip simulator."

[26] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the mpi message passing interface standard," *Parallel Computer.*, vol. 22, no. 6, pp. 789–828, 1996.

[27] M. Snir and S. Otto, *MPI-The Complete Reference: The MPI Core*. Cambridge, MA, USA: MIT Press, 1998.

[28] Y. Zhao, J. Liu, and E. A. Lee, "A programming model for time-synchronized distributed real-time systems," in *RTAS '07: Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 259–268.

[29] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[30] L. Lamport and P. M. Melliar-Smith, "Synchronizing clocks in the presence of faults," *J. ACM*, vol. 32, no. 1, pp. 52–78, 1985.

[31] J. M. Rushby and F. von Henke, "Formal verification of algorithms for critical systems," *IEEE Trans. Softw. Eng.*, vol. 19, no. 1, pp. 13–23, 1993.