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-2009-59
http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-59.html

May 8, 2009

Acknowledgement

This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS: PRET) and #0720841 (CSR-CPS)), the U. S. Army Research Office (ARO #W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312), the Air Force Research Lab (AFRL), the State of California Micro Program, and the following companies: Agilent, Bosch, Lockheed-Martin, National Instruments, and Toyota.

This work is also supported by Vietnam Education Foundation (VEF).

On-time Network On-Chip: Analysis and Architecture

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

Abstract— **Game, multimedia, consumer and control applications demand low power and high performance computing platforms capable of providing real-time services. 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 their communications give rise to unpredictable delays.**

We propose simple and flexible on-chip protocol and architecture that provide application level communication services with end-to-end timing guarantees. We prove the correctness of our protocol using analytical models and we validate our implementation using detailed simulations.

I. Introduction

Multi-core and possibly heterogeneous architectures allow to reduce power consumption and memory access latency by distributing workload on several cores, provided that a certain level of concurrency and data locality is present in the software applications. The communication protocol and architecture among cores is a critical parameter that affects the overall performance of these complex architectures. The Network-on-Chip (NoC) [6, 21] design paradigm defines the criteria to provide a high performance, scalable and silicon-optimized interconnection fabric for multi-core architectures.

The class of applications that could potentially leverage the availability of multiple cores on a chip spans from consumer products, such as gaming consoles and mobile devices, to safety critical systems, such as flight control. Common to these applications, especially for safety critical systems, are the dynamic nature of the computation and communication workloads, and the need for real-time and deterministic execution. Because in multi-core architectures, communications account 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 applications and the communication primitives. One may attempt to provide a zero-time communication abstraction to the application software by increasing network throughput. However, not only is this solution not scalable, but most importantly, congestion can still occur, resulting in non-deterministic delays which ultimately translate into jitter values that may be as harmful as large delays. Therefore, if real-time communications cannot be provided, the benefits promised by the NoC design paradigm may be rendered uninteresting for real-time applications.

This problem has sparked the interest of the research community in Quality-of-Service for NoC [9, 12, 18, 28]. The Æthereal network [12], 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 [22] from network nodes at network interfaces, and by using contention-free routing that solves contention by *delaying* packets at source nodes. Global scheduling limits the scalability of this approach. TDMA slot allocation is done at *design time* [22] resulting in a number of router configuration tables that, even after optimization, may be very large. Moreover, static allocation is unable to exploit path diversity to avoid congestion.

In the SoCBUS architecture [28], exclusive communication paths can be reserved between a source and a destination core to provide real-time guarantees. The network resources on a reserved path cannot be used by any other communication flow until they are released by the sending core. This solution has the drawback that the resources reserved by a low throughput real-time connection are underutilized. In principle, those resources could be used by other flows during the inactivity periods of the one that originally reserved them (as we will do in our approach.)

The QNoC architecture [9] 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 is a valid 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 traveling the network is application dependent, a precise bound on the maximum interference between signaling packets and real-time packets is hard to compute.

Summary of the On-time NoC Technology. The On-time NoC solution addresses the drawbacks identified in these previous approaches by using four techniques: *real-time packet scheduling*, *admission control*, *run-time configuration* and *spatial diversity*. The implementation of these techniques need to be simple yet efficient and flexible enough to fulfill the needs of a large set of applications.

The first technique deals with the scheduling of packets at a node. It has been shown [11, 29] 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 [2, 20] to achieve this goal. Moreover, we reallocate unused bandwidth of reserved links to best-effort traffic. The second technique is admission control for which we adopt the resource reservation [30] idea. When a source core needs to send packets with real-time guarantees to a destination core, it issues a request to reserve enough communication resources first. The request can be accepted or rejected depending on the current state of the NoC, i.e. if there are enough resources to satisfy real-time requirements. The last two techniques handle the dynamic nature of application traffic. Requests for real-time flows at run-time are followed by the establishment of routing paths for packets. The path establishment algorithm, once executed, reconfigures the components of the NoC by updating the local state of the nodes on the new paths. This technique avoids design time configuration and it can be seen as a dynamic routing algorithm. However, to best leverage path diversity and increase the chances for a request to be accepted, we run the path establishment algorithm using global network information.

Finally, we make very little assumptions on the NoC architecture. We assume that packets are divided into *flits* [7] that can be *head, body* or *tail* depending on their position in the packet. Only head flits contain routing information, while body and tail flits follow head flits on the same path from source to destination. We also assume that the network uses worm-hole flow control.

II. Preliminaries

The topology of an NoC is a directed graph $G(V, E)$ where the set of vertexes V is partitioned into two disjoint sets, the set of cores C and the set of routers R, and $E \subseteq V \times V$ is the set of links. A *real-time flow* is a tuple $f = (C_s, C_d, t, l, d)$ where $C_s \in C$ is the source core, $C_d \in C$ is the destination core, $t \in \mathbb{N}$ 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 maximum end-to-end delay. We will refer to the elements of the tuple directly as $C_{s,f}$, $C_{s,f}$, t_f, l_f, d_f. A *path* is an alternating sequence of vertexes and links $\pi = (v_0, e_0, \ldots, e_{n-1}, v_n)$, such that $e_i = (v_i, v_{i+1})$. The length of a path π is the number of links in the path. With abuse of notation, by $e \in \pi$ we mean that path π contains link e, and by $\pi_1 \cap \pi_2$ we denote the set of links and vertexes in common between two paths. The path followed by a flow f is denoted by π_f. A *configuration* of the On-Time NoC is a pair (F, Π) of a set of flows and a set of paths, one for each flow, i.e. $F = \{f_1, \ldots, f_k\}$ and $\Pi = \{\pi_{f_1}, \ldots, \pi_{f_k}\}$.

To be valid, a configuration must satisfy a number of constraints. The simplest one is the *connectivity constraint*. For a flows f and path $\pi_f = (v_0, e_0, \ldots, e_{n-1}, v_n)$, the following must hold: $v_0 = C_{s,f} \wedge v_n = C_{d,f}$. The second constraint is the *capacity constraint* defined as follows. 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 operation. 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[1], therefore $s_{f,e} = l_f$. When multiple real-time flows share the same link, its total link capacity should not exceeded. Thus, a valid configuration must satisfy the following constraint [11, 27]:

$$\sum_{f \in F : e \in \pi_f} \frac{s_{f,e}}{t_f} = \sum_{f \in F : e \in \pi_f} \frac{l_f}{t_f} \leq 1, \ \forall e \in E \qquad (1)$$

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. As for the service time, we associate the delay experienced by the packets of a flow f at u, and waiting to be forwarded to the output link e, with the output link itself, and we denote the delay by $d_{f,e}$. The end-to-end delay constraint can be written as follows:

$$\sum_{e \in \pi_f} d_{f,e} \leq d_f, \ \forall f \in F \qquad (2)$$

The computation of $d_{f,e}$ is in general non-trivial and it is the subject of Section III-A. In Section IV, we derive a criterion for preserving the validity of a configuration when a new flow if added to F and its routing path is added to Π.

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

III. Real-time packet scheduling

The delay $d_{f,e}$ experienced by a packet at a node of the NoC depends on the packet scheduling algorithm. Several packet scheduling algorithms have been proposed in literature [1, 4, 8, 23, 29]. We use a fixed priority non-preemptive scheduling policy and we follow the methodology described in [11, 27]. However, this model is for store-and-forward flow control while we use worm-hole flow control, which has better throughput characteristics. Therefore, we will modify the computation of the delay bound accordingly.

A. Computation of the delay bound.

Priorities among packets are defined by a function $O_e : F \rightarrow \mathbb{N}$ that induces a total ordering on the flows. The ordering function may be different for each link $e \in E$ and does not have to be globally defined. Consider two packets p_1 and p_2 belonging to two real-time flows f_1 and f_2, respectively, and competing at u for the 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 delay $d_{f,e}$ experienced by a packet is the sum of a propagation delay $p_{f,e}$, and a queuing delay $q_{f,e}$, both measured in cycles. The propagation delay is the amount of time that elapses from the selection of a packet to be forwarded, to the arrival of the same packet at the next node. We assume that $p_{f,e} = 1 \forall e \in E$[2]. Since real-time packets cannot be preempted, the propagation delay of a flit is also the propagation delay of a packet:

$$d_{f,e} = q_{f,e} + p_{f,e} = q_{f,e} + 1 \tag{3}$$

The upper bound on the queuing delay, denoted by $\hat{q}_{f,e}$, is intuitively computed as the sum of the service times of all higher priority packets that may be present in the input queues of the router. The computation is simplified by assuming that, for each higher priority flow, at most one packet is waiting in the queues. This condition can be written as follows:

$$\hat{q}_{g,e} + \hat{q}_{f,e} < t_f, \ \forall f, \ g \in F \ s.t. \ e \in \pi_f \cap \pi_g \tag{4}$$

Under this assumption, the following expression for the queuing delay upper bound holds:

$$\hat{q}_{f,e} = \sum_{\substack{g \in F : e \in \pi_g \\ O_e(g) < O_e(f)}} s_{g,e} + \sup_{\substack{h \in F : e \in \pi_h \\ O_e(h) > O_e(f)}} \left\{ s_{h,e} \frac{l_h - 1}{l_h} \right\} \tag{5}$$

[2]In pipeline routers [15, 24, 25], some extra cycles should be added to this delay

Where we define $\sup\{\emptyset\} = 0$. This equation shows that the worst case queuing delay that a packet may experience is the time required to send all higher priority packets (one packet for each higher priority flow as for condition 4), plus the maximum time required to send the flits of a lower priority packet of a real-time flow that may be already in service. This time is $s_{h,e}\frac{l_h-1}{l_h}$ since 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 queuing delay upper bounds for packets p_1, p_2 and p_3 are 5, 6, 4, respectively, as computed by Equation 5.

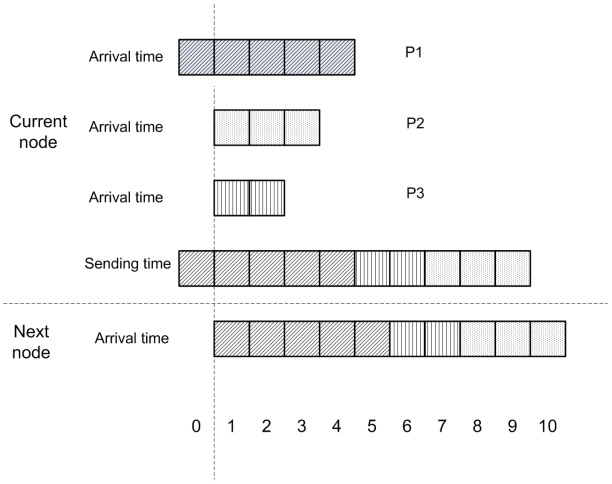


Fig. 1
An example of queuing delay experienced by the packet of three flows arriving at a node and competing for the same output edge

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. Using the assumption that $s_{f,e} = l_f, \forall e \in E$, Equation 2 becomes:

$$\sum_{e \in \pi_f} \left[\sum_{\substack{g \in F : e \in \pi_g \\ O_e(g) < O_e(f)}} l_g + \sup_{\substack{h \in F : e \in \pi_h \\ O_e(h) > O_e(f)}} \{(l_h - 1)\} + 1 \right] \leq$$
$$\leq d_f - l_f + 1, \ \forall f \in F \tag{6}$$

B. Packet Scheduling Algorithm

To satisfy the delay bound of Equation 6, the condition in Equation 4 must hold at all nodes in the network. The

scheduling algorithm presented in this section ensures that such condition is satisfied.

Let the maturation time of a packet be the latest time a packet is expected to arrive at a node. Also, let $\hat{d}_{f,e} = \hat{q}_{f,e} + 1$ be the upper bound of the delay on link $e(u,v)$. This delay is a parameter that is stored locally at u and that is used to estimate the maturation time of a packet. The maturation time of a packet p of flow f at node v_k (along path $\pi_f = (v_0, \ldots, v_k, \ldots, v_n)$) can be computed as follows:

$$m_{f,k}^{(p)} = t_p + \sum_{i=1}^{k-1} \hat{d}_{f,(v_{i-1},v_i)} \tag{7}$$

where t_p is the departure time of the head flit of p at the source v_0. Packets of real-time flows that have maturation time smaller or equal to the current time are called *mature packets*.

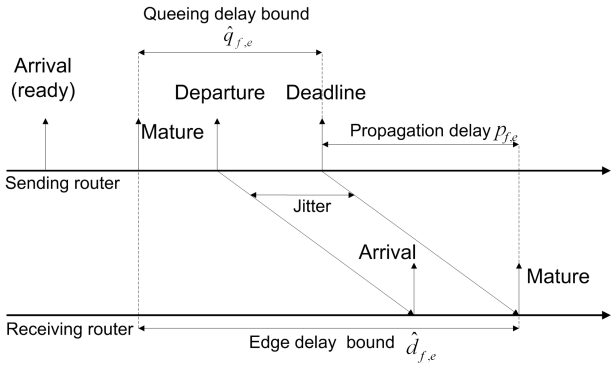


Fig. 2
TIMING DIAGRAM OF PACKET FORWARDING

Figure 2 shows the timing diagram of the events involved in forwarding a packet. A packet is *ready* when its head flit has been received. This is called the packet's *arrival time*. It becomes mature at the maturation time, and it must be forwarded before its deadline, which is the sum of the maturation time plus the upper bound on the queuing delay. A packet departs a node when its head flit is forwarded. Algorithm 1, implemented at each node, schedules competing mature packets according to the ordering function O_e.

Theorem 1: For a valid configuration (F, Π), if each source $C_{s,f}$ satisfies Equation 4 for each flow $f \in F$, then Algorithm 1 guarantees the queuing delay bound $\hat{q}_{f,e}$ for each edge $e \in E$ and each flow $f \in F$.

Proof: Suppose that packet p of flow f is the *first* packet that fails to be forwarded within its queuing delay bound. Also, let m_0 be its maturation time at node with outgoing edge e. From Equation 4, we can see that any previous

Algorithm 1 Packet Scheduling Algorithm at node u

Require: current time t
  for all $e(u,v) \in E$ **do**
    if e is idle **then**
      $P \leftarrow \emptyset$
      for all $f \in F$ s.t. $e \in \pi_f$ with a packet p in queue **do**
        if $m_{f,u}^{p} \leq t$ **then**
          insert p in P
        end if
      end for
      forward $p \in P$ with highest priority according to O_e
      [Optional step (for work-conservation)]
      if p does not exist **then**
        pick any immature packet to forward
      end if
    end if
  end for

packets of the flow f have been forwarded before m_0 since $t_f > \hat{q}_{f,e}$, which means that the interval between any previous packet and p is greater than the previous packet's maximum queuing delay. 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 thas has higher priority than f and the remaining flits of one lower priority packet plus the head flit of p since the length of the interval is $\hat{q}_{f,e} + 1$ cycles. Therefore, there must be some other flow that has higher priority than f with two mature packets forwarded within $[m_0, m_0 + \hat{q}_{f,e}]$, otherwise, the head flit of p must have been forwarded before or at $m_0 + \hat{q}_{f,e}$ and the packet p does not miss its deadline. Let $g \in F$ be that flow and p_1 and p_2 are the two mature packets forwarded during in the interval $[m_0, m_0 + \hat{q}_{f,e}]$. 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 7, the following holds:

$$\begin{aligned} m_2 - m_1 = t_2 - t_1 \geq t_g \\ \Rightarrow m_2 \geq m_1 + t_g \end{aligned} \tag{8}$$

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:

$$\begin{aligned} m_0 - \hat{q}_{g,e} \leq m_1 \\ \Rightarrow m_0 \leq m_1 + \hat{q}_{g,e} \end{aligned} \tag{9}$$

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:

$$m_0 + \hat{q}_{f,e} \geq m_2 \tag{10}$$

From Equation 4, 8 and 10 we have: $m_0 + \hat{q}_{f,e} \geq m_2 \geq m_1 + t_g > m_1 + \hat{q}_{g,e} + \hat{q}_{f,e} \Rightarrow m_0 > m_1 + \hat{q}_{g,e}$, which contradicts the hypothesis expressed by Equation 9. □

To compute the maturation time of a packet, the scheduling algorithm uses the delay upper bound of each link of the path followed by the packet. These values are global information that we want to avoid storing at each node. We use the *jitter* as a proxy to compute the maturation time (see Figure 2).

C. Jitter and buffering size.

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

$$T_{f,n}^{(p)} = m_{f,n}^{(p)} + \hat{q}_{f,e_n} \tag{11}$$

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

$$j_{f,n}^{(p)} = T_{f,n}^{(p)} - D_{f,n}^{(p)} \tag{12}$$

The jitter measures the length of time from the departure of a packet to its deadline. Denoting by $r_{f,n+1}^{(p)}$ the arrival time of packet p at node v_{n+1}, the following holds: $m_{f,n+1}^{(p)} = r_{f,n+1}^{(p)} + j_{f,n}^{(p)}$. Combining this last equation with Equation 11 and 12, we obtain:

$$j_{f,n+1}^{(p)} = r_{f,n+1}^{(p)} + j_{f,n}^{(p)} + \hat{q}_{f,e_{n+1}} - D_{f,n+1}^{(p)} \tag{13}$$

This equation suggests that the maturation time of a packet can be computed using only two local parameters: the jitter from the previous node, and the upper bound on the queuing delay. Moreover, these parameters are used to compute the jitter value to be sent to the next node on the path. Using the jitter information allows to have a completely distributed scheduling algorithm.

The buffer requirement at a node is the maximum number of immature packets that may be forwarded to the node earlier than the deadline plus the maximum number of mature packets. The buffer requirement for a flow f at node v_n, denoted by $B_{f,n}$, can be lower bounded as follows:

$$B_{f,n} \geq \lceil \frac{\hat{j}_{f,n-1} + \hat{q}_{f,e_n}}{t_f} \rceil \tag{14}$$

If the optional step of Algorithm 1 is not executed, meaning that immature packets are never forwarded, then Algorithm 1 guarantees that for any packet p of flows f and any node n on the path π_f the following upper bound holds true:

$$j_{f,n}^{(p)} \leq \hat{q}_{f,e_n} \tag{15}$$

Using this jitter upper bound, a safe value for the buffer size at node v_n is:

$$B_{f,n} \geq \lceil \frac{\hat{q}_{f,e_{n-1}} + \hat{q}_{f,e_n}}{t_f} \rceil \tag{16}$$

From Equation 4 we have that $t_f > \hat{q}_{f,e}, \ \forall e \in E$, therefore a safe upper bound for the buffer size is:

$$\lceil \frac{\hat{q}_{f,e_{n-1}} + \hat{q}_{f,e_n}}{t_f} \rceil \leq \lceil \frac{t_f + t_f}{t_f} \rceil = 2 \tag{17}$$

meaning that the minimum buffer requirement is 2 packets (or $2 \cdot l_f$ flits.) This upper bound is conservative. As we will see in Section VI, in some cases the buffer requirement can be lowered to l_f, which leads to an inexpensive router implementation.

D. Communication mechanism.

Based on the analysis presented in Section III, we can bound the buffer requirements at each node. Therefore, real-time packets can be sent *asynchronously*, without any need for back-pressure, resulting in high bandwidth utilization. Instead, best-effort packets require the use of acknowledgments from the receiving node. Figure 3 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 acknowledgments they can be sent twice as fast as best-effort ones.

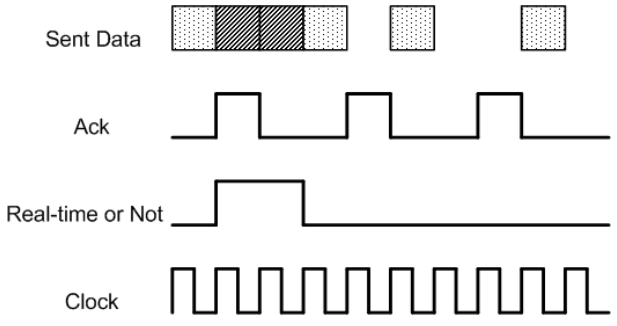


Fig. 3
EXAMPLE OF MIXED REAL-TIME AND BEST-EFFORT COMMUNICATION.

IV. Admission control and new path establishment

Consider a network in a valid configuration (F, Π). A new request from a source core to establish a real-time connection to a destination core entails answering the following question: given a flow f' is there a path $\pi_{f'}$ such that $(F \cup \{f'\}, \Pi \cup \{\pi_{f'}\})$ is still a valid configuration?

According to Equation 6, we define the slack associated with a flow f as follows:

$$slack_f = d_f - s_{f,r} - \sum_{e \in \pi_f} (\hat{q}_{f,e} + 1) \tag{18}$$

The slack of a flow is the difference between the maximum admissible delay and the actual delay resulting from the configuration, i.e. the amount of time that a flow can be delayed for, without violating its real-time requirements. Consider a new flow f' and a path $\pi_{f'}$ such that the connectivity constraint is satisfied. Flow f will suffer the interference from flow f' at all those links that are in common between the two paths and for which f' has higher priority. The total interference must be smaller than the slack of flow f, i.e. :

$$\sum_{\substack{e \in \pi_f \cap \pi_{f'} \\ O_e(f') < O_e(f)}} s_{f',e} \leq slack_f \tag{19}$$

If this condition is satisfied by the new configuration, and if path $\pi_{f'}$ also satisfies the delay constraint in Equation 6, then the new configuration is valid. If such path cannot be found, then the request is rejected and must be re-issued by the application at a later time. It is interesting to notice that there is room for a joint optimization of the priority assignment of each flow at each link and the routing algorithm. Once the new flow and path pair is added to the configuration, the nodes belonging to the new path must be updated with their new queuing delays.

V. On-Time NoC Protocol and Architecture

A. Transport layer protocol.

In the On-Time NoC, the transport layer is in charge of accepting or rejecting requests to establish new real-time connections. The transport layer implements the following `SETPATH` primitive:

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 and destination nodes, an identifier associated with the flow[3], 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 added to the network and responds with an indication that contains 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. Table I and III show the structure of the request and indication messages, respectively, while Table II shows the structure of 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.

We consider a flit-width of 32 bits (which is a common size for the data-path of an NoC). We use 8 bits for each node ID, 2 bits for the CTRL and SETUP fields, 5 bits for the identification of the gate, 7 bits for the priority field, and 10 bits for the jitter.

B. 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, when a new request is issued, the 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 associated with each flow. Moreover, the concurrent execution of the distributed routing algorithm requires synchronization among the requests to decide the order in which they are going to be satisfied.

We choose to adopt a centralized architecture shown in Figure 4. One of the nodes of the network is appointed the special role of a *master node*. The master node is the only one receiving requests and responding to them. 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 4 by the red arrow from $PE1$ to $PE6$). The master node implements the admission control and routing algorithm to find suitable paths for new flows. When a path exists, the master node sends a *path setup* command message to each router on the path. The structure of the message is shown in Table IV. Each router will update its internal state with the

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

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

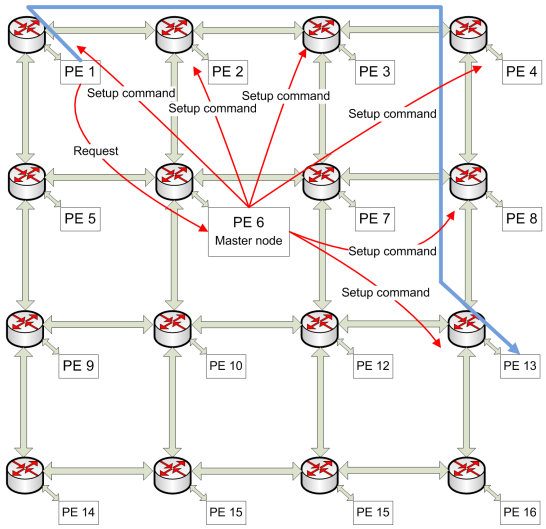


Fig. 4
EXAMPLE OF IMPLEMENTATION OF THE ON-TIME NOC ON A 2D MESH TOPOLOGY: MASTER NODE AND PATH SETUP PROCEDURE.

following information: the input and output ports for the flow, the flow priority and a delay field that is needed to compute the maturation time of the packets belonging to the flow. After finishing setting up its internal configuration, each router sends back an acknowledgment message to the master node. The master node waits for the acknowledgment messages from all of the routers involved in the path setup procedure, and finally sends an accept or reject message to the node that issued the request. The node can start sending data packets after receiving the indication from the transport layer that the path has been successfully established. When a path is no longer needed, it can be released using a path-tear-up protocol.

The centralized architecture requires very little network overhead. In fact, the master node need only to communicate with a limited number of routers that are the ones needed by the new path. In principle, multiple master nodes may be present in an On-Time NoC.

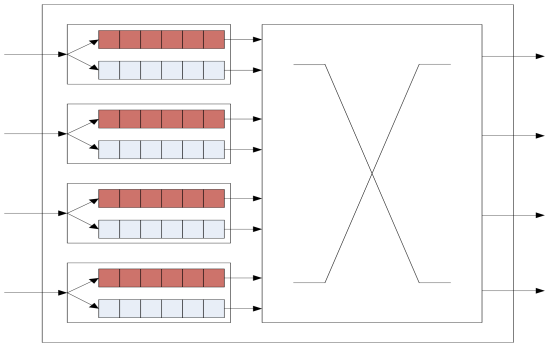


Fig. 5
A SAMPLE ROUTER ARCHITECTURE.

C. Proposed Router Architecture

The router architecture that we use is shown in Figure 5. It is similar to the one presented in [14] where each input port has a number of virtual channel queues [5]. When a real-time flow is assigned to an input port, one of the queues is reserved for the flow. When not reserved, the queues can be used as regular virtual channels for best-effort traffic that can benefit from higher router's performance. We do need changes in the control logic to support the packet scheduling algorithm, but these changes have minimal impact on the area and power overhead. Given the similarity of the On-Time NoC router micro-architecture with standard routers, it is easy to incorporate some of the extensions presented in [15, 24, 25] to improve routers' performance.

D. Routing algorithm

Several routing algorithms can be implemented to optimize and balance the load of the On-Time NoC. The routing algorithm for On-Time NoC is always *deadlock*

free since sending nodes do not need to wait to send a packet. The protocol that we defined is in fact independent from the routing algorithm. In our experiments, we implemented an exhaustive modified depth-first search (DFS) routing algorithm. The skeleton of the algorithm is the one of a classical DFS with some extra conditions inserted in the code to maintain a network configuration valid.

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 6(a) shows an example of how the routing algorithm would operate under an XY routing strategy (try X direction first). The first request is for Flow 1 that is routed as indicated by the long-dashed path. The second request is for Flow 2 that 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 the request for 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 utilization would exceed 1. Therefore, Flow 3 is routed to node 12 and finally reaches its destination.

VI. EXPERIMENTAL RESULTS

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 6(a). We used the Noxim simulator [10] that is based on SystemC. We simulated the network with the scenarios shown in Figure 6(a) and 6(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:

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

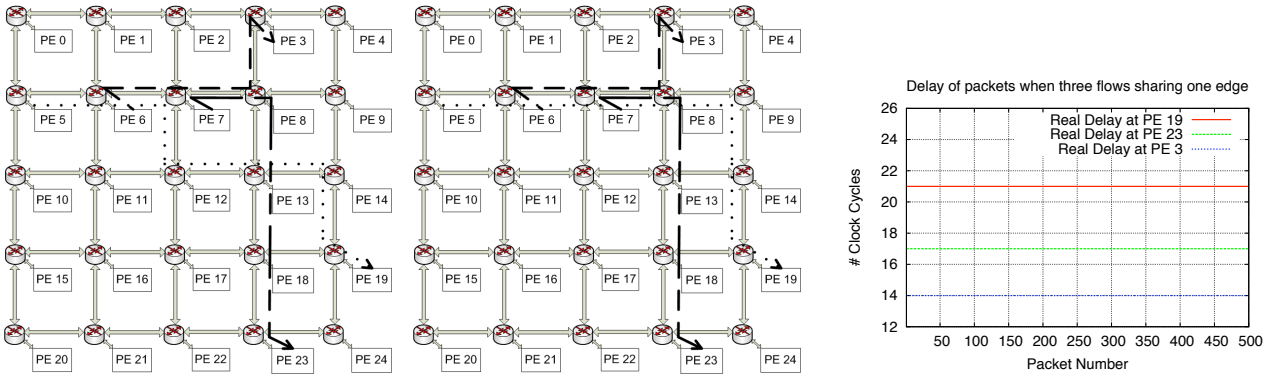
The first set of results with traffic characteristics in Table V that we report are shown in Figure 6(d), 6(e) and 6(f) when the optional step in the packet scheduling algorithm is executed. The results compare the delay of packets measured at the destinations of the flow with the delay estimated by our analysis (Section III). 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 3 and 5 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 3 and 5, 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 remaining flits 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 6(e).

Figure 6(g), 6(h) and 6(i) show the buffer utilization at the routers where the flows contend for the output links without executing the optional step in the packet scheduling algorithm. We observe that the number of packets in the buffers is never greater than 1. This is due to the constraint expressed by Equation 16: $\hat{q}_{f,e_{n-1}} + \hat{q}_{f,e_n} \leq t_f$ 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 16.

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

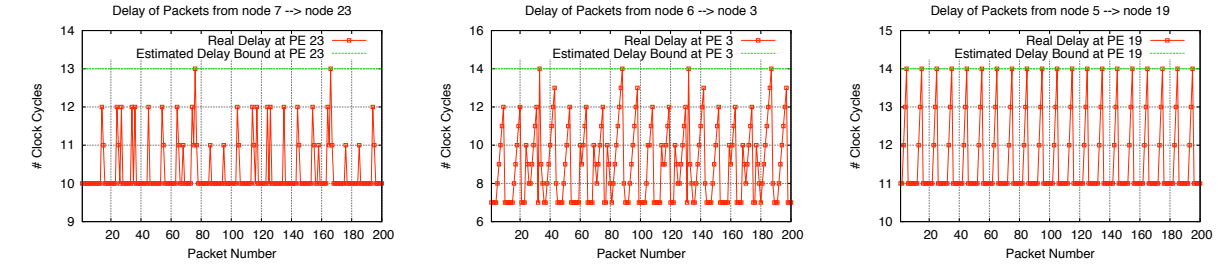
Now we change the configuration of the three flows so that they share the link from node 7 to node 8. We increase the interval between packets of each flows as in Table VI so that the total utilization of that link is not greater than 1. The three flows are then routed as in Figure 6(b). The result in Figure 6(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 since all jitters are absorbed when the optional step in the packet scheduling algorithm is not executed. These packets are sent at maxi-



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

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

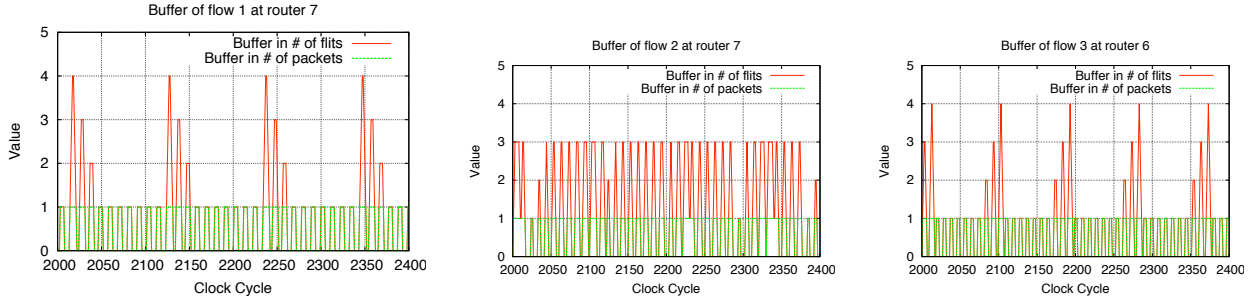
(c)Delays of packets when three flows share one link.



(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 7

(i)Buffer usage of flow 3 at router 6

Fig. 6
EVALUATION RESULTS

mum packet lengths.

A. *Comparison with other NoC protocols*

We compared the On-Time NoC with existing architectures. The results are shown in Table VII with values expressed in number of cycles. For this experiment, we use two different network configurations: a 5x5 mesh network with transpose traffic pattern and a 8x8 mesh network with butterfly traffic pattern. We assume that packets are 5 flits long and that they are periodically injected in the network. We vary the period to change the packet injection rate. We define the saturation rate as the maximum packet injection rate after which we observe no delays higher than 1000 cycles. We run simulations for 200000 cycles and we measure the maximum and average delay of all flows in the network. We also note that while validation of standard NoC can only be done empirically by processing simulation traces, results for the On-time NoC can be derived analytically.

To make the comparison fair, we use the same total buffer size of 10 flits (i.e. 2 packets) for each input port of each router in all cases. For the On-Time NoC, this buffer size corresponds to a maximum of two real-time flows for each link. We compare the performance of the On-Time NoC against two other architectures. The centralized one is an NoC with standard routers in Noxim [10], but with

Traffic	Protocol	Saturation Rate	Max delay	Average delay
Transpose (5x5)	OT NoC	0.1	34	19.64
	Centralized	0.033	61	23.48
	DyAd	0.022	150	22.47
Butterfly (8x8)	OT NoC	0.1	60	21.03
	Centralized	0.025	76	28.70
	DyAd	0.01	314	37.80

TABLE VII
COMPARISON WITH OTHER ARCHITECTURES.

a centralized routing algorithm that routes flows trying to avoid congestion [4]. This architecture can be seen as the On-Time NoC without the mechanisms that we develop to guarantee real-time packet delivery. The other architecture used DyAd [13] as a dynamic routing algorithm. The results show that the On-Time NoC is able to handle higher injection rate (three to five times for transpose traffic and four to ten times for butterfly traffic). The maximum delay is two to five time smaller while the average delay is 12 % to 45 % smaller than the other two architectures.

VII. Conclusion and Future Work

We presented the On-Time NoC, a Network-on-Chip for real-time applications suitable for multi-core 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 PTIDES [31]. 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 [16, 17, 26].

Our next steps include identifying distributed routing algorithms that will optimize and balance a network of PRET processors [19], thus providing a real-time multi-core system. This work includes evaluating effective uses of the properties of PRET machines to enhance packet scheduling and an optimization procedure to define the number and position of master nodes. We also plan to evaluate the buffer bound in detail for worm-hole flow control. Finally we plan to design a hardware router and evaluate power and area costs.

[4] We use the method of forbidden turns in [3] to avoid deadlocks.

References

[1] Jon C. R. Bennett and Hui Zhang. Wf 2 q: Worst-case fair weighted fair queueing. In *INFOCOM '96: Proceedings of IEEE Conference on Computer Communications*, 1996.

[2] Enrico Bini. Schedulability analysis of periodic fixed priority systems. *IEEE Trans. Comput.*, 53(11):1462–1473, 2004. Member-Buttazzo,, Giorgio C.

[3] Ge-Ming Chiu. The odd-even turn model for adaptive routing. *IEEE Trans. Parallel Distrib. Syst.*, 11(7):729–738, 2000.

[4] David D. Clark, Scott Shenker, and Lixia Zhang. Supporting real-time applications in an integrated services packet network: architecture and mechanism. In *SIGCOMM '92: Conference proceedings on Communications architectures & protocols*, pages 14–26, New York, NY, USA, 1992. ACM.

[5] William J. Dally. Virtual-channel flow control. *SIGARCH Comput. Archit. News*, 18(3a):60–68, 1990.

[6] William J. Dally and Brian Towles. Route packets, not wires: on-chip interconnection networks. In *DAC '01: Proceedings of the 38th conference on Design automation*, pages 684–689, New York, NY, USA, 2001. ACM.

[7] William James Dally and Brian Patrick Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, San Francisco, CA, USA, 2004.

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

[9] Rostislav (Reuven) Dobkin, Ran Ginosar, and Israel Cidon. Qnoc asynchronous router with dynamic virtual channel allocation. In *NOCS '07: Proceedings of the First International Symposium on Networks-on-Chip*, page 218, Washington, DC, USA, 2007. IEEE Computer Society.

[10] Fabrizio Fazzino, Maurizio Palesi, and Davide Patti. Noxim: Network-on-chip simulator.

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

[12] Kees Goossens, John Dielissen, Jef van Meerbergen, Peter Poplavko, Andrei Rădulescu, Edwin Rijpkema, Erwin Waterlander, and Paul Wielage. Guaranteeing the quality of services in networks on chip. pages 61–82, 2003.

[13] Jingcao Hu and Radu Marculescu. Dyad: smart routing for networks-on-chip. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 260–263, New York, NY, USA, 2004. ACM.

[14] N.K. Kavaldjiev, Dr.ir. G.J.M. Smit, and Ir. P.G. Jansen. A virtual channel router for on-chip networks. In *IEEE International SOC Conference*, pages 289–293. IEEE Computer Society Press, 2004.

[15] Amit Kumar, Li-Shiuan Peh, Partha Kundu, and Niraj K. Jha. Express virtual channels: towards the ideal interconnection fabric. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 150–161, New York, NY, USA, 2007. ACM.

[16] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[17] Leslie Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *J. ACM*, 32(1):52–78, 1985.

[18] Jae W. Lee, Man Cheuk Ng, and Krste Asanovic. Globally-synchronized frames for guaranteed quality-of-service in on-chip networks. *SIGARCH Comput. Archit. News*, 36(3):89–100, 2008.

[19] Ben Lickly, Isaac Liu, Sungjun Kim, Hiren D. Patel, Stephen A. Edwards, and Edward 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*, pages 137–146, New York, NY, USA, 2008. ACM.

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

[21] Giovanni De Micheli and Luca Benini. *Networks on Chips: Technology and Tools*. Morgan Kaufmann, 2006.

[22] Arno Moonen, Chris Bartels, Marco Bekooij, René van den Berg, Harpreet Bhullar, Kees Goossens, Patrick Groeneveld, Jos Huiskens, and Jef van Meerbergen. Comparison of an Aethereal network on chip and traditional interconnects - two case studies. In Giovanni De Micheli, Salvador Mir, and Ricardo Reis, editors, *VLSI-SoC: Research Trends in VLSI and Systems on Chip*, number 249 in IFIP International Federation for Information Processing. Springer, 2007.

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

[24] Li-Shiuan Peh and William 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*, page 255, Washington, DC, USA, 2001. IEEE Computer Society.

[25] Li-Shiuan Peh and William J. Dally. A delay model for router microarchitectures. *IEEE Micro*, 21(1):26–34, 2001.

[26] J. M. Rushby and F. von Henke. Formal verification of algorithms for critical systems. *IEEE Trans. Softw. Eng.*, 19(1):13–23, 1993.

[27] Dinesh C. Verma, Hui Zhang, and Domenico Ferrari. Delay jitter control for real-time communication in a packet switching network. In *In Proceedings of TriComm '91*, 1991.

[28] Daniel Wiklund and Dake 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.

[29] Hui Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of the IEEE*, 10:1374–1396, 1995.

[30] Lixia Zhang, Steve Deering, Deborah Estrin, and Scott Shenker. Rsvp: A new resource reservation protocol. *IEEE Network*, 7:8–18, 1993.

[31] Yang Zhao, Jie Liu, and Edward 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*, pages 259–268, Washington, DC, USA, 2007. IEEE Computer Society.