Dynamic Guaranteed Service Communication on Best-Effort Networks-on-Chip

Peter Munk Matthias Freier
Robert Bosch GmbH
70442 Stuttgart, Germany
{peter.munk, matthias.freier}@bosch.com

Jan Richling
South Westphalia University of Applied Sciences
58095 Hagen, Germany
richling.jan@fh-swf.de

Jian-Jia Chen
TU Dortmund University
44227 Dortmund, Germany
jian-jia.chen@cs.uni-dortmund.de

***Abstract*—In order to execute applications under real-time constraints on many-core processors with a Network-on-Chip (NoC), guaranteed service (GS) communication with guaranteed end-to-end latency and bandwidth is required. Several hardware-based solutions for GS communication have been proposed in literature. However, commercially available many-core processors, e. g., Tilera's TilePro64 or Adapteva's Epiphany, do not support such features. In this paper, we propose a software solution that allows GS communication on 2D-mesh packet-switching NoCs. Our investigation is based on a hardware model that is applicable to commercially available processors, which include multiple NoCs to separate request and response packets and support only best-effort communication. We prove that a common upper bound of the injection rate for all sources limits the congestion which leads to an upper bound of the worst-case transmission latency (WCTL) for any transmission, i. e., the combination of a request and a response packet. Furthermore, our approach supports arbitrary transmission streams that can be modified at runtime without violating the upper bound of the WCTL, as long as the injection rate is not violated. This enables adaptive features such as task migration or dynamic scheduling policies. Experiments evaluate our solution for different traffic patterns.**

I. Introduction

The performance of single core processors is mainly limited by the power dissipation that accompanies high clock frequencies. While Moore's law is still valid, the rising number of transistors leads to a better performance if number of cores per chip is increased instead of building larger monolithic cores [1]. Hence, processors with more than thousand cores are expected within the next decade [1]. These many-core processors require a scalable and flexible interconnection. On-chip networks or Networks-on-Chip (NoCs) are an adaptation of well-known networking concepts for many-core processors that scale well with chip size and complexity [2].

Due to the increasing computational demand of topics like autonomous driving, many-core processors will emerge in safety-relevant embedded systems with real-time requirements. Since compliance with real-time constraints is a system property, NoCs have to support guaranteed service (GS) communication, i. e., guaranteed end-to-end latencies for all transmissions. Each transmission consists of one request packet and one corresponding response packet.

Adaptive or dynamic features such as online reconfiguration of the application, task migration or dynamic scheduling policies allow to manage the temperature distribution on the chip, to optimize the power efficiency of the processor, and to decrease the failure probability. Using these dynamic features under real-time constraints requires dynamic adjustment of GS communication, i. e., changing the stream of transmissions at runtime while preserving the end-to-end latency guarantees.

Basically, there exist two common approaches to achieve GS communication on NoCs: i) hardware implementation and ii) comprehensive analysis. In hardware, virtual circuits in packet-switching NoCs can provide bounded end-to-end latencies [2]. Logically independent resources, typically realized as virtual channels (VCs), separate the virtual circuits to avoid contention [2]. Dynamic modification of GS communication requires a new analysis of the virtual circuits and an online reconfiguration of the VCs. Comprehensive analysis methods such as network calculus, on the other hand, theoretically consider all delay effects in order to calculate the end-to-end latencies. However, any result is only valid for the fixed stream of transmissions it was derived for. Whenever the stream of transmissions is modified the analysis has to be repeated.

In this paper, we present a software solution that enables dynamic GS communication on NoCs. Our hardware model focuses on NoCs of many-core processors that are commercially available, e. g., in Tilera's TilePro64 [3] and Adapteva's Epiphany [4]. Both processors have in common that they include multiple 2D packet-switching NoCs to separate request and response packets. The NoCs use the dimension order (XY) routing policy with wormhole switching but do not guarantee end-to-end latencies. Thus, we consider a similar hardware model that includes separate request and response NoCs with the dimension order (XY) routing policy and wormhole switching. We assume that the links and routers of the NoC are deterministic and process data in a fixed number of cycles.

Our key idea is to enforce a common upper bound of the injection rate for each source. The injection rate of a source is the inverse of the time between sending two consecutive transmissions. An upper bound of the injection rate can be implemented in software without additional hardware support. We prove that the congestion in both NoCs is limited and derive an upper bound of the worst-case transmission latency (WCTL) of any transmission. Hence, the stream

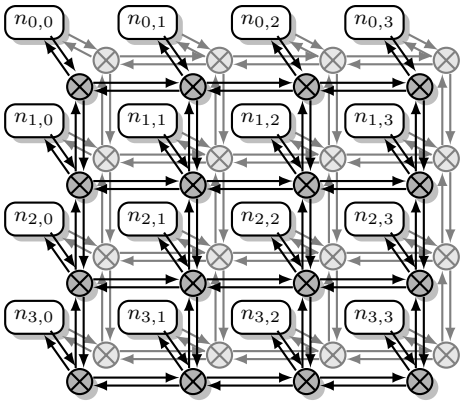


Figure 1. The platform consisting of 4x4 nodes connected with bidirectional links to 4x4 routers of the request NoC (dark) and 4x4 routers of the response NoC (light).

of transmissions can be modified online and dynamic GS communication is possible as long as all sources obey the common injection rate. Note that our approach is inverse to the comprehensive analysis methods mentioned above. Instead of taking a fixed transmission stream as given, we shape the stream in such way that the latency is inherently bounded. In contrast to methods based on time-division multiplexing (TDM), our solution is more flexible since it does not restrict injections to specific time slots.

The main contribution of this work is a software solution that allows GS communication on NoCs that supports only best-effort communication. Our solution enables dynamic modification of the GS communication as long as the maximum injection rate is not exceeded, which has to be supported by the application. Experiments evaluate our approach under different traffic patterns.

The remainder of this paper is structured as follows: Section II introduces our system model. In Section III, we explain our limited injection rate approach. Experimental results are presented in Section IV. Section V provides an overview of the related work. Our conclusions are drawn in Section VI.

II. System Model

In the following, we provide a model of the hardware and the communication. The hardware model is later used to derive the latency of a transmission. The communication model captures all transmitted data and hence allows to determine collisions of packets.

Typical real-time applications consist of communicating, periodically executed tasks. We explicitly exclude applications from our model. Instead, we capture any communication on the chip in a *traffic pattern*, i. e., the stream of transmissions $\mathcal{T}$ that all sources inject into the request NoC. Any transmission $\tau_i \in \mathcal{T}$, e. g., read or write command, consists of a request packet $p_i^{req} \in \mathcal{P}$, which is injected into the request NoC at time t_i^{req}, and the corresponding response packet $p_i^{rsp} \in \mathcal{P}$, which is injected into the response NoC by the destination after processing the request packet. Note that higher-level abstractions such as periodically broadcast messages with arbitrary sizes can be represented by an infinite set of transmissions in the traffic pattern.

We consider a hardware platform that consists of $x \times y$ nodes $n_{i,j} \in \mathcal{N}$, as depicted in Figure 1. The bijective functions $src : \mathcal{P} \to \mathcal{N}$ and $dst : \mathcal{P} \to \mathcal{N}$ denote the corresponding source and destination nodes of each packet. We consider only unicast packets and transmissions with exactly one source and one destination.

The $x \times y$ nodes are connected by two separate but identical 2D-mesh packet-switching NoCs. Two separate NoCs are required to break the request-response dependency [5]. Most protocols for communication between nodes are based on a request / response mechanism, e. g., the Virtual Component Interface (VCI) protocol, which allows to connect different intellectual property (IP) components. If both request and response packets were transmitted on the same NoC, deadlocks could still arise due to dependencies at transmission level [5]. E. g., the request packet of one transmission could hold resources required by the response packet of another transmission and vice versa.

A network interface (NI) connects each node $n_{i,j} \in \mathcal{N}$ to both NoCs. The NI either stalls the CPU until a response packet is received, or it continues the CPU's execution and deals with the response packet when it is received. The former case is known as *synchronous* or blocking communication. The latter case is known as *asynchronous* or non-blocking communication.

Each NoC consists of $x \times y$ routers $r \in \mathcal{R}$. A schematic overview of a router r is presented in Figure 2. The routing logic implements the dimension-order (XY) routing policy. Under this policy, a packet is first transmitted along the X axis. When the column of the destination node is reached, the packet is transmitted along the Y axis until the destination node is reached. This policy is known to be deadlock-free and deterministic [2]. Under a deterministic routing policy, the same path between any source and destination pair is always chosen [6]. Hence, deterministic routing policies ensure the in-order delivery of packets [2]. The function $h : \mathcal{P} \to \mathbb{N}^+$ denotes the number of routers on the route of a packet from its source to its destination.

Two neighbor routers of one NoC and an NI and a router are connected by a link $\lambda \in \mathcal{L}$. A link is a bidirectional connection, so two neighbor routers can exchange packets in both directions without a conflict. A flow control unit (flit) is an atomic entity that is transmitted over any link $\lambda \in \mathcal{L}$ in one clock cycle. Thus, all links have the same bandwidth $b = 1\frac{\text{flit}}{\text{cycle}}$. Each packet has a common fixed size that is denoted by s and specified as a natural number of flits. Note that a link is physically implemented by a data signal with the size of a flit and another signal that is required by the router-level flow control mechanism.

Consider the packet $p_i \in \mathcal{P}$ in an otherwise empty request or response NoC, i. e., no other packets can interfere with packet p_i. The router determines the requested output port

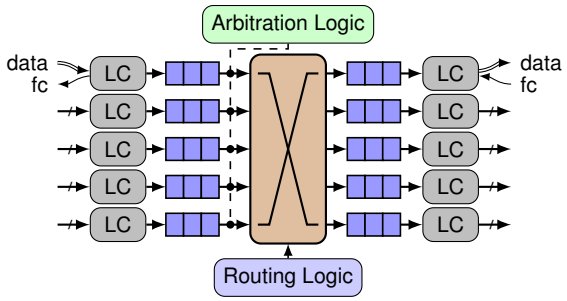


Figure 2. Schematic overview of a router with five input and output link controllers (LCs) connecting the data and flow control (fc) signals to the FIFO input and output buffers, and a switch controlled by the routing and arbitration logic.

from the destination address stored in the first (header) flit of packet p_i. If there is no collision, the router reserves the connection from input to output port until the last flit of the packet p_i is forwarded. If there is enough space in the bounded FIFO input buffer of the next router, each following flit of packet p_i is forwarded immediately. This is known as wormhole switching [2]. In the following, d_r denotes the fix amount of time it takes for one flit to be forwarded from the input to the output port of a router without a conflict. If the input buffer of the next router and bounded FIFO output buffer of the current router are full, the link controller (LC) employs a router-level flow control mechanism to ensure that the current router does not accept new flits and a buffer overflow is prevented. Hence, a packet can be split over multiple routers. Since the upstream routers will eventually be blocked as well, the blockage propagates through the NoC. This mechanism is also known as backpressure [2]. If the backpressure reaches a node and its NI, respectively, no further packets can be injected into the NoC.

If multiple packets are present in one NoC, collisions can occur. A collision happens if multiple header flits request the same output port at the same time or if a router already reserved the output port requested by the header flit of a newly arriving packet for another packet. In the first case, the arbitration logic applies a round-robin early access scheme to select one packet to be transmitted. Early access means that a packet is forwarded immediately if there is no collision. The header flit and all following flits of the losing packets are stored in their input buffers. When the tail flit of the winning packet is transmitted, the conflict of the remaining packets is solved in the same manner. In case the output port was already reserved, no arbitration is required and the newly arriving packet is blocked until the tail flit of the other packet has passed. In the following, d_{rb} denotes the upper bound of time a packet is blocked by a collision with another packet in one router, i. e., the worst-case time it takes a router to arbitrate a conflict and process all flits of the winning packet.

III. Limited Injection Rate Approach

Our goal is to determine an upper bound of the latency of all transmissions of any traffic pattern. The latency of a transmission $l_\tau(\tau_i)$ is composed of 3 parts. i) The time required for the request packet p_i^{req} of the transmission τ_i to traverse the request NoC from $src(p_i^{req})$ to $dst(p_i^{req})$. ii) The time the destination node needs to process the request packet p_i^{req} and to inject a response packet p_i^{rsp}. iii) The time required for the response packet p_i^{rsp} to traverse back in the response NoC from $src(p_i^{rsp})$ to $dst(p_i^{rsp})$. So,

$$l_\tau(\tau_i) = l_p(p_i^{req}) + d_{dst} + l_p(p_i^{rsp}), \tag{1}$$

where $l_p(p_i)$ is the latency of the packet p_i in the request or the response NoC and d_{dst} is an upper bound of the time the destination node needs to process the request packet p_i^{req} and inject the response packet p_i^{rsp}.

The WCTL l_τ^{wc} specifies the maximum latency of all transmissions of any traffic pattern, so $l_\tau^{wc} = \max_{\forall \tau_i \in \mathcal{T}} l_\tau(\tau_i)$. According to (1), the WCTL l_τ^{wc} is only achieved if the latency of the request packet and the response packet are both maximal. The worst-case packet latency (WCPL) in one NoC $l_p^{wc} = \max_{\forall p_i \in \mathcal{P}} l_p(p_i)$ is the same for the request and the response NoC, since they are identical. Hence, the WCTL is

$$l_\tau^{wc} = 2l_p^{wc} + d_{dst}. \tag{2}$$

Note that (2) is an upper bound since the request and the response packets of one transmission in general do not both experience the worst-case in each NoC .

Furthermore, we want to be able to adapt the traffic pattern online, i. e., we want to allow arbitrary streams of transmissions without any restriction of the source, destination, or injection time. However, in general this changes the latency of other transmissions. In the following, we show that an upper bound of the WCTL l_τ^{wc} in any traffic pattern can be found if the common injection rate f of all sources $n_{i,j} \in \mathcal{N}$ is limited by a certain bound. This maximum injection rate f is defined as $f = \frac{1}{l_\tau^{wc}}$. Thus, the minimum time each source has to wait before injecting the request packet of the next transmission is at least l_τ^{wc} clock cycles. Note that due to the definition of the WCTL l_τ^{wc}, any source will receive a response packet before injecting the next request packet. Hence, our approach is applicable to both synchronous and asynchronous NoCs.

For the sake of brevity, we define the system $\mathbf{S}$ that consists of two identical 2D-mesh wormhole switching NoCs, each with the dimension order (XY) routing policy, $x \times y$ nodes, bidirectional links, and FIFO buffers. Furthermore, each source in the system $\mathbf{S}$ obeys the maximum injection rate f.

In the following, we use some properties of the system $\mathbf{S}$ to determine the latency of a packet without any collisions. By adding the worst-case delay introduced by collisions with other packets, we derive the WCPL l_p^{wc}.

A. Traversal Delay

We start by analyzing the traversal delay $d_t(p_i)$, i. e., the latency of the packet p_i in one otherwise empty NoC of the system $\mathbf{S}$. The traversal delay $d_t(p_i)$ is the time required for the packet p_i to traverse the request or the response NoC from $src(p_i)$ to $dst(p_i)$ without any collisions. Due to wormhole switching, the traversal delay is

$$d_t(p_i) = h(p_i)(d_r + \frac{1\,\text{flit}}{b}) + \frac{s}{b}, \tag{3}$$

where d_r is the number of clock cycles required by a router to process one flit, b is the bandwidth of any link $\lambda \in \mathcal{L}$, and s is the size of a packet in flits.

Thus, the worst-case traversal delay d_t^{wc} of any packet $p_i \in \mathcal{P}$ in one otherwise empty NoC of the system $\mathbf{S}$ is

$$d_t^{wc} = \max_{\forall p_i \in \mathcal{P}} d_t(p_i) = (x + y - 1)(d_r + \frac{1\,\text{flit}}{b}) + \frac{s}{b}. \tag{4}$$

Due to the dimension-order (XY) routing protocol, the maximum number of routers to be traversed in each 2D-mesh NoC with $x \times y$ routers is $x + y - 1$, namely from one corner to the diagonally opposite corner. Note that the worst-case traversal delay d_t^{wc} holds for the request and for the response NoC, since both NoCs are identical.

B. Blockage Delay

As soon as packets of other transmissions are present in one NoC, the delay of each packet potentially increases due to collisions and blockages by other packets, respectively. In the following, we use some properties of the system $\mathbf{S}$ to derive the maximum blockage delay of any packet in one NoC of the system $\mathbf{S}$.

Lemma 1. *Two packets of an arbitrary traffic pattern collide at most once in one NoC of the system* $\mathbf{S}$.

Proof: Suppose two packets p_a and p_b collide once, i. e., they arrive in a router r and request the same link λ_c. The router r resolves this conflict by blocking one packet and forwarding the other one. The dimension order (XY) routing policy of the NoC implies that the remaining route of packet p_a and the remaining route of packet p_b in the NoC each have at most two parts A and B. The part A of both remaining routes is identical. It contains at least the link λ_c. In part A, no further collisions of packet p_a and packet p_b are possible due to the FIFO buffers and the order-preserving property of the dimension order (XY) routing policy. The part B of the remaining route of each packets exists if $dst(p_a) \neq dst(p_b)$. The dimension order (XY) routing policy ensures that the part B of both routes is different, thus no further collisions of packet p_a and packet p_b are possible. Without loss of generality, since the remaining routes of the packets p_a and p_b have at most two parts A and B which cannot lead to further collisions, two packets collide at most once in any router of one NoC. ■

Lemma 2. *For an arbitrary traffic pattern in the system* $\mathbf{S}$*, a packet collides at most once with other packets injected by another source in one NoC.*

Proof: Let the packet p_a be injected at t_a. Let p_b^i with $i \in \mathbb{Z}$ denote a set of consecutive packets injected by another source at t_b^i, with

$$t_b^{i+1} \geq t_b^i + l_\tau^{wc},\ i \in \mathbb{Z}. \tag{5}$$

Suppose the packet p_a collides with a certain packet p_b^x of p_b^i at t_c. According to the definition of the WCPL l_p^{wc}, the packet p_a and all packets p_b^i are present in one NoC of the system $\mathbf{S}$ at most for l_p^{wc} clock cycles. Therefore, the packet p_a can only collide with the packet p_b^x if

$$t_a < t_c < t_a + l_p^{wc} \text{ and} \tag{6}$$

$$t_b^x < t_c < t_b^x + l_p^{wc} \tag{7}$$

hold. According to Lemma 1, p_a collides at most once with p_b^x in one NoC. However, the packet p_a could collide with another packet of p_b^i. From (6) and (7) follows that $t_b^x - l_p^{wc} < t_a < t_b^x + l_p^{wc}$. From (5) we know that all previous packets p_b^{x-n}, $n \in \mathbb{N}^+$ are present in the NoC at most until $t_b^x - l_p^{wc} < t_a$. Likewise, any subsequent packet p_b^{x+n} is not injected into the NoC before $t_b^x + l_\tau^{wc} > t_a + l_p^{wc}$. Hence, all previous and subsequent packets of p_b^i cannot collide with the packet p_a in the same NoC. Therefore, packet p_a collides at most once with any packet p_b^i injected by another source in one NoC. ■

Due to the definition of the injection rate f, the number of packets present in the NoCs of the system $\mathbf{S}$ is limited. In the following, we prove that this also limits the maximum number of times any packet can be blocked.

Theorem 1. *For an arbitrary traffic pattern in the system* $\mathbf{S}$*, any packet* $p_i \in \mathcal{P}$ *collides at most* $xy - 2$ *times with other packets in one NoC.*

Proof: In one NoC of the system $\mathbf{S}$, there exist xy sources. According to Lemma 2, any packet $p_i \in \mathcal{P}$ collides at most once with any packet from another source in one NoC of the system $\mathbf{S}$. Due to the dimension order (XY) routing policy and bidirectional links, the destination node $dst(p_i)$ of the packet p_i cannot inject packets that collide with the packet p_i. Furthermore, the packet p_i cannot collide with any other packet $p_j \in \mathcal{P}$ injected by the same source $src(p_i) = src(p_j)$ if all sources obey the injection rate f. Thus, any packet $p_i \in \mathcal{P}$ collides at most $xy - 2$ times with other packets in one NoC. ■

The delay added by a collision $d_b(p_i)$ of the packet p_i with another packet not only depends on the traffic pattern, but also on the internal state of the router. Since a low level analysis of all relevant facts is complex and since we are only interested in the worst-case behavior, we assume a packet is always blocked. The upper bounded blocking delay d_{rb} is the worst-case delay experienced by a packet when colling with another packet, namely the time required by a router to arbitrate a conflict and process all flits of the other packet.

Corollary 1. *For an arbitrary traffic pattern in the system* $\mathbf{S}$*, the worst-case blocking delay* d_b^{wc} *of any packet* $p_i \in \mathcal{P}$ *in*

one NoC is at most

$$d_b^{wc} = (xy - 2)d_{rb}. \quad (8)$$

Proof: Based on Theorem 1, any $p_i \in \mathcal{P}$ collides at most $xy-2$ times with other packets in one NoC. Hence, the worst-case blocking delay d_b^{wc} of any $p_i \in \mathcal{P}$ is the number of collisions of the packet multiplied with the maximum amount of time it takes a router to arbitrate the conflict and process the winning packet. ■

C. Worst-Case Latencies

The WCPL l_p^{wc} in the system $\mathbf{S}$ is the sum of the worst-case traversal delay d_t^{wc} and the worst-case blockage delay d_b^{wc} of any packet, so

$$l_p^{wc} = d_t^{wc} + d_b^{wc}. \quad (9)$$

For a given set of parameters of the system $\mathbf{S}$, (2) calculates the WCTL l_τ^{wc} of any transmission in the system $\mathbf{S}$. The WCTL l_τ^{wc} defines the maximum injection rate $f = \frac{1}{l_\tau^{wc}}$. By definition, the injection rate limits the number packets present in the system $\mathbf{S}$ at any instant of time, which again limits the number of collisions and bounds the WCTL l_τ^{wc}.

D. Dynamic Traffic Pattern

In the following, we prove that the definition of the injection rate f allows to dynamically modify the traffic pattern, i. e., the stream of transmissions, without violating the WCTL l_τ^{wc}.

Theorem 2. *In the system $\mathbf{S}$, the source and destination of any future transmission can be changed without increasing the WCTL l_τ^{wc}. Furthermore, in the system $\mathbf{S}$, new transmissions can be added to the traffic pattern without increasing the WCTL l_τ^{wc}.*

Proof: The WCTL l_τ^{wc} calculated in (2) is upper bounded if all parts of (2) are upper bounded. The delay of the destination d_{dst} is upper bounded by definition. The WCPL l_p^{wc} is defined by (9). According to Corollary 1, any packet $p_i \in \mathcal{P}$ is blocked for at most d_b^{wc} clock cycles in the system $\mathbf{S}$. The worst-case traversal delay d_t^{wc} of any packet $p_i \in \mathcal{P}$ in one otherwise empty NoCs of the system $\mathbf{S}$ is upper bounded, since the maximum number of hops in each NoC is fix. Thus, l_p^{wc} and l_τ^{wc} cannot increase in the system $\mathbf{S}$. ■

IV. Experimental Results

In this section, experimental results evaluate our limited injection rate approach under different traffic patterns. The following experiments were performed on SoCLib [8], a cycle-accurate, bit-accurate simulator based on SystemC. It includes an implementation of DSPIN [9], a 2D mesh NoC that implements the dimension order (XY) routing policy, wormhole switching, bidirectional links, and FIFO buffers. We created a platform consisting of 4×4 nodes connected by two DSPIN NoCs to separate request and response packets, as shown in Figure 1. Each node contains a MIPS32El core and local memory without caches connected by a crossbar. Thus, node-local traffic does not affect the NoC. The parameters of the simulation platform are $x = y = 4$, $s = 3$ flits, $d_r = 3$ cycles, $d_{rb} = 4$ cycles, $d_{dst} = 2$ cycles, and $b = 1\ \frac{\text{flit}}{\text{cycle}}$.

A. Traffic Patterns

First, we describe the traffic patterns applied in the following experiments. Unfortunately, the traffic pattern that leads to the worst-case latency of a transmission in both NoCs is unknown. However, we assume such a traffic pattern exists in general in order to derive a safe upper bound of the WCTL.

In our experiments we use two well known traffic patterns. The first traffic pattern maximizes the latency of all transmissions with a common source in the request NoC. This *latency* traffic pattern is achieved if all sources inject transmissions to the same destination with their maximum allowed injection rate f. Since all sources inject transmissions to a common destination, all request packets potentially conflict in the last link connecting the destination node with the request NoC. The request packets with the highest number of hops from source to destination arrive at the last link after all other request packets and hence are potentially blocked by all other packets.

The opposite extreme traffic pattern balances the traffic optimally in the request NoC. This can be achieved if each source node $n_{y,x}$ injects transmissions to the destination node $n_{n-y-1,n-x-1}$. Since this maximizes the network's throughput, it is also known as the worst-case *throughput* traffic pattern [7].

B. Delay Measurements

In a first experiment, we measure the latencies of transmissions under the latency traffic pattern. Thus, all sources send request packets to node $n_{0,0}$. Due to the router-level flow control, a NI cannot inject new request packets into the request NoC if the input buffer of the first router is full. Since we want to capture the maximum latency of each packet, we set the buffer size of both NoCs to 50 packets, i. e., 150 flits. Since 50 transmissions are injected from each source, the NI is always able to inject a new request packet. Figure 3 shows the traces of node $n_{3,3}$, the source which injects the transmissions with the largest traversal delay, i. e., packets with highest number of hops. By inserting NOP instructions between two consecutive injections, we decrease the injection rate f until the latency of consecutive transmissions l_τ is constant.

1) Synchronous NoC: The upper two traces of Figure 3 show the latencies in a synchronous version of DSPIN, where the NIs wait on the response packet while the CPU is stalled. Without decreasing the injection rate (0 NOPs), the latency of the first transmission of $n_{3,3}$ is 138 cycles. This is due to sources closer to the destination which have a shorter traversal delay and inject multiple transmissions before the request packet of first transmissions of source $n_{3,3}$ arrives

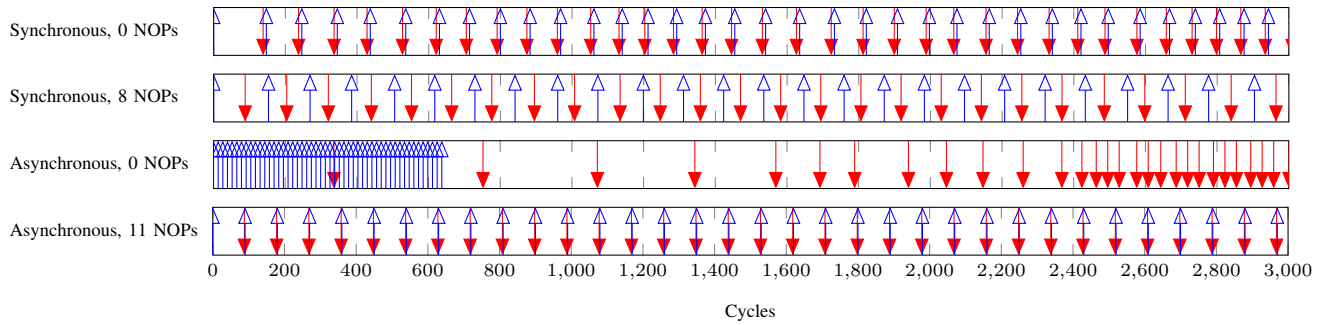


Figure 3. Trace of the first 3,000 cycles of source node $n_{3,3}$ in synchronous and asynchronous NoCs. All sources inject 50 transmissions to node $n_{0,0}$ with different injection rates (NOPs). Blue empty up arrows represent injected request packets, red filled down arrows represent arriving response packets.

at the destination. By decreasing the injection rate for all sources (8 NOPs), this effect is prevented. Hence, the latency of first transmission of $n_{3,3}$ is reduced to 88 cycles.

The blocking nature of the synchronous NoC has the disadvantage that the CPU is stalled during the transmission, while it could potentially execute memory-independent instructions. This time is different for each source since it depends on the traffic pattern. In order to achieve a common injection rate for all sources, the time the CPU stalls has to be subtracted from the additional waiting time added by software.

2) Asynchronous NoC: We adapted DSPIN's NIs to support asynchronous behaviour in order to alleviate the mentioned disadvantages. The lower two traces of Figure 3 show the latencies in the asynchronous version of DSPIN. Without decreasing the injection rate (0 NOPs), both NoCs are severely overloaded. In order to reduce the blocking effects, we decrease the injection rate by adding instructions between two injections (11 NOPs). This decreases the load of the NoCs and the latency of all transmissions of source $n_{3,3}$ is reduced to 88 cycles. Since the CPU is not stalled, more instructions are inserted as compared to the synchronous NoC.

C. Worst-Case Traversal Latency Evaluation

Next, we calculate the WCTL and compare it to the measured values of different traffic patterns. We use the parameters of the simulation platform and (2) to calculate the WCTL $l_\tau^{wc} = 176\,\text{cycles}$. According to the definition of the injection rate f, we define $f = \frac{1}{l_\tau^{wc}} = \frac{1}{176\,\text{cycles}}$. Due to disabled caches and the local memory latency, each NOP takes 7 cycles, so 176 cycles relate to 26 NOPs. In Figure 4, the measured WCTLs for different traffic patterns are plotted. With a delay of 26 NOPs between consecutive injections, none of the simulated traffic patterns had a WCTL higher than 176 cycles. Hence, the calculated WCTL is indeed an upper bound for all simulated traffic patterns.

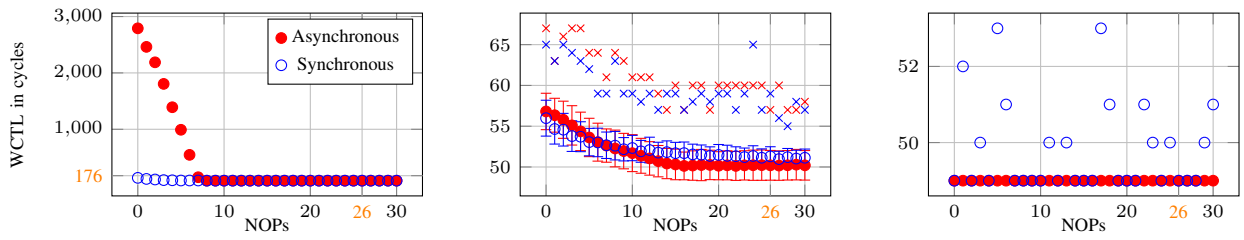
The latency traffic pattern results in large latencies, especially if the injection rate is not limited and the NoCs are asynchronous, as shown in Figure 4a. We performed 800 simulations with a random traffic pattern in which each source injects 1,000 transmissions to random destinations but not to itself. The mean, the standard deviation, and the maximum latencies are plotted in Figure 4b. However, none of the 800 runs with the random traffic pattern resulted in a latency in the range of latency traffic pattern. The throughput traffic pattern leads to a minimization of the measured latencies, as shown in Figure 4c.

The results show that the calculated WCTL l_τ^{wc} and the respective injection rate f are pessimistic compared to the measured WCTLs. The reasons are manifold: i) The simulated latency traffic pattern only results in the worst-case latency of some request packets, as explained in Section IV-A. ii) The worst-case behavior is assumed for each collision, i. e., a packet is always assumed to be blocked for the duration to forward an entire packet. In reality, a packet might win the arbitration or be blocked only be the last flit of another packet. iii) In the worst-case, a packet is blocked by packets from all other sources. In our implementation of the latency traffic pattern, all sources start injecting packets at the same time. Hence, the packet p_i^{req} with $src(p_i^{req}) = n_{3,3}$ is blocked by only 9 packets since the other packets are transferred before they could interfere with the packet p_i^{req}. iv) The latency of a router d_r is only 2 cycles if the link was free before. We calculated with 3 cycles, since it takes one additional cycle to detect whether a previously occupied link is free again.

D. Load Measurements

In this experiment, we compare the load of the NoCs generated by the calculated and the measured injection rates. We calculate the load of each link by dividing the number transferred flits by the number of cycles of the entire experiment. The load of the NoCs is the sum of the loads of all links in both NoCs divided by the number of links.

Figure 5 shows the measured load of the latency traffic pattern with different injection rates. With an injection rate $f = \frac{1}{176\,\text{cycles}}$ and 26 NOPs between consecutive injections, respectively, the load is 1.06% in asynchronous NoCs and 0.869% in synchronous NoCs. The first experiment shows that a delay of 11 NOPs between consecutive injections in asynchronous NoCs and 8 NOPs in synchronous NoCs,



(a) 50 injections per source under latency traffic pattern

(b) Mean, std. deviation, and maximum of 800 repetitions of the random traffic pattern

(c) 1,000 injections per source under throughput traffic pattern

Figure 4. Measured WCTL of various traffic patterns with different injection rates in synchronous and asynchronous NoCs.

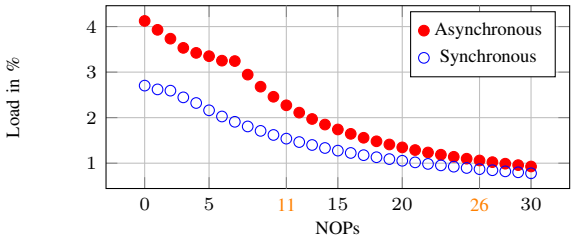


Figure 5. Measured load of the synchronous and asynchronous NoCs for different injection rates under the latency traffic pattern.

respectively, is sufficient to achieve a bounded WCTL. With a delay of 11 NOPs and 8 NOPs, respectively, the load is 2.27% in asynchronous NoCs and 1.81% in synchronous NoCs. Thus, for the latency traffic pattern our approach is pessimistic by a factor of 2.14 in asynchronous NoCs and 2.08 in synchronous NoCs. However, our approach considers an unknown traffic pattern that leads to higher latencies as the simulated latency traffic pattern. Furthermore, our approach allows the online modification of the traffic pattern.

V. Related Work

Several approaches to support GS communication on NoCs have been proposed. Most of them are based on hardware support or perform an extensive theoretical analysis. Only few solutions focus on the injection rate.

1) NoCs with Hardware Support: Most solutions that allow GS communication on NoCs are implemented in hardware. A common approach is to employ TDM [10], [11]. Typically, the time slots are assigned statically, depending on a specific traffic pattern. Shi and Burns [12] provide an offline schedulability analysis for priority-based NoCs with wormhole switching and the dimension-ordered (XY) routing policy. Based on this, several NoCs provide different priority levels to allow GS communication [13], [14]. Both the TDM and the priority level solution do not allow a dynamic modification of the traffic pattern without a new analysis of the slot or priority level assignment.

Algorithms for an online allocation of time slots in a TDM NoC have been presented by Kavaldjiev et al. [15] and Stefan et al. [16]. Another approach to enable dynamic GS communication is to provide an application programmable interface (API) to control the Quality of Service (QoS) features of the NoC by software, as proposed by Ruaro et al. [17] and Carara et al. [18]. Liu et al. [19] and Kranich and Berekovic [20] propose to use free bandwidth to send configuration packets that try to reserve resources for a new GS communication.

In contrast to all the above mentioned approaches, our approach does not require special hardware support like TDM, priority levels, multiple physical channels, or central configuration unit to support GS communication.

2) Analytic WCTL Derivation: Dasari et al. [21] concentrate on a very similar system model, namely commercially available NoCs without timing support in hardware. The authors propose the *Branch Prune and Collapse* algorithm to determine the WCTL. The algorithm is based on existing approaches that recursively analyze the contention at each router on the path of the analyzed flow of packets. The authors enhance these approaches by leveraging the input arrival patterns. Due to the blocking semantics, they identify a minimum inter-release time of packets and incorporate this into the recursive algorithm. However, the proposed analysis is very complex and depends on the traffic pattern.

Network Calculus is a powerful methodology to derive worst-case latencies. It uses an elegant abstraction with arrival curves and service curves on top of a min-plus algebra [22]. However, backpressure is hard to be integrated in network calculus that is designed for forward networks [21].

Qian et al. [23] propose a method to calculate the end-to-end delay bound of a flow on an NoC with wormhole switching and flow control. The method is based on network calculus and operates based on a contention tree, that covers direct and indirect contention by constructing complex contention scenarios from three basic contention patterns.

Network calculus does not support dynamic modification of the traffic pattern, i. e., the analysis is only valid for one specific traffic pattern. Hence, our approach and network calculus are inverse. Network calculus models one traffic pattern by arrival curves and allows to determine an upper bound of the latency of each transmission. Our approach shapes the traffic pattern by enforcing an injection rate in order to guarantee an upper bound of the WCTL.

3) Rate-Controlled NoCs: Ogras and Radu [24] address the congestion problem in NoCs by introducing a flow control mechanism that controls the injection rate at the

traffic sources. The authors propose an ON/OFF model, where packets are sent in a bursty manner during the ON phase and no packet is sent during the OFF phase. The distribution of the ON phases depends on the application. A state space model of the routers allows to predict the availability of the routers and is used to control the injection at source level. In contrast to our work, Ogras and Radu focus only on best-effort traffic.

Nychis et al. [25] examine congestion in a bufferless NoC. The authors propose a congestion control mechanism by measuring and influencing the injection rate per application. Similar to our approach, the key mechanism is the instructions-per-flit ratio. However, Nychis et al. focus on enhancing the system's performance and do not consider GS communication.

Compared with computer networks, our approach is similar to bandwidth management, traffic shaping, and rate limiting, which influence the injection rate in order to provide QoS guarantees. While such methods are common e. g., in industrial Ethernet, to the best of our knowledge, they have not been applied to NoCs.

VI. Conclusion

Considering a hardware model that is applicable to commercially available NoCs, we have proven that an upper bound of the latency of all transmissions can be guaranteed if all sources obey a maximum injection rate. This injection rate can be controlled entirely in software. If the application supports the limited injection rate, our approach enables GS communication on NoCs without explicit hardware support for guaranteed end-to-end latencies. Furthermore, we have proven that dynamic modification of GS communication is possible, as long as the common injection rate is not violated. Hence, our approach contributes to enabling adaptive features such as task migration on many-core processors under real-time constraints. Experiments evaluate our approach under different traffic patterns.

We plan to evaluate our approach on Adapteva's Epiphany and under real-world mixed-critical applications. Furthermore, we want to improve the load of the NoCs and the scalability of our approach by tightening the upper bound of the injection rate. NoCs with multiple VCs allow to map a subset of the traffic pattern to one VC and increase the injection rate. The proposed approach contributes to our long-term objective, the usage of many-core processors in dynamically reconfigurable real-time systems.

References

[1] S. Borkar, "Thousand core chips: A technology perspective," in *Proc. of DAC*, 2007, pp. 746–749.

[2] G. De Micheli and L. Benini, *Networks on Chips: Technology and Tools*. Elsevier Science, 2006.

[3] Tilera Corporation, "TILEPro64 processor – product brief," 2011.

[4] Adapteva Corporation, "Epiphany architecture reference (G3)," 2012.

[5] A. Hansson, K. Goossens, and A. Rădulescu, "Avoiding message-dependent deadlock in network-based systems on chip," *VLSI Design*, Article ID 95859, pp. 1–10, 2007.

[6] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Pub., 2003.

[7] B. Towles and W. J. Dally, "Worst-case traffic for oblivious routing functions," in *Proc. of SPAA*, 2002, pp. 1–8.

[8] N. Pouillon, A. Becoulet, A. V. de Mello, F. Pecheux, and A. Greiner, "A generic instruction set simulator API for timed and untimed simulation and debug of MP2-SoCs," in *Proc. of RSP*, 2009, pp. 116–122.

[9] I. Miro Panades, A. Greiner, and A. Sheibanyrad, "A low cost network-on-chip with guaranteed service well suited to the GALS approach," in *Proc. of Nano-Net*, 2006, pp. 1–5.

[10] C. Paukovits and H. Kopetz, "Concepts of switching in the time-triggered network-on-chip," in *Proc. of RTCSA*, 2008, pp. 120–129.

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

[12] Z. Shi and A. Burns, "Real-time communication analysis for on-chip networks with wormhole switching," in *Proc. of NOCS*, 2008, pp. 161–170.

[13] Y. Salah and R. Tourki, "Design and FPGA implementation of a QoS router for networks-on-chip," in *Proc. of NGNS*, 2011, pp. 84–89.

[14] T. Bjerregaard and J. Sparsø, "A scheduling discipline for latency and bandwidth guarantees in asynchronous network-on-chip," in *Proc. of ASYNC*, 2005, pp. 34–43.

[15] N. Kavaldjiev, G. J. Smit, P. T. Wolkotte, and P. G. Jansen, "Providing QoS guarantees in a NoC by virtual channel reservation," in *Proc. of ARC*, 2006, pp. 299–310.

[16] R. Stefan, A. B. Nejad, and K. Goossens, "Online allocation for contention-free-routing NoCs," in *Proc. of INA-OCMC*, 2012, pp. 13–16.

[17] M. Ruaro, E. Carara, and F. Moraes, "Runtime adaptive circuit switching and flow priority in NoC-based MPSoCs," *IEEE Trans. VLSI Syst.*, vol. PP, no. 99, pp. 1–12, 2014.

[18] E. A. Carara, N. L. V. Calazans, and F. G. Moraes, "Differentiated communication services for NoC-based MPSoCs," *IEEE Trans. Comput.*, vol. 63, no. 3, pp. 595–608, 2014.

[19] S. Liu, A. Jantsch, and Z. Lu, "Parallel probe based dynamic connection setup in TDM NoCs," in *Proc. of DATE*, 2014.

[20] T. Kranich and M. Berekovic, "NoC switch with credit based guaranteed service support qualified for GALS systems," in *Proc. of DSD*, 2010, pp. 53–59.

[21] D. Dasari, B. Nikolić, V. Nélis, and S. M. Petters, "NoC contention analysis using a branch-and-prune algorithm," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 3s, pp. 113:1–113:26, 2014.

[22] A. E. Kiasari, A. Jantsch, and Z. Lu, "Mathematical formalisms for performance evaluation of networks-on-chip," *ACM Comput. Surv.*, vol. 45, no. 3, pp. 38:1–38:41, 2013.

[23] Y. Qian, Z. Lu, and W. Dou, "Analysis of worst-case delay bounds for on-chip packet-switching networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 5, pp. 802–815, 2010.

[24] U. Y. Ogras and R. Marculescu, "Analysis and optimization of prediction-based flow control in networks-on-chip," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 13, no. 1, pp. 11:1–11:28, 2008.

[25] G. P. Nychis, C. Fallin, T. Moscibroda, O. Mutlu, and S. Seshan, "On-chip networks from a networking perspective: Congestion and scalability in many-core interconnects," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 407–418, 2012.