Design-Time Analysis of Real-Time Traffic for Networks-on-Chip using Constraint Models

Anderson R. P. Domingues*, Sergio Johann Filho*, Alexandre de M. Amory†, Fernando Gehm Moraes*
*School of Technology, Pontifical Catholic University of Rio Grande do Sul – PUCRS – Porto Alegre, Brazil
†RETIS Lab., Sant'anna School of Advanced Studies – Pisa, Italy
anderson.domingues@edu.pucrs.br, sergio.filho@pucrs.br, alexandre.amory@santannapisa.it, fernando.moraes@pucrs.br

2022 35th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI) | 978-1-6654-8128-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/SBCCI55532.2022.9893222

***Abstract*—Real-time networks-on-chips (RT-NoCs) were proposed to suit the needs of communication-intensive systems with real-time requirements. However, most of the current implementations found in the literature are based on custom routers, thus requiring a complete redesign of the interconnect architecture. This work presents a novel approach to tackle the analysis and scheduling of real-time traffic that requires no special mechanism to be implemented within the NoC design. Our solution relies on an auxiliary hardware module to synchronize the injection of packets into the network instead of custom routers, benefiting existing non-RT NoC designs. Our approach guarantees scheduled traffic to be congestion-free by using a design-time optimization process. Results present a didactic proof-of-concept using a synthetic application mapped onto a small NoC design.**

***Index Terms*—real-time analysis, scheduling, system simulation**

I. Introduction and Motivation

Networks-on-chips (NoCs) are the preferred interconnection media in many-core systems due to their potential for massively parallel communication and scalability while inserting low overhead for area and energy consumption into the design. The organization of a typical NoC mimics conventional computing networks, where routers provide the communication infrastructure, allowing multiple data streams to traverse the system simultaneously. As routers represent a tiny portion of the chip area, NoCs can suit the communication needs of resource-constrained systems.

An important aspect of NoC technology is the support for real-time communication provided by *real-time* (RT-) NoCs. RT-NoCs are particularly important in application domains such as control-theoretic systems, cyber-physical systems, and robotics. These applications, e.g., autonomous driving systems (self-driving cars), must comply with stringent safety constraints and rules [1, 2] as failures in these systems may result in catastrophic outcomes, e.g. financial loss, environmental damage, and risk to human lives. Supporting the operation of RT systems becomes even more pressing during the COVID-19 pandemic, where robots could replace humans to avoid potential contamination [3, 4]. In such a scenario, RT-NoCs appear as an alternative to guarantee timing in many-cores.

The community proposed RT-NoCs in the last two decades, as shown in Section II. In NoC-based systems, communication behavior becomes difficult to predict due to the non-determinism imposed by several components such as buffers, arbiters, routing algorithms, and internal cross-bar. Although it is possible to predict the timing behavior of some of these components individually, their models rarely scale to a system-wide level. Designers often assume a worst-case scenario and treat non-deterministic components as *black-boxes*. RT-NoCs mainly focus on eliminating black-box components by modifying routers to achieve determinism.

The *goal* of this work is to schedule real-time traffic in NoC-based systems, targeting non-RT NoCs. Our approach differs from the NoC RT literature as it requires no particular NoC architecture. Instead, we attach a time-controlled network interface (TCNI) to routers local ports to synchronize the time in which packets enter the network, enforcing contention-free traffic. The configuration of the TCNI modules matches the results of a fully automated, off-line optimization process, which makes our approach close to the ones adopted in contention-free NoCs [5, 6]. We dissociate our optimization model from the target NoC architecture zero-load latency (ZLL) model. The dissociation between both models makes it possible to apply our approach to different NoC designs.

We organize this paper as follows. Section II presents a short review on RT-NoCs. Section III presents an overview of our approach, its requirements, and its scope. We detail the adopted flow model in Section III-A, and the proposed optimization process in Section III-D. In Section IV, we present a didactic example for a synthetic application. Finally, we present the conclusion of this work in Section V.

II. A Short Review on RT-NoCs

We present a brief review of the state-of-the-art for RT-NoCs research. This section aims to guide the reader through the available approaches to guarantee real-time communication in NoC-based systems and highlight the community's interest in the topic. It is important to note that most RT-NoC studies rely on specialized architectures. Our work goes towards the opposite direction, as discussed throughout this paper.

A. RT-NoC Architectures

Hesham et al. [7] group NoC architectures into two categories: (i) best-effort, and (ii) guaranteed service. While best-effort NoCs focus on delivering correctness and completion of transmission, guaranteed service NoCs have performance bounds and predictability as their main concern. Guaranteed service NoCs are referred as RT-NoCs (when *timing* is the underlying non-functional requirement), further categorized as (i) ad hoc networks, (ii) packet-switching networks with priorities, (iii) circuit-switching networks, and (iv) time-division

978-1-6654-8128-1/22/$31.00 ©2022 IEEE

multiplexing networks. Please note that some designs might fall in more than one category, often named hybrid NoCs [7].

1) Ad hoc networks: One of the first approaches for achieving guaranteed response time is to handcraft an NoC design to meet specific timing requirements by dimensioning NoC parameters such as buffer width and depth. Xpipes [8] is among the first studies to explore ad hoc NoCs. However, manually tuning the NoC can be error-prone and time-consuming, aggravated by nowadays applications complexity.

2) Packet-Switched Networks with Priorities (PS-NoCs): Packet-switching is a connection-less communication scheme in which data traverse the network fragmented in one or more packets [7, 9]. In PS-NoCs, the path and transmission time of a packet are bound to the control flow mechanism, routing algorithm, arbitration rules, and traffic volume [7]. Since it is hard to predict the behavior of packet-switched NoCs [7], prioritization mechanisms were proposed [10]. These mechanisms permit high priority flows to preempt low priority flows, commonly implemented through virtual channels (VC) [11].

3) Circuit-switched Networks (CS-NoCs): In circuit switching, the network establishes a dedicated connection between source and destination routers so that packets can freely traverse the network without interruptions [12]. Three phases compose the circuit-switching process [7]: (i) path setup, where the network check for path availability and allocate the necessary nodes and links; (ii) data transfer; and (iii) path release. Another important concept in CS-NoCs lies in spatial-division multiplexing (SPM). In SPM, flows are physically separated, occupying one or more lanes, a subdivision of the communication channel between two routers [12].

4) Time-Division Multiplexing NoCs (TDM-NoCs): TDM NoCs can combine VCs and circuit-switching techniques in a single approach, simultaneously supporting best-effort and real-time flows to traverse the network concurrently [13]. For instance, in this approach, the reservation of routers in a path affects only a couple of virtual channels, so the path is still available to handle other flows. Time-slots orchestrate links sharing, whose assignments are kept into slot tables. Each router has its slot table, updated through configuration messages. These tables store flit information, which could be forwarded using either circuit- or packet-switching [14].

B. Scheduling Mechanisms in RT-NoCs

Heisswolf et al. [15] classifies scheduling mechanisms in: (i) synchronous TDM, (ii) asynchronous TDM, (iii) priority, (iv) round-robin, and (v) weighted round-robin. Except for Synchronous TDM, all other mechanisms perform asynchronous scheduling, which requires routers to implement buffering [15].

1) Synchronous TDM (S-TDM): The scheduling (path allocation) is performed over the same virtual channel for all routers in the path. Although it reduces the degree of freedom with the possibilities of paths becoming more restricted, it has simpler hardware when compared to asynchronous TDM (A-TDM). However, compared to A-TDM, S-TMD can reduce overall NoC utilization as it requires all virtual channels to be scheduled, even if they are idle.

2) Asynchronous TDM (A-TDM): Asynchronous TDM scheduling allows distinct virtual channels for path allocation. Consequently, the A-TDM overhead tends to be higher than S-TDM, although it has better overall utilization due to its flexibility on the path allocation.

3) Prioritization: As an asynchronous scheduling mechanism, prioritization does not require the reservation of virtual channels as higher traffic preempts lower priority traffic during runtime.

4) Round-robin (RR) and Weighted Round-Robin (W-RR): VCs are served one after another in round-robin scheduling, with guaranteed fair bandwidth distribution [15]. W-RR breaks service fairness and allows some VCs to be scheduled more often than others, using a weight parameter.

C. Other Classifications

1) Resource Allocation in RT-NoCs: Heisswolf et al. [15] categorize resource allocation as (i) static [16] and (ii) dynamic. For the former, resources are allocated to either real-time or best-effort traffic at design time. For the latter, allocation is performed at the runtime.

2) Requirements for Traffic Scheduling Algorithms in RT-NoCs: Grot et al. [17] and Stiliadis et al. [18] suggest some requirements for scheduling algorithms in RT-NoCs: (i) low end-to-end delays; (ii) fairness (flows with the same priority should have the same bandwidth reserve); (iii) isolation of flows (flows should cause none or minimum interference to each-other's delays); (iv) real-time composability; (v) efficient bandwidth utilization (network utilization should be maximized whenever possible); (vi) flexible bandwidth allocation; (vii) low performance overhead; $(viii)$ delay proportional to bandwidth usage; (ix) low area overhead; (x) low energy overhead; (xi) escalability; and (xii) facility of implementation.

D. Our Approach

We propose an optimization technique to schedule RT traffic in NoCs. Instead of designing another RT-NoC, we tackle the problem of real-time traffic scheduling by attaching a TCNI module to the target NoC, which potentially does not provide guarantees of periodic real-time communication by default. We configure TCNIs at design time as we assume the traffic parameters to be static for the application lifetime as this is the default behavior of the applications mentioned above (control-theoretic systems, cyber-physical systems, and robotics). Besides, we understand that best-effort traffic can be handled by an auxiliary NoC, thus we treat only RT traffic in our approach. Our approach also guarantees congestion-free traffic whenever possible. Additional advantages include: (i) no additional end-to-end delays, (ii) no virtual channels or priority system required, (iii) guaranteed isolation of flows, and (iv) fully-automated traffic analysis.

III. The Proposed Approach

Figure 1 presents our proposal, organized in five steps: (i) traffic analysis and characterization, (ii) flow unwrapping, (iii) network delay analysis, (iv) schedulability analysis and optimization, (v) TCNI configuration, and (vi) deploy or simulation. We present these steps throughout this section.

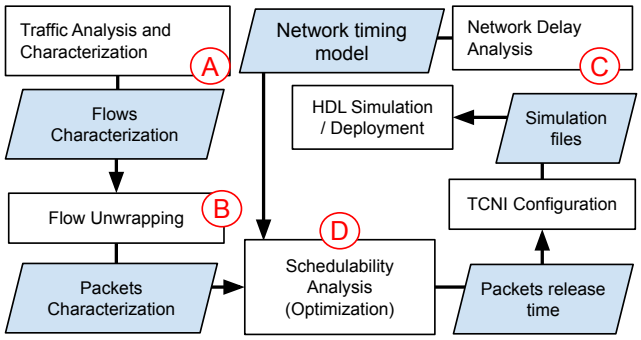


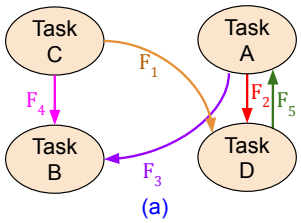
Fig. 1. The steps of the proposed approach (white rectangles), their outcomes (blue shapes), and Ⓐ – Ⓓ subsections of this section.

A. Traffic Analysis and Characterization

We model a *periodic* traffic flow as a 5-tuple $f_i = (p, c, d, s, t)$, where p is the period of the flow, c is the amount of data to traverse the network for each packet in that flow (worst-case packet size), d is the *relative* deadline, and s and t are the source and destination nodes for that flow. Period and deadline can be represented in any time unit, although we use cycles in Section IV. The same applies to data size (we use bytes as the unit). Source and target nodes are labeled according to their position in the topology. Finally, the flow model of the whole system is a set of flows $F = \{f_0, f_1, ..., f_n\}$.

We restrict our analysis to the hyperperiod. Ripoll and Ballester-Ripoll [19] define hyperperiod as "the smallest interval of time after which the periodic patterns of all tasks is repeated". We borrow this definition while replacing the word *tasks* by *flows*. In our context, the hyperperiod is given by the least common multiplier among the periods of all flows, i.e., $H = lcm(F_p)$. As the lcm function tends to "misbehave" for larger sets, we suggest practitioners to manipulate packets period to reduce the value of H as much as possible, e.g. adjusting periods to the nearest multiple factor.

Traffic characterization can be expressed by means of a (potentially) cyclic direct graph, or simply *digraph*. We label edges on flow data, and nodes on tasks, as Figure 2 shows. For each flow, information on their *period*, *data size*, and *deadline* must be presented.



Flow	Period	Data	Deadline
F_1	55	20	55
F_2	55	52	55
F_3	55	24	55
F_4	55	32	55
F_5	55	16	55

(b)

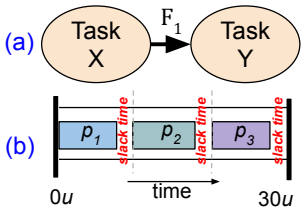
Fig. 2. Example of application characterization using a digraph. The application *synthetic-flow-A* comprises four tasks, labeled from A to D and represented by nodes in the graph. Edges indicate flows (a), and are labeled with the corresponding period, data size, and deadline information (b). In this application, flow have their period equals to their relative deadline, although this is not a limitation of our approach.

B. Flow Unwrapping

The set of all packets injected into the network during the hyperperiod is given by P, whose size $k = |P|$ is shown in Equation (1), where n is the number of flows and f_i^p is the period of the i^{th} flow $f \in F$. Each flow can be unwrapped to generate a finite number of packets for a given hyperperiod. Generated packets have the same data size, although they *necessarily* differ in their *minimum release time* and *absolute deadline*. A packet $p \in P$ is a 3-tuple $p = (f \in F, r, a)$, where f is the corresponding flow, r is the minimum release time, and a is the absolute deadline. By unwrapping all flows using an unwrapping function, we achieve the set of packets $P = \{p_0, p_1, ..., p_k\}$.

$$|P| = \sum_{i=0}^{n} \frac{H}{f_i^p} \tag{1}$$

For instance, one flow with a period equals 10 time units (u) would generate three packets for a hyperperiod of 30u. The first packet must be release after zero time units (at the startup). Thus, the minimum release time (p_i^r) for this packet is zero. The second packet must be released after 10u, and the last packet must be released after 20u, so they are evenly distributed within the hyperperiod. Absolute deadline is given by $p_i^a = p_i^r + f^d$, i.e. their minimum release time, plus the relative deadline of the corresponding flow. Finally, the deadline of packets p_1, p_2, and p_3 would be 10u, 20u and 30u, respectively. Figure 3 presents an example of flow unwrapping.



Flow	Period	Data	Deadline
F_1	10	8	10

Packet	Min. rel. t.	Data	Abs. D.
p_1	0	8	10
p_2	10	8	20
p_3	20	8	30

(c)

Fig. 3. Example application and flow unwrapping for a hyperperiod of 30u (a). Three packets were generated: p_1, p_2, and p_3 (c). Packets are evenly distributed in time (b).

C. Network Delay Analysis

We assume an NoC system to be a pair $S = (G, Z)$, where $G = E \times V$ is a directed graph representing the network topology, and Z is the set of all network resources. Edges (E) and vertices (V) represent network links and nodes, respectively. Full-duplex links must be denoted as two directed edges. We do not discuss half-duplex links in this work, although a single double-directed edge can represent them without harm to the approach. Finally, the links connecting network nodes to processing elements *must* be in the graph.

The routing function $\theta : V \times V \rightarrow A \subset E$ enumerates the path of links traversed by a packet departing from one node to another in the network. Such a function must be complete. We assume XY routing algorithm (function) for the rest of the paper, although any deterministic algorithm suits our approach. For the sake of simplicity, we also assume the network to

have a 2D-mesh topology, although our approach does not restrict the topology as long as the network has a digraph representation. Figure 4 depicts a digraph generated from an example network.

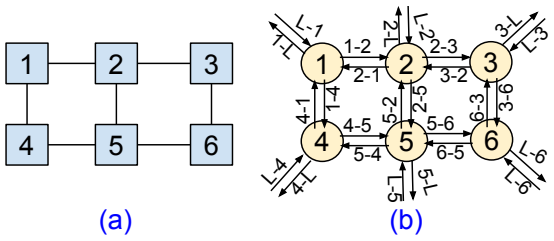


Fig. 4. Illustration of a 2D-mesh topology (a) and a digraph representing the same topology (b). Paths from/to local ports are treated as if they were connected to the same node L (omitted from the illustration).

The zero-load latency (ZLL) model is required for the given topology representation. This model is the only part of our approach that strictly depends on the NoC architecture. For instance, the target NoC (Section IV) permits multiple packets to traverse the same network node if their source and target ports are not the same due to its crossbar capabilities. Other system characteristics must be accounted for, e.g., link bandwidth, routing time, port arbitration.

The ZLL model for the target NoC is shown in Equation (2), where M is the Manhattan distance between source and destination nodes, q is the routing time, j is the number of flits to transport the data (payload). Routing time is up to 6 cycles, and packets must be routed once per node in the path ($M.q$). Links are 32-bit wide; one *flit* corresponds to 32 bits. Each router takes up to 6 cycles to route the first flit of the packet, and 1 cycle to route each following flit. Finally, the NoC protocol adds 2 flits of overhead (header). The first flit (header) takes $M.q$ cycles to route, the second flit (size flit) takes 1 cycle, and payload flits (j) take 1 cycle each.

$$ZLL = M.q + j + 1 \tag{2}$$

D. Constraint Model and Optimization

This step determines the allocation of resources for each packet to traverse the network during the hyperperiod. Each resource $z \in Z$ represents either a link or a router. In this work, we consider only links due to the capabilities of the NoC (Section IV). We define *occupancy* as the allocation time of a single resource in the system for the transmission of one packet. The goal is to determine the time window in which resources will be busy and organize packets so that the following constraints hold:

(**C1**) *Packet release time must be greater or equals than their minimum release time (3).* Since packets are evenly distributed in the hyperperiod, we cannot inject packets into the network until the last packet of that flow reaches its destination. Please note that resources are treated individually, and one packet may require more than a single resource at the same time. For this reason, $min\_release$ is represented as a matrix, denoting the *minimum* point in time where *each* resource can be allocated for a given packet.

$$\forall(z \in Z, p \in P)(release[z,p] \geq min\_release[z,p]) \tag{3}$$

(**C2**) *Two packets cannot share the same resource at the same time (4).* We call this constraint the "*nonoverleap constraint*", as it enforces resources to be allocated only to a single packet. This constraint can also be written as a *predicate* (5) on the release time and occupancy of packets.

$$\forall(p1, p2 \in P).nonoverlap(p1, p2) \tag{4}$$

$$\begin{aligned} nonoverlap(p_a, p_b \in P) = \forall(z \in Z).(\\ release[z,p_a] + occupancy[z,p_a] \leq release[z,p_b] \vee \\ release[z,p_b] + occupancy[z,p_b] \leq release[z,p_a]) \end{aligned} \tag{5}$$

(**C3**) *Each packet must meet its deadline (6).* Since we treat resources individually, each pair (p, r) (packet and resource) has its own deadline.

$$\begin{aligned} \forall(z \in Z, p \in P)(release[z,p] + occupancy[z,p] \\ \leq deadline[z,p]) \end{aligned} \tag{6}$$

(**C4**) *For the same packet, all resources must be allocated at the same time (optional).* This constraint forks our analysis into two possibilities. For the first, we could create a *strict* network model representing the behavior of links as close as possible to the real operation of the target NoC. This model will generate minimal waste (possibly none) of network bandwidth, although it can be tricky to develop depending on the target NoC (e.g., due to buffer modeling). The second possibility is to design a heuristic constraint, accepting some waste of network bandwidth. In the former case, the constraint carries out the complexity of the network, and the allocation time for all packets is only the necessary time for the packet to traverse without congestion.

For the sake of simplicity, we design a non-strict, heuristic model. The heuristic assumes that, if a link is necessary for the transmission of a packet, the link will be allocated during all the lifetime of that packet (7), roughly corresponding to the behaviour of circuit-switching (even for a wormhole network).

$$\forall(z_a, z_b \in Z, p \in P)(release[z_a,p] = release[z_b,p]) \tag{7}$$

Release Time Computation: Formally, *occupancy* is a function $O : P \times Z \rightarrow T \subset \mathbb{N}$, where P is the set of all packets in the model, Z is the set of all resources of the target, and T corresponds to the discrete-time domain. Occupancy gives us the amount of time that each packet requires from each of the resources in the subject system. The problem of "finding the release time of packets" can be interpreted as a variant of the classical "Job Shop Problem" (JSP), an optimization problem studied in the scheduling theory, where occupancy maps to the time spent by each machine to process a job. In our approach, machines are the network resources, and the packets are the jobs. The constraints enforce the many machines (links) to work on packets in the right order, thus representing the dependency between flits, to be transmitted one after another. The ZLL model represent the properties (and

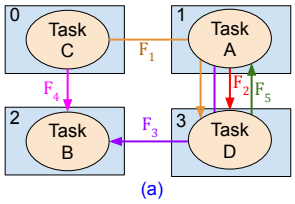
computing power) of each machine (links). The reduction of our problem to JSP is out of the scope of this paper as the proof can be extensive. However, it is important to note that JSP is an NP-complete problem and it may not be suitable for "large" problems. Finally, our approach aims to solve the problem formalized as "*Given the relations occupancy (O), minimum start, and deadline over $P \times R$, determine a relation $S : P \times R \rightarrow T$ that satisfies $C1$, $C2$, $C3$, and $C4$.*" It is important to note that more than one solution can be found in most cases, and the set of all valid solutions is not countable, although finite. We do not rank solutions as we propose a decision problem (not a minimization problem).

IV. Proposal Evaluation

This section illustrates our approach through a didactic example using a synthetic application on an open-source NoC design [20]. The NoC has the following features: 2D-mesh topology, input buffering, wormhole packet switching, 5 I/O ports, and round-robin arbitration.

A. Application Characterization and Optimization

We use the *Synthetic-flow-A* application to demonstrate the basic mechanics of our approach. Figure 5 shows the application characterization and task mapping for this application for a 2x2 NoC instance. Please note that our analysis is sensitive to task mapping, which is performed before characterization. Discussing task mapping is out of the scope of this paper since task mapping is a well-established field of study [21].



Flow	Period	Cp.	Flits	ZL latency
F_1	55	20	5	1q+5+1
F_2	55	52	13	2q+13+1
F_3	55	24	6	1q+6+1
F_4	55	32	8	2q+8+1
F_5	55	16	4	2q+4+1

*absolute deadline equals to period

(b)

Fig. 5. The Synthetic-flow-A application mapped onto a 2x2 NoC and its flows (a) and flows characterization including zero-load latency calculation (b).

Using Minizinc (https://www.minizinc.org/), we developed a constraint model description using Minizinc language. The inputs for the model consist of a table, shown in Table I. We excluded unused links from the input since it positively impact the performance of the solver. We also set the occupancy of all links to the latency of each packet, and the minimum release time values match the heuristic defined in Section III-D (**C4**).

Table II shows the results generated by Minizinc solver (Gecode 6.3). Since occupation and minimum start values for all links in the same packet are the same, the results indicate that all resources (links) must be allocated simultaneously. Although the target NoC does not implement a mechanism for allocating links, the packets can be released at the time indicated in the results for the first link in the path as there is no congestion in the network. Congestion avoidance is guaranteed by the *non-overlap* constraint in the model, thus the path is free during the allocation time.

As one can observe from Table II, the solver indicates that packets P_1 and P_3 must be released at cycle 33. Other packets must be released at the startup (cycle zero). Conflicting flows were set to enter the network one after another. In Figure 5 (a), we observe that flow F_3 conflicts with flows F_2 and F_4 because they share at least one link in their traversal path, including input/output operations of local links (omitted from Figure 5). The same stands for flow F_1, which also conflicts with flows F_2 and F_4. One possibility (the one generated by Minizinc) consists of injecting P_1 and P_3 prior to P_2 and P_4. Although P_2 and P_4 have distinct data sizes, they can be released in the network simultaneously as there is enough resource time to do so.

TABLE I
Synthetic-flow-A characterization.

	Pck.	Links											
		0-1	1-3	2-3	2-0	3-1	3-2	0-L	L-0	1-L	L-2	3-L	L-3
occupancy	P_1	18	–	–	–	–	–	–	18	18	–	–	–
	P_2	32	32	–	–	–	–	–	32	–	–	32	–
	P_3	–	–	19	–	–	–	–	–	–	19	19	–
	P_4	–	–	27	–	27	–	–	–	27	27	–	–
	P_5	–	–	–	23	–	23	23	–	–	–	–	23
min. start	P_1	0	–	–	–	–	–	–	0	0	–	–	–
	P_2	0	0	–	–	–	–	–	0	–	–	0	–
	P_3	–	–	0	–	–	–	–	–	–	0	0	–
	P_4	–	–	0	–	0	–	–	–	0	0	–	–
	P_5	–	–	–	0	–	0	0	–	–	–	–	0
deadline	P_1	55	–	–	–	–	–	–	55	55	–	–	–
	P_2	55	55	–	–	–	–	–	55	–	–	55	–
	P_3	–	–	55	–	–	–	–	–	–	55	55	–
	P_4	–	–	55	–	55	–	–	–	55	55	–	–
	P_5	–	–	–	55	–	55	55	–	–	–	–	55

TABLE II
Optimization results for Synthetic-flow-A

	Pck.	Links											
		L-0	0-1	1-L	1-3	3-L	L-2	2-3	3-1	L-3	3-2	2-0	0-L
alloc. begin	P_1	33	–	–	–	–	–	–	33	33	–	–	–
	P_2	0	0	–	–	–	–	–	0	–	–	0	–
	P_3	–	–	33	–	–	–	–	–	–	33	33	–
	P_4	–	–	0	–	0	–	–	–	0	0	–	–
	P_5	–	–	–	0	–	0	0	–	–	–	–	0

B. RTL Simulation, Discussion and Lessons Learned

As a proof-of-concept, we simulate the *Synthetic-flow-A* application at the register-transfer level (RTL) using Questa Sim [22], configuring the injectors following the results generated by Minizinc (Table II). The configuration consists of programming TCNIs to delay packets injection.

The waveform in Figure 6 shows signals for the local ports of routers in the target NoC system. We simulated 55 cycles, corresponding to the computed hyperperiod. For each processing element (PE), the figure shows their transmitting (tx) and receiving (rx) activity. Packets P_2, P_4 and P_5 are injected at cycle zero, and P_1 and P_3 are injected at cycle 33, following the results shown in Table II. No receiving signal is active after cycle 55, and no packets deadlines were missed.

The adopted constraint **C4** represents a pessimistic estimation of the resources usage, as seem in Figure 7. In practice, resources should be allocated one after another, following the path of the packet in the network. Similarly, resources should be released in that same order. By adopting the constraint **C4**, we allocate all the links in the path, increasing the network usage by m cycles per packet, where m is the Manhattan

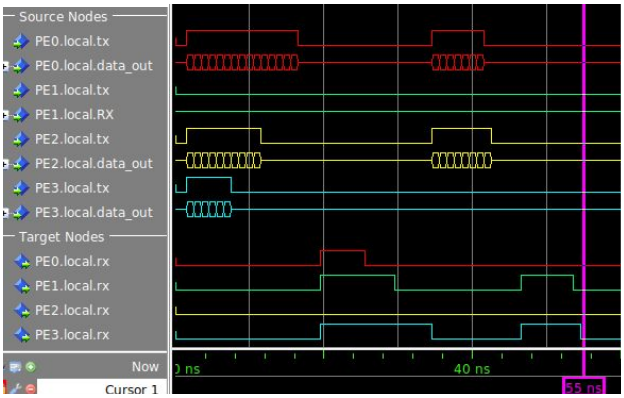


Fig. 6. Waveform depicting the activity of the local ports of routers in the target NoC. Colors denote signals of the same ports. Receiving activity (rx) is presented in the bottom waves. Local links, PE*x*.local, correspond to L-*x* in Figure 7.

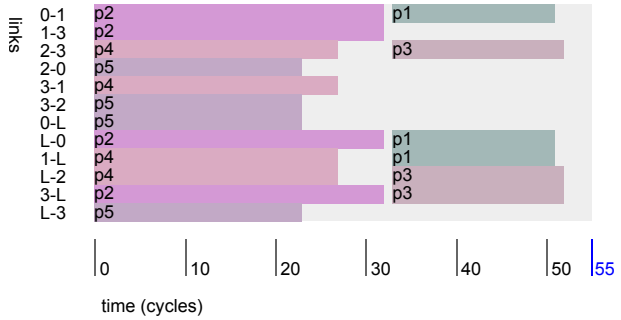


Fig. 7. Link utilization considering the allocation of all links in the path for all flows during the hyperperiod. The gray area represents the unused network time ($< 39\%$).

distance between source and target routers. The waste of networks resources increases as the distance between nodes increase (expected in large NoCs), although representing, in the worst case, $M+N$ cycles per packet, the worst Manhattan distance between two nodes in a 2D mesh-based NoC. We understand that the waste is negligible in applications with large packets size, e.g. the application shown in [23].

V. Conclusion

In this work, we presented an approach for scheduling real-time traffic in (potentially non-RT) NoCs. Our contributions include the presented models, the formulation of the optimization problem, and the automation tools. We demonstrate our approach for a synthetic application while briefly discussing the applicability of our approach for larger applications (e.g. [23]).

Future works include experimenting with larger applications, explore other NoC topologies and designs, and evaluate other approaches for searching the solution space (e.g. using heuristics). Next steps also include evaluating area and energy consumption overhead added to the design by the inclusion of the TCNI module.

Finally, it would come in hand to include real-time computation analysis in the approach. Our approach assumes that packets strictly enter the network at a given periodic time, and this can only be enforced at the origin of the packet, e.g. at the processing element attached to the TCNI.

Acknowledgment

This work was financed in part by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES (Finance Code 001), and CNPq (grant 309605/2020-2).

References

[1] International Organization for Standardization (ISO), "ISO 26262:9 — Automative Safety Integrity Level (ASIL)." [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:26262:-9:ed-2:v1:en
[2] AUTOSAR, "Standards – AUTOSAR." [Online]. Available: https://www.autosar.org/standards/
[3] National Geographic, "The pandemic has been good for one kind of worker: robots." [Online]. Available: https://www.nationalgeographic.com/science/article/how-pandemic-is-good-for-robots
[4] WeRobotics, "Teaming up with Pfizer on New Cargo Drone Project." [Online]. Available: https://blog.werobotics.org/2021/01/12/teaming-up-with-pfizer-on-new-cargo-drone-project/
[5] T. Picornell, J. Flich, C. Hernández, and J. Duato, "DCFNoC: A Delayed Conflict-Free Time Division Multiplexing Network on Chip," in *DAC*, 2019, pp. 1–6.
[6] D. Lee, B. Lin, and C.-K. Cheng, "Smt-based contention-free task mapping and scheduling on smart noc," *IEEE Embedded Systems Letters*, vol. 13, no. 4, pp. 158–161, 2021.
[7] S. Hesham, J. Rettkowski, D. Goehringer, and M. A. Abd El Ghany, "Survey on real-time networks-on-chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 5, pp. 1500–1517, 2017.
[8] D. Bertozzi and L. Benini, "Xpipes: a network-on-chip architecture for gigascale systems-on-chip," *IEEE Circuits and Systems Magazine*, vol. 4, no. 2, pp. 18–31, 2004.
[9] A. Mello, N. Calazans, and F. Moraes, *QoS in Networks-on-Chip – Beyond Priority and Circuit Switching Techniques*. Springer US, 2009, pp. 1–22.
[10] Z. Shi and A. Burns, "Real-time communication analysis for on-chip networks with wormhole switching," in *NOCS*, 2008, pp. 161–170.
[11] P. Mesidis and L. S. Indrusiak, "Genetic mapping of hard real-time applications onto noc-based mpsocs — a first approach," in *ReCoSoC*, 2011, pp. 1–6.
[12] P. T. Wolkotte, G. J. M. Smit, G. K. Rauwerda, and L. T. Smit, "An energy-efficient reconfigurable circuit-switched network-on-chip," in *IPDPS*, 2005, pp. 8 pp.–.
[13] Q. Ye, J. Liu, and L. Zheng, "Switch design and implementation for network-on-chip," in *HDP*, 2005, pp. 1–7.
[14] J. Yin, P. Zhou, S. S. Sapatnekar, and A. Zhai, "Energy-efficient time-division multiplexed hybrid-switched noc for heterogeneous multicore systems," in *IPDPS*, 2014, pp. 293–303.
[15] J. Heisswolf, R. König, M. Kupper, and J. Becker, "Providing multiple hard latency and throughput guarantees for packet switching networks on chip," *Computers & Electrical Engineering*, vol. 39, no. 8, pp. 2603 – 2622, 2013.
[16] T. Picornell, J. Flich, D. Jose, and C. Hernández, "HP-DCFNoC: High Performance Distributed Dynamic TDM Scheduler Based on DCFNoC Theory," *IEEE Access*, vol. 8, pp. 194 836–194 849, 2020.
[17] B. Grot, S. W. Keckler, and O. Mutlu, "Preemptive virtual clock: A flexible, efficient, and cost-effective qos scheme for networks-on-chip," in *MICRO*, 2009, pp. 268–279.
[18] D. Stiliadis and A. Varma, "Design and Analysis of Frame-Based Fair Queueing: A New Traffic Scheduling Algorithm for Packet-Switched Networks," *SIGMETRICS Perform. Eval. Rev.*, vol. 24, no. 1, p. 104–115, 1996.
[19] I. Ripoll and R. Ballester-Ripoll, "Period selection for minimal hyperperiod in periodic task systems," *IEEE Transactions on Computers*, vol. 62, no. 09, pp. 1813–1822, sep 2013.
[20] F. G. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip," *Integration, the VLSI Journal*, vol. 38, no. 1, pp. 69–93, 2004.
[21] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: survey of current and emerging trends," in *DAC*, 2013, pp. 1:1–1:10.
[22] Siemens, "Questa Advanced Simulator." [Online]. Available: https://eda.sw.siemens.com/en-US/ic/questa/simulation/advanced-simulator/
[23] Z. Shi, A. Burns, and L. Indrusiak, "Schedulability analysis for real time on-chip communication with wormhole switching," *IJERTCS*, vol. 1, pp. 1–22, 04 2010.