A Dynamic and Distributed TDM Slot-Scheduling Protocol for QoS-Oriented Networks-on-Chip

Nicola Concer†, Andrea Vesco‡ Riccardo Scopigno‡ and Luca P. Carloni†
†*Columbia University, Dept. of Computer Science, New York, NY 10027, USA*
‡*Istituto Superiore Mario Boella, 10138 Torino, Italy*
{*concer, luca*}*@cs.columbia.edu* {*vesco, scopigno*}*@ismb.it*

***Abstract*—We propose a run-time distributed and dynamic scheduling protocol for Quality-of-Service oriented Networks-on-Chip implementing Time Division Multiplexing of the channels. Our protocol automatically allocates the channel time-slots according to the specific routing algorithm implemented by the routers and the communication requirements of the applications currently executing. We present a theoretical analysis to model the protocol performance based on the traffic characteristics and to optimize the configuration of the NoC architecture which we propose to implement the protocol. This architecture relies on two independent and parallel networks: the first network transfers QoS-demanding data while the second is used to control the first and for best-effort traffic. System-level simulations demonstrate the effectiveness of this approach by showing that the runtime-generated reservation requests for QoS-demanding data are accepted by the NoC with very low blocking probability and that once a reservation has been established the requested latency and throughput guarantees are consistently met.**

I. Introduction

Networks-on-Chip (NoCs) are now considered the most promising solution to overcome the limitations of traditional bus-based architectures and satisfy the communication requirements of future Systems-on-Chip (SoCs) [1], [2]. In particular, future interconnects must provide a set of key features including: guaranteed latency and throughput, fairness on the access to the shared resources *e.g.*, memories and accelerators, and task isolation to partition the execution of the applications running in parallel on the SoC [3]. Many NoC architectures proposed in the literature rely on the Time-Division-Multiplexing (TDM) of the channels to satisfy these requirements [4], [5], [6], [7], [8]. In fact, multiplexing multiple messages on shared channels offers many advantages such as the elimination of the delays generated by the contentions in the NoC routers and the reduction of the overall communication delays. Additionally, when the applications to be supported are known at design time, the NoC architecture, and in particular the size of the buffers in the router and network interfaces, can be optimized while guaranteeing the requested performance [9].

On the other hand, future SoCs will integrate an increasingly large number of heterogeneous cores to match the high-performance and low-power consumption requirements demanded by portable devices such as smart-phones and tablet PCs [10], [11], [12]. The combination of programmable cores, reconfigurable cores, and specialized accelerators, which will be powered-up only when necessary [13], will make the NoC traffic vary greatly at run-time and very difficult to predict at design time. To address this challenge we propose *Pipeline Forwarding* (PF), a connection-based distributed scheduling protocol that builds on top of a synchronous TDM interconnect and can establish dynamic unidirectional connections between NoC nodes that remain active as long as the application task requesting them remains active. The proposed protocol is similar to the packet-switching protocols proposed for MAN and WAN networks [14], [15] which can be efficiently implemented in the SoC domain thanks to the common clock signal found in synchronous NoCs[1] and the low complexity of the router and network interface implementation logic.

This work is partially supported by the GSRC focus center, the Semiconductor Research Corporation program funded under the FCR Focus Center Research Program and the IOT - "*Piattaforma tecnologica innovativa per l'Internet of Things*" project, funded by Regione Piemonte.

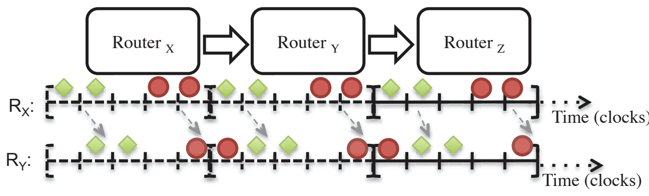


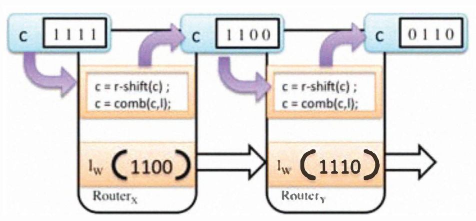
Fig. 1. Distributed-scheduling in a NoC with a PF-cycle of 5 frames and $\delta_D = 1$. Two independent messages: the green (diamonds) and red (circles) share the two links across the r_X, r_Y and r_Z routers.

As discussed in more detail in Section VI, most of the NoCs which use TDM rely either on the static assignment of the time-slots of the channels done at design time or on a centralized node that computes the semi-optimal time-slots allocation of the NoC resources at runtime [7], [17], [18], [19]. Instead, our protocol relies on a distributed and dynamic reservation algorithm that is implemented collectively by all routers and network interfaces and does not use any centralized node to govern the definition of the connections.

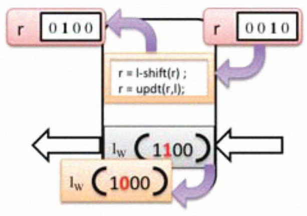
II. Pipeline Forwarding

Pipeline Forwarding is a connection-based distributed scheduling protocol that relies on a synchronous NoC with TDM channels to schedule the transmission of messages along the paths of the interconnect. The outcome of the scheduling procedure is that the data flits (flow-units) traversing the network are never delayed by a contention to access any

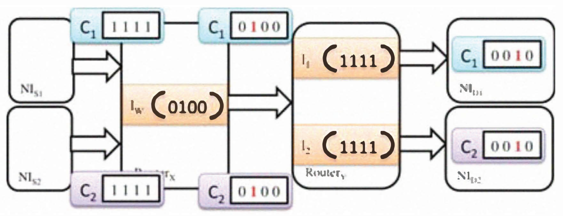
[1]Notice that state-of-the-art multi-core architectures, such as the Intel SCC [16], may have multiple clock domains but typically have a single-clock domain for the NoC connecting the cores of the system.



(a) Request update phase: the $\vec{c}$ is right-shifted δ times and combined with the local $\vec{l}$.



(b) Reservation phase: the $\vec{l}$ is updated with the new left-shifted $\vec{r}$.



(c) Contention on the west port of router r_X: two independent requests report that the same frame is available to different network interfaces.

Fig. 2. SVP-setup procedure.

given channel. The distributed scheduling protocol uses a time-reference structure common to all components of the interconnect. This structure is composed by *frames* (or time-slots) lasting one network-clock cycle and by *PF-cycles* that are a fixed sequence of C frames.

Fig. 1 shows the time expressed in network-clock cycles divided in PF-cycles of five frames and where the west links of routers r_X and r_Y are assigned to two different messages, respectively the green (diamond) and the red (circle) messages and forwarded according to the immediate forwarding rule presented below.

A *source network interface* NI_S requests a Guaranteed-Service (GS) connection towards a *destination network interface* NI_D by initiating a link-reservation procedure. This procedure is used to identify on each router on the path between NI_S to NI_D, a set of frames in the PF-cycle that can be reserved without contentions. If the set-up is successful the connection remains available until NI_S terminates it. At each step of the link-reservation procedure, frames are selected according to the *immediate forwarding* rules defined as:

1) all flits to be sent during the frame t by a router must be received at frame $t - \delta_D$, where δ_D is the forwarding latency of the data-routers;
2) a flit sent by router r_i during frame t must be forwarded by the downlink router r_{i+1} at frame $t + \delta_D$[2].

These rules force the routers to forward incoming flits immediately after the router-processing pipeline, thus minimizing the network-transfer delay and reducing the router buffering requirements[3]. We refer to the set of frames that respect the forwarding rules along the path between source and destination network interfaces as *schedules.*

The reservation procedure then is a solution for the *distributed scheduling problem* of finding a set of *available schedules* valid for reservation. The sub-set of available schedules selected to satisfy a specific connection request is called a *Synchronous Virtual Pipe* (SVP). After being established, an SVP offers a *guaranteed service* in the sense that: *i*) the delivery delay is deterministic and depends only on the distance between NI_S and NI_D, and the number of frames per PF-cycle assigned to the SVP; *ii*) the bandwidth supported by that SVP, *i.e.* the number of flits that are delivered to the destination during a PF-cycle, is deterministic and a function of the number of frames over the PF-cycle assigned to the SVP.

[2]Note that here we do not consider pipelined links. However it is possible to design PF-aware repeaters that are compliant with the forwarding rules and the reservation procedure presented in Section II-A.

[3]The original work on PF for wide-area networks [14] defines also more complex forwarding rules, but these solutions are not suitable for the embedded systems because they require large router-buffer capacities.

A. Distributed Resource Reservation

PF maintains the distributed scheduling through the global network-clock and through dedicated data structures (bit-vectors) called *availability vectors.* All routers of the control-network maintain a C bit-wide register called *link availability vector* $\vec{l}$ on each I/O port. Each element i of a $\vec{l}$ is associated to the i^{th} frame in the PF-cycle and its value indicates the reservation status of that frame on the link connected to that output port. The bits in a link vector $\vec{l}$ are defined as:

$$\vec{l}[i] = \begin{cases} 0 & \text{if the } i^{th} \text{ frame is reserved} \\ 1 & \text{if the } i^{th} \text{ frame is available} \end{cases}$$

Similarly way each NI has a *source availability vector* $\vec{s}$ defined as:

$$\vec{s}[i] = \begin{cases} 1 & \text{at the } i^{th} \text{ frame the source can send a data-flit} \\ 0 & \text{at the } i^{th} \text{ frame the source can not send a data-flit} \end{cases}$$

To initiate an SVP, a NI_S sends a reservation-request message to NI_D through the control network. The message contains the address of NI_S, the bandwidth requirement α indicating the number of frames per PF-cycle required for the SVP, and a *call availability vector* $\vec{c}$. The value of the parameter α is decided either in a static way at the NoC-design time or dynamically by NI_S according to the needs of the currently-executed application.

The $\vec{c}$ vector is used to collect a snapshot of the reservation status of the frames on the links traversed by the reservation-request message towards the destination. The $\vec{c}$ vector is initialized as equal to the $\vec{s}$ vector of NI_S. As shown in Fig. 2(a), $\vec{c}$ is updated at each router encountered on the path in the following way:

1) $\vec{c} = r\_shift(\vec{c}, \delta_D)$
2) $\vec{c} = combine(\vec{c}, \vec{l_p})$

where r_shift, is a function that shifts $\vec{c}$ by δ_D times to the right and $combine$ performs a bit-wise Boolean AND between $\vec{c}$ and the local $\vec{l_p}$, which is stored on the output port p. Once the reservation request reaches its destination, the bits of $\vec{c}$ that are set to 1 indicate the frames of the PF-cycle for which there exists an available schedule. When the NI_D processes the request, it selects α frames among the available ones and generates a reply message containing the address of the source of the request and the availability vector $\vec{r}$ whose bits are set as follows:

$$\vec{r}[i] = \begin{cases} 1 & \text{if the } i^{th} \text{ frame is selected} \\ 0 & \text{if the } i^{th} \text{ frame is not selected} \end{cases}$$

Then, NI_D sends the reply message back to NI_S along the *reverse path* followed by the reservation-request message. As shown in Fig. 2(b), upon the reception of the reply message each router updates its local availability vector $\vec{l_p}$ stored in the output port used by the relative request by setting:

1) $\vec{r} = l\_shift(\vec{r}, \delta_D)$
2) $\vec{l_p} = update(\vec{l_p}, \vec{r})$

where l_shift shifts $\vec{r}$ by δ_D times to the left and $update$ subtracts the vector $\vec{r}$ from $\vec{l_p}$.

Fig. 2(b) shows the update procedure executed by the control-routers when processing a reply message. NI_D has selected the third frame in vector $\vec{r}$. Therefore when the reply message reaches router r_1, the vector $\vec{r}$ is left-shifted and the local $\vec{l}$ is updated by setting $\vec{l_p}[1]$ to zero. The reservation procedure is completed when the reply message reaches NI_S and the local vector $\vec{s}$ is updated with the value of $\vec{r}$. At this point the SVP is established and NI_S can send the data-flits into the pipe according to the values of $\vec{s}$ and a frame-counter that indicates the current frame within the PF-cycle.

When the connection is no longer needed, the SVP is terminated by sending a *tear-down* message along the control network. The tear-down availability vector $\vec{t}$ of this message is initialized with the local $\vec{s}$ and processed by each router as:

1) $\vec{t} = r\_shift(\vec{t}, \delta_D)$
2) $\vec{t} = reset(\vec{t}, \vec{l_p})$

where *reset* adds the vector $\vec{t}$ to the local $\vec{l_p}$.

A link-reservation procedure may fail for two main reasons: *i*) a requested link does not have sufficient free frames to accommodate the requested α frames indicated in a $\vec{c}$, or *ii*) two or more $\vec{r}$ vectors have a contention for some frames of a specific channel. We address the first issue by rejecting the request through a *clear* message that is sent back to NI_S indicating the failure of the procedure. A clear message is a standard reply message where all the bits of the $\vec{r}$ vector is set to 1; the solution for the second issue instead is described in the following section.

B. SVP Setup through Atomic-Reservation procedure

When the reservation request message reaches a NI_D, the interface selects the frames to be assigned to the SVP. Since the snapshot procedure is not instantaneous, contentions on the frame selection may arise where one or more reservation requests indicate as available the same frames for the same router. Fig. 2(c) shows an example of a contention where two reservation requests traverse routers r_X and r_Y to reach their respective destinations, NI_{D_1} and NI_{D_2}. Because the two reservation requests cross the shared link on r_X's west port before a reply updates the $\vec{l}$ vector of this port, both network interfaces may select the same frame on r_X's west port to establish the SVP[4].To avoid conflicts, we use an *Atomic-Reservation* procedure: we conservatively limit the number of reservation requests that can traverse a link of the control network before a reply is received. This solution uses a control bit called *atomic flag* for each router output port and works as follows:

1) when a reservation request message passes through the output port of a router the relative atomic flag is set;
2) if another reservation request message reaches the same output port before the first reply messages comes back, the second reservation request is rejected and
3) the control router sends back to the NI_S of the rejected request a clear message to indicate the failure;
4) upon the reception of a clear message a source must re-attempt the SVP reservation request;
5) each router receiving the clear message resets the atomic flag set for the aborted request;
6) the atomic flag of the router triggering the clear message is then reset once the reply for the request message that set it finally arrives.

Clearly the blocking and re-attempt of the reservation requests lead to a probabilistic delay of the reservation procedure. In the following section we present an analytical model to study the blocking probability of a request and we justify the choice of the simple Atomic-Reservation procedure.

III. Theoretical QoS Analysis

An SVP is an *end-to-end independent time-invariant pipe with deterministic delay, assured bandwidth and loss probability equal to zero.* Therefore data traversing the NoC through a SVP experience: *i)* bounded end-to-end delay, and (*ii*) neither congestion nor data loss.

In the sequel we assume that a connection is established to send one single message. This simplification of the real case helps us in the definition of the statistical model and defines a worst-case scenario for the real application where multiple messages can use an already established connection.

Thanks to the deterministic operating principles of PF, throughput and delay can be predicted with an analytical model. If α is the number of frames requested for an SVP on a C-long PF-cycle, the guaranteed bandwidth BW is:

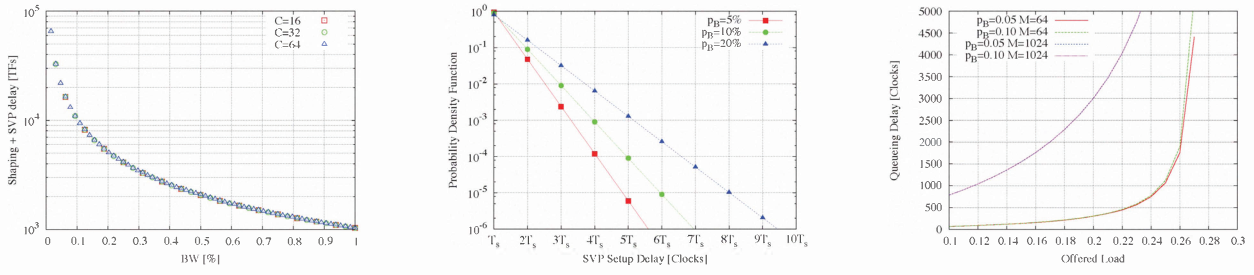
$$BW = \frac{\alpha}{C}, \qquad 1 \leq \alpha \leq C, \tag{1}$$

In this model we study two types of delay that a NI_S can experience: *i)* the deterministic data-transfer delay given by *shaping delay* T_{SH} and *SVP delay* T_{SVP} experienced by all flits when crossing the network within a SVP; *ii)* the statistical reservation delay T_R composed by the *request queueing delay* T_W and by the *SVP-setup delay* T_S[5].

The shaping delay T_{SH} measures the delay elapsed at NI_S to forward the entire data message of size M into the SVP in

[4]Note that in the example of Fig. 2(c) there is only one set bit so the conflict is certain.

[5]Note that the tear-down delay is negligible for the purpose of this analysis because the tear-down message never delays the generation of a new reservation request.



(a) Upper-bound of the Shaping and SVP delay components as function of the bandwidth reservation for different PF-cycle lengths; the SVP crosses 8 hops and the data consists of 1024flits.

(b) Probability density function of the SVP setup delay component as function of the reservation request blocking probability.

(c) Mean queueing time of reservation requests at the source NI for different reservation request blocking probability and data width.

Fig. 3. PF theoretical analysis.

accordance to the schedule resulted from the SVP setup phase. This delay can be deterministically upper-bounded given the data length M and the bandwidth reservation α/C as follows:

$$T_{SH} = \begin{cases} M \cdot C - 1, & \text{if } \alpha = 1; \\ (C-\alpha)\left(1 + \lfloor \frac{M}{\alpha} \rfloor\right) + (M-1), & \text{if } 1 < \alpha \le C; \end{cases} \quad (2)$$

The SVP delay T_{SVP} can be deterministically calculated given the length of the path that data-flits take through the network. According to the *immediate forwarding* rule, all flits to be sent at frame t by router r must be stored in its input buffer during frame $t - \delta_D$. Hence, when a NI_S selects a destination that is distant H hops from the source, the SVP delay experienced by all flits once they are injected into the pipe is:

$$T_{SVP} = \delta_D \cdot H \quad (3)$$

where δ_D is the number of pipeline stages at the data routers. Fig. 3(a) shows the deterministic upper-bound of the sum of the buffering and SVP delays as function of the bandwidth reservation α/C. In principle, the greater the bandwidth reservation, the lower the delay independently of the PF-cycle length C. When the whole PF-cycle is allocated to a single SVP, a flit is forwarded into the pipe at every clock cycle. Therefore T_{SVP} depends only on the amount of data M to be transferred and the delays coincide for all the C values of Fig. 3(a). For example, a message M of 1024 flits crossing 8 hops experiences a delay $T_S + T_{SVP} = 1031$ clock cycles, when the whole PF-cycle is allocated to a single SVP. The reservation delay $T_R = T_W + T_S$ is intrinsically statistical because it depends on the blocking probability p_B that the SVP reservation requests experience on the control network. Assuming that each network interface on the control network can handle one SVP reservation requests at a time, the network interface can be modeled as a queueing system where requests are buffered waiting to be served. Moreover, assuming that the IP cores perform reservation requests according to a Poisson process, this system can be described with the $M/G/1$ queueing model following the well-known Kendall's notation. As a result, the mean request queueing delay at NI_S $E[T_W]$ is given by the Pollaczeck-Khinchin formula [20]:

$$E[T_W] = \frac{\lambda \overline{S^2}}{2(1-\rho)} \quad (4)$$

where ρ indicates the system utilization, $\lambda = \frac{OL}{M}$ is the mean arrival time of reservation requests at the network interface, OL is the offered load measured as *flit/cycle/node*, and μ is the service rate. The mean service time S:

$$S = \frac{1}{\mu} = E[T_S] + T_{SH} + T_{SVP} \quad (5)$$

and consequently the system utilization $\rho = \frac{\lambda}{\mu}$.

$$\overline{S^2} = E\left[S^2\right] = \sigma_S^2 + E[S]^2 \quad (6)$$

Assuming that the probability for the IP cores to stall is negligible and the NI_D processing delay equal to the control router one, the minimum SVP-setup delay T_s is:

$$T_s = \delta_C \cdot 2H \quad (7)$$

where δ_C is the processing time of control routers. Therefore, for a given p_B, the probability-density function describing the relative likelihood for the SVP setup delay to occur at xT_s, $\forall 1 \le x \le \infty$ is[6]:

$$P[T_S = xT_s] = p_A \cdot p_B^{(x-1)} \quad (8)$$

where the probability for a reservation request to be accepted p_A is given by:

$$p_A = \frac{1}{\sum_{i=0}^{\infty} p_B^i} \quad (9)$$

Accordingly, the mean SVP setup delay is given by:

$$E\left[T_S\right] = p_A \sum_{x=1}^{\infty} x \; T_s \; p_B^{(x-1)} \quad (10)$$

Notice that the blocking probability p_B is hard to compute because it depends on static properties of the NoC such as topology and routing as well as dynamic parameters such as traffic pattern and offered load. For this reason in this analysis we keep p_B as a parameter to which we assign values derived from the simulation model described in Section V.

Fig. 3(b) shows the probability-density function of the SVP setup delay for different values of p_B. The probability for T_S to increase because of the failure of multiple setup attempts is given by $P(T_S = xT_s)$ when $x > 1$. According to Eq. 8,

[6]According to the worst-case analysis, we assume that the penalty for a blocked request is always T_s.

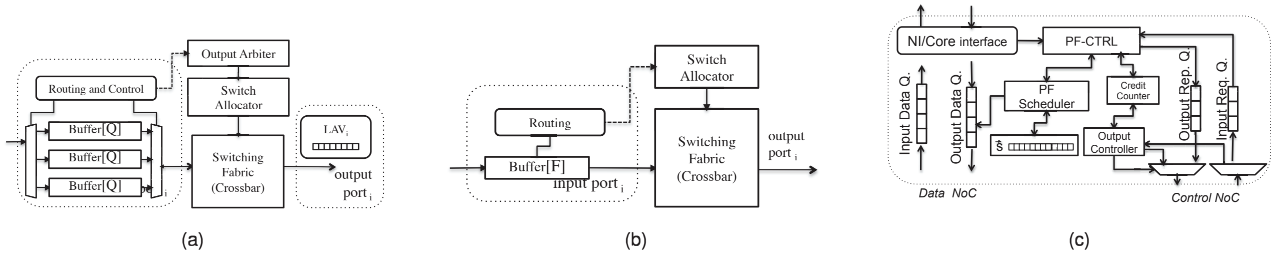


Fig. 4. Block diagram of: (a) control router, (b) data router, and (c) PF-aware Network Interface.

this probability grows as the blocking probability p_B also grows. Therefore, the probability of having a minimal setup delay is $P(T_S = T_s)$ which decreases for larger values of p_B. However, according to Eq. 8, the probability for a request to be repeated multiple times decreases in a very sharp way for each value of p_B.

This important result justifies the implementation of the Atomic Reservation procedure that uses low-complexity routers at the price of higher blocking probability p_B. Notice, however, that even with a high blocking probability p_B the chances of a request of having multiple successive blockings are very low, as shown in Fig. 3(b),

Fig. 3(c) shows the mean value of the queuing delay $E[T_W]$ experienced by reservation requests at the source NI as function of the Offered Load (OL) when the queuing system is stable ($\rho < 1$). The mean value of the queuing delay shows a low correlation with p_B whereas it is strongly dependent on the data message length. This can be explained by looking at the service time in Eq. 5. The SVP-setup delay T_S depends on p_B but since the probability for multiple consecutive blockings of reservation request is very low, $E[T_S]$ has a very low correlation with p_B. On the contrary, the shaping delay T_{SH} sharply increases with M. Therefore, since the queuing delay T_W is directly proportional to the service time, $E[T_W]$ increases more with M than with p_B.

Finally, the mean throughput G achieved by each IP core in the NoC can be estimated as:

$$G = \frac{M}{E[T_S] + T_B + T_{SVP}}. \tag{11}$$

Next, we describe the NoC architecture that we developed to test the performance of the PF protocol.

IV. The PF-NoC Architecture

By leveraging the large availability of wires in the NoC environment [21], [22], we propose a NoC architecture to implement the PF scheduling protocol that consists of two independent and parallel networks. We use a control network for handling the link-reservation procedure described in Sec. II-A and to deliver best-effort (BE) traffic and a data network implementing the pipeline forwarding plane. The use of two separated networks allows us to simplify the design of the routers so that they can be optimized and configured for the kind of network that they belong to (BE or GS) and avoid necessary tradeoffs. For the current implementation of both networks we opted for a 2D-Mesh topology but the architecture is not bounded to any specific topology (it just requires the support of reversible paths).

Control Router: We implemented a standard input-queued router (Fig. 4 (a)) with three *virtual channels* (VCs) for each input port, and router-lever credit-based flow control [23]. To avoid protocol deadlock we used the *strict ordering* of the messages allocating one VC for each message type [24]. In addition to these standard features, the control routers implement the logic of the bit-wise operations described in Section II-A. Finally for the reservation request and the reply messages, which must follow the same path in opposite directions, the router implements the *XY* and *YX* routing algorithms, respectively[7].

Data Router: Thanks to the reservation procedure handled by the control network, data routers have the very simple architecture shown in Fig. 4 (b): they do not implement a router-level flow control and data flits can be forwarded right after being received. Hence their input queue size can be set to one slot. To avoid complex and expensive routing tables data-routers implement the *XY* routing algorithm that they compute for every data-flit that they handle. This feature requires the specification of the flit destination in all data-flits. Note that the control and data networks do not need to be synchronous. The only connection points between them in fact are the network interfaces.

Network Interface: As shown in Fig. 4(c), NIs are composed by four main queues, a control and scheduling logic, a frame counter to keep track of the current frame in the PF-cycle and the register $\vec{s}$. The control part uses two input and output queues to store the incoming and outgoing SVP reservation requests. The link to the control router is multiplexed with the output controller that handles the injection of end-to-end credits for the flits from the data network. The control block handles the generation of requests and updates the scheduler. The scheduler uses the frame-counter to keep track of the current frame and reads the $\vec{s}$ vector to regulate the access to the link towards the data network. In this work we limit NIs to support one incoming and one outgoing SVP reservation request at the time. This restriction can be removed by increasing the number of input and output queues and adding an arbitration logic to handle them.

The PF-NoC avoids by construction deadlock and message-dependent deadlock on the data network thanks to the link-

[7]Note that PF is orthogonal to the routing algorithm as long as this allows requests and replies to follow the same path on opposite directions.

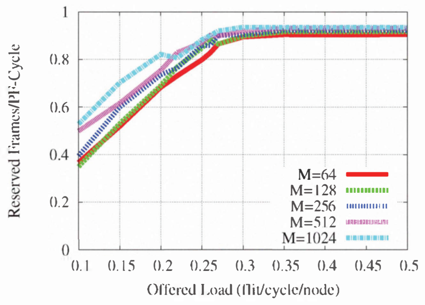


Fig. 5. Maximum number of assigned frames over PF-cycle depends on the offered load with Transpose task graph.

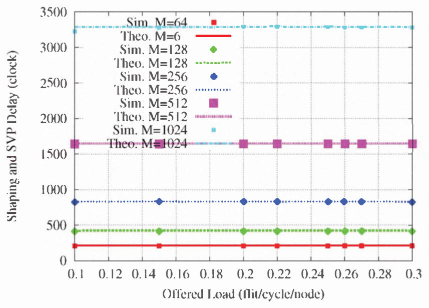


Fig. 6. Simulated and theoretical points of Shaping and SVP delay as function of the offered load and the size of the message to be transferred.

reservation procedure. To avoid data overflow at N_D we use a credit-based end-to-end flow control [23] and we use the BE network to deliver the end-to-end credits. The major drawback of this approach is that credits travel along the control network with no guarantees for the delivery delay. This can potentially cause the underutilization of the bandwidth offered by the SVP, thus breaking the "quality of service contract" defined during the SVP setup. On the other hand, this solution is a good compromise between efficiency and implementation costs.

V. Experimental Results

We developed an event-driven simulator with detailed models of the NoC components, such as routers and network interfaces and high-level models for the processing units that are the sources of the network traffic. We analyzed the SVP architecture with the well-known synthetic traffic patterns Tornado and Transpose [23] as well as two realistic task graphs, MWD and VOPD [25] on a 4×4 Mesh NoC.

In the following experiments the number α of frames requested by each SVP is statically selected according to the specific traffic pattern[8]. In particular, we chose α such that all the SVPs traversing any given channel are supported and the number of occupied frames is maximized. Formally: given a task graph $G(V,E)$ with V the set of tasks and E the set of directed edges (representing the communication between the tasks) and given the topology graph $N(U,L)$ with U the set of Intellectual Properties (IP) and L the set of links connecting the IPs, the bijective mapping function η is defined as [26]:

$$\eta : V \rightarrow U,\ s.t.\ \eta(v_i) = u_i, \forall v_i \in V,\ \exists u_i \in U \qquad (12)$$

Given a channel $c_i \in L$ and the function $\phi_{\eta(G,N)}(c_i)$ which returns the number of SVPs passing through c_i, the static number α of requested frames for each SVP on a PF-cycle of size C is:

$$\alpha = min(\lfloor C/\phi(c_i)\rfloor),\ \forall c_i \in E \qquad (13)$$

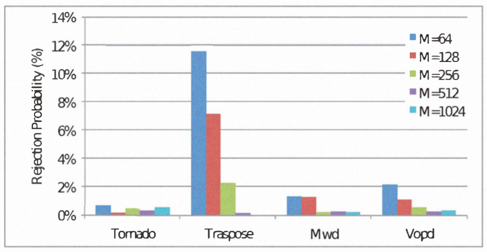
Fig. 5 shows the maximum number of reserved frames on a PF-cycle as function of the offered load (OL) which varies between 0.1 and 1.0 flit per clock-cycle. The graph considers a Transpose traffic pattern where an SVP is set-up to transmit one single message whose size varies between 64 and 1024 flits. The figure shows that the average number of reserved frames on a PF-cycle increases linearly with the offered load until it reaches a maximum point where no more frames can be allocated. Also, as expected, those SVP which are opened for larger messages use more efficiently the network as relatively less time is spent in setting them up.

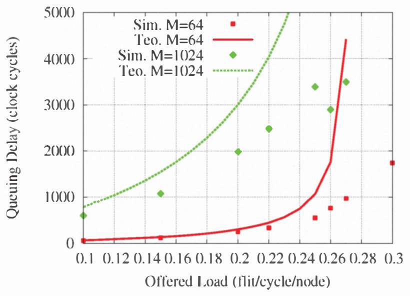
In our analysis, the network is stressed up to the 90% of the saturation point. Additionally to ease the comparison between the theoretical model and the simulation results we consider only data messages traveling along the longest path for each given traffic pattern. As reported in Sec. III the delay associated to an SVP consists of two components: one deterministic and one statistical. Fig. 6 compares the theoretical and simulation-based deterministic delays as function of the offered load and the size of the messages to transfer along an already-established SVP. In the figure, all points from the simulation results match the curves of the theoretical model and are orthogonal to the load stressing the network. This indicates that the simulation results are correct and that the "QoS contract" is always respected.

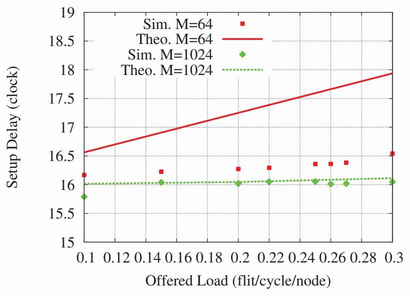
To quantify the statistical delay T_R, we studied the probability for a SVP request to be blocked for the reasons presented in Sec. II-B. Fig. 7(a) shows the total blocking probability for different traffic patterns and different message sizes. The blocking probability is below 2% for most cases, especially the ones with large messages. In fact, the length of the message to be transferred is inversely proportional to the number of generated SVP-setup requests. The worst case is given by the 64-flit long messages for the Transpose traffic pattern. This poor performance is not surprising as this traffic pattern forces very distant nodes (such us those at the opposite corners of the Mesh) to communicate. Clearly this is a worst-case scenario. In a well-designed SoC the notion of data locality is an important design factor that can greatly affect the network communication delay and the power consumption of the system.

Fig. 7(b) shows the measured average queuing delay T_W for SVP requests compared to the analytical curves of Eq. 4 plotted using the blocking probability p_B from the simulation traces. The delay increases with the offered load due to the queuing effect of the requests to be satisfied at the network interface. In accordance with the queuing theory, when the OL becomes greater than $\frac{M}{S}$, where S is the mean service time expressed in Eq. 5, the system saturates. For example, in the

[8]An optimal value of α can be extracted by annotating the communication requirements of the application at compile time [19]. Here we keep α constant to simplify the analysis.







(a) Blocking probabilities under Transpose different traffic patterns and session sizes.

(b) Average queueing delay T_W for a SVP request with a Transpose traffic pattern.

(c) Average SVP-setup delay after the queing delay T_W in a Transpose traffic pattern.

Fig. 7. Simulation analysis.

Transpose traffic with messages of 1024 flits this value is close to $0.3 flit/cycle/node$[9].

Fig. 7(c) shows the actual delay T_S taken by the system to setup an SVP (after the queing delay) compared to the curves of Eq. 10. Clearly the SVP-setup delay has a minor effect on the overall communication phase. For example the impact of a high blocking probability such as the one registered for Transpose and $M = 64$ contributes to the whole reservation delay only by a 3%.

Fig. 8 shows the cumulative success probability as function of the number of attempts. For most traffic patterns, the probability of success at the first attempt is beyond 98%. In the worst case, for the Transpose traffic, 100% is reached within the first five attempts.

VI. Related Work

The literature offers a rich class of NoC architectures which leverage TDM mechanisms to support communication with guaranteed QoS. The Æthereal NoC was among the first in this class of architectures: it is based on a router design which supports both GS and BF traffic and it uses a highly optimized mechanism for the static allocation of the frames which is possible thanks to the design-time knowledge of the communication requirements of the target applications [4], [27]. Ælite is an evolution of the Æthereal NoC which also relies on the static reservation of the channels based on the design-time knowledge of the communication requirements [8].

Moreira, Marescaux *et.al.* propose two TDM NoCs with dynamic and centralized reservation of the channels. Given the network status and the application requirements, a central management unit computes a suboptimal channel resource reservation to support the new application [19], [28]. Our approach differs from these TDM NoCs because we rely on a distributed and dynamic resource scheduler instead of a centralized manager.

Gebremichael, Goossens *et al.* discuss a distributed and dynamic time-slot reservation procedure for the Æthereal NoC where the source decides which slots should be reserved to the connection to be created and the procedure fails in case one of the requested slots are not available [29], [30]. In PF instead, it is the destination that given the set of available SVPs indicated in request ($\vec{c}$) selects which slots to use. The Æthereal approach in fact is optimal once the characteristic of the application to support are known at design time, so that source network interfaces can be statically assigned with a fix number of frames (slots) to request in case a connection is needed. Our protocol instead is more flexible because the source only needs to know the number of frames to reserve in order to meet the application requirements. Additionally, our NoC architecture differs from Æthereal because it relies on two partitioned and parallel networks. Our approach reflects our previous works where we show that simpler router architectures can be beneficial for the overall interconnect performance [31].

QoS on NoCs can be achieved also without the use of TDM. Srinivasan *et al.* propose an automated heuristic to generate application-specific NoCs that is optimized to meet the latency and throughput constraints known at design time [17]. Whereas Vellanki *et al.* guarantee QoS through a set of virtual circuits which are selected according to a priority policy optimized to deliver the requested bandwidth and latency [32]. The Mango approach also uses virtual channels to guarantee QoS as part of an asynchronous NoC architecture. [5]. Guz *et al.* propose a greedy algorithm that given a basic NoC configuration and a set if application requirements (in terms of latency and throughput) cyclically increases the storage capacity of the routers traversed by the data-flows until the QoS constraints are met [33]. A different approach is presented by Anders *et al.* who propose an NoC implementing two networks: a packet-switched interconnect to handle best-effort traffic and control messages and a parallel circuit-switched network used to transport guaranteed data [34]. The main differences with our work are the NoC architecture and the handling of the contentions during the link-reservation procedure.

VII. Conclusions and Future Work

We proposed a distributed and dynamic scheduling protocol for TDM NoCs that allows us to setup and tear down Guaranteed-Service connections between any source and any destination network interfaces with limited failure probability. We also presented an analytical model to predict the performance of the protocol and to tune the parameters of the

[9]The curves differ because the analytical model captures a *worst-case* while the simulation model reports the average values extracted from the traces of the simulator. However, the simulated points are always lower than the analytical curves as expected for the worst-case scenario.

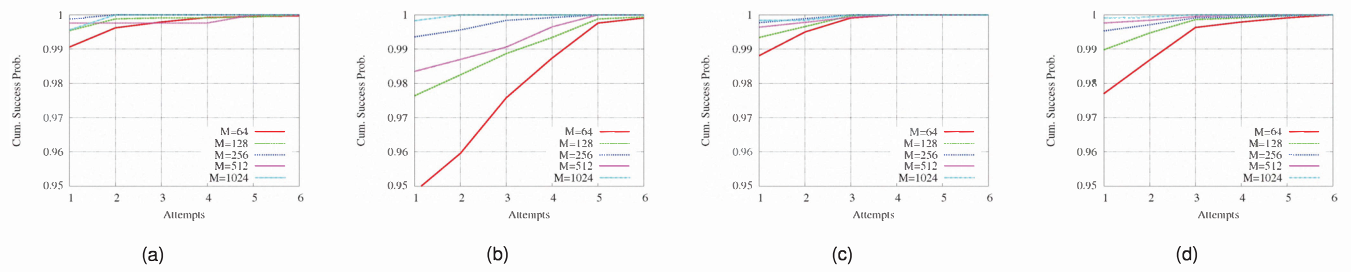


Fig. 8. Cumulative Success Probability for (a) MWD, (b) Transpose, (c) Tornado and (d) VOPD traffic patterns as function of the SVP setup attempts.

interconnect. The proposed protocol leads to a fairly simple NoC architecture (PF-NoC) with minimal storage requirement. The experimental results that we obtained encourage us to continue the development of PF-NoC. Future work includes: *i*) optimize the dynamic allocation of the frames to the different SVPs; *ii*) explore the definition of classes of services to expose the choice of the number of frames to reserve for a given SVP to the user. *iii*) implement the PF-NoC in RTL to obtain accurate power and area estimations; *iv*) improve the reservation procedure to reduce the blocking probability for long-distance communications; *v*) add support for best-effort traffic in the control network; *vi*) explore adaptive routing protocols to diversify the paths used by the GS connections and improve the resource utilization of the NoC.

References

[1] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. of the Design Automation Conference*, Jun. 2001, pp. 684–689.

[2] L. Benini and G. D. Micheli, "Networks on chip: A new SoC paradigm," *IEEE Computer*, vol. 49, no. 2/3, pp. 70–78, Jan. 2002.

[3] A. Hansson, M. Coenen, and K. Goossens, "Undisrupted quality-of-service during reconfiguration of multiple applications in networks on chip," in *Proc. of the Conf. on Design, automation and test in Europe*. IEEE Press, Apr. 2007, pp. 954–959.

[4] J. Dielissen *et al.*, "Concepts and implementation of the Philips network-on-chip," in *IP-Based SoC Design*, Nov. 2003.

[5] T. Bjerregaard and J. Sparso, "A router architecture for connection-oriented service guarantees in the mango clockless network-on-chip," in *Proc of the Conf. on Design, Automation and Test in Europe*, Apr. 2005, pp. 1226–1231.

[6] Z. Lu and A. Jantsch, "TDM Virtual-Circuit Configuration for Network-on-Chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 8, pp. 1021–1034, Jul. 2008.

[7] M. Schoeberl, "A time-triggered network-on-chip," in *Field Programmable Logic and Applications*, Aug. 2007, pp. 377–382.

[8] A. Hansson and M. Subburaman, "Ælite: A flit-synchronous network on chip with composable and predictable services," in *Conf. on Design, Automation and Test in Europe*, Jun. 2009, pp. 250–255.

[9] A. Hansson *et al.*, "Enabling application-level performance guarantees in network-based systems on chip by applying dataflow analysis," *IET Computers & Digital Techniques*, 2009.

[10] C. H. van Berkel, "Multi-core for mobile phones," in *Conf. on Design, Automation and Test in Europe*, Apr. 2009, pp. 20–24.

[11] M. Lyons, M. Hempstead, G.-Y. Wei, and D. Brooks, "The accelerator store framework for high-performance, low-power accelerator-based systems," *IEEE Comput. Archit. Lett.*, vol. 9, pp. 53–56, July 2010.

[12] P. Kollig, C. Osborne, and T. Henriksson, "Heterogeneous multi-core platform for consumer multimedia applications," in *Conf. on Design, Automation and Test in Europe*, Apr. 2009, pp. 1254–1259.

[13] S. Borkar and A. A. Chien, "The future of microprocessors," *Commun. ACM*, vol. 54, pp. 67–77, May 2011.

[14] C. Li, Y. Ofek, A. Segall, and k. Sohraby, "Pseudo-isochronous cell switching in atm networks," *Computer Networks and ISDN systems*, vol. 30, no. 24, pp. 2359–2372, Dec. 1998.

[15] M. Baldi and Y. Ofek, "Fractional lambda switching - principles of operation and performance issues," *Simulation: Trans. of The Society for Modeling and Simulation Int.*, vol. 80, no. 7, Jul. 2004.

[16] T. G. Mattson *et al.*, "The 48-core SCC processor: the programmer's view," in *Proc. of the Int. Conf. for High Performance Computing, Networking, Storage and Analysis (SC10)*, Nov. 2010, pp. 1–11.

[17] K. Srinivasan *et al.*, "Application specific network-on-chip design with guaranteed quality approximation algorithms," in *Proc. of Asia and South Pacific Design Automation Conference*, 2007, pp. 184–190.

[18] A. Hansson, K. Goossens, and A. Rădulescu, "A unified approach to mapping and routing on a Network-on-Chip for both best-effort and guaranteed service traffic," *VLSI Design*, pp. 1–17, 2007.

[19] O. Moreira, J. J.-D. Mol, and M. Bekooij, "Online resource management in a multiprocessor with a network-on-chip," in *Proc. of the Symp. on Applied computing*, Mar. 2007, pp. 1557–1564.

[20] L. Kleinrock, *Queueing Systems. Volume 1: Theory*. Wiley Interscience, 1975.

[21] J. Balfour and W. J. Dally, "Design tradeoffs for tiled CMP on-chip networks," in *Conf. on Supercomputing*, Nov. 2006, pp. 187–198.

[22] E. Carara *et al.*, "Router architecture for high-performance NoCs," in *Proc. of the Conf. on Integrated circuits and systems design*, Jan. 2007, pp. 111–116.

[23] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Mateo, CA: Morgan Kaufmann Publishers, 2004.

[24] A. Hansson, K. Goossens, and A. Rădulescu, "Avoiding message-dependent deadlock in network-based systems on chip," *VLSI Design*, vol. 2007, pp. 1–10, May 2007.

[25] D. Bertozzi *et al.*, "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Trans. on Parallel and Distributed Systems*, vol. 16, no. 2, pp. 113–129, Feb. 2005.

[26] S. Murali and G. De Micheli, "Bandwidth-constrained mapping of cores onto noc architectures," in *Conf. on Design, Automation and Test in Europe*, 2004, pp. 208–2013.

[27] E. Rijpkema *et al.*, "Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip," *IEE Proc.-Computers and Digital Tech.*, vol. 150, no. 5, pp. 294–302, Sep. 2003.

[28] T. Marescaux *et al.*, "Dynamic TDM virtual circuit implementation for NoC," in *Workshop on Embedded Systems for Real-Time Multimedia, 2005.*, Oct. 2005, pp. 47–52.

[29] B. Gebremichael *et al.*, "Deadlock prevention in the Æthereal protocol," in *Proc. Conf. on Correct Hardware Design and Verification Methods*, Oct. 2005, pp. 345–348.

[30] K. Goossens, J. Dielissen, and A. Rădulescu, "Æthereal network on chip:concepts, architectures, and implementations," *IEEE Design and Test of Computers*, vol. 22, no. 5, pp. 414–421, May 2005.

[31] Y. J. Yoon, N. Concer, M. Petracca, and L. Carloni, "Virtual channels vs. multiple physical networks: a comparative analysis," in *Proc. of the Design Automation Conference*, Jun. 2010, pp. 162–165.

[32] P. Vellanki, N. Banerjee, and K. S. Chatha, "Quality-of-service and error control techniques for mesh-based network-on-chip architectures," *Integr. VLSI J.*, vol. 38, pp. 353–382, Jan. 2005.

[33] Z. Guz *et al.*, "Network delays and link capacities in application-specific wormhole nocs," *VLSI Design*, vol. 2007, p. 15, Feb. 2007.

[34] M. Anders *et al.*, "A 2.9 tb/s 8w 64-core circuit-switched network-on-chip in 45nm CMOS," in *Proc. of the European Solid-State Circuits Conference*, Sep. 2008, pp. 182–185.