Multiple-objective Management based on a Distributed SDN Architecture for Many-cores

Marcelo Ruaro, Fernando G. Moraes
School of Tecnhology - PUCRS – Av. Ipiranga 6681, 90619-900, Porto Alegre, Brazil
marcelo.ruaro@acad.pucrs.br, fernando.moraes@pucrs.br

***Abstract*—The management of many-core systems is evolving to meet multiple objectives simultaneously. The Software-Defined Networking (SDN) has benefits explored in recent works that point it as a candidate to address this requirement at the communication level, at the same time that promotes management flexibility and reduced hardware complexity. Most of the research in SDN for many-cores assumes a centralized SDN (C-SDN) Controller and single-objective management. This work proposes multi-objective management based on a distributed SDN (D-SDN) architecture (SELF-SDN). The management is self-adaptive, addressing QoS and fault-tolerance simultaneously at the communication level. Experiments targeting QoS show that SELF-SDN provides a reduced amount of latency misses (-67%) and fast reaction time (-49.6%) to recover the QoS constraints compared to a C-SDN approach. Fault-tolerance experiments highlight the simplicity of the SDN paradigm to recover from faults in the NoC, not requiring additional hardware. Results related to multi-objective management demonstrate the fast reaction time of SELF-SDN to recover the communication latency faced to QoS loss and faults, reducing, on average, in 43% the reaction time compared to a C-SDN approach.**

***Index Terms*—Distributed Management, Many-core, Network-on-Chip (NoC), Software-Defined Networking (SDN)**

I. Introduction

Many-Core Systems-on-Chip (MCSoC) adopt Networks-on-Chip (NoCs) as the communication infrastructure due to its scalability and parallelism compared to buses. Packet-switching (PS) is the most commonly adopted switching method, which can be compared to "roads", allowing several flows to share the same link. Like roads, PS is subject to traffic, inducing congestion, and affecting the Quality-of-Service (QoS) constraints of applications. To ensure QoS, techniques available in the literature create "rails" over the NoC resources, by using circuit-switching (CS). Such rails are dedicated paths between communicating pairs, eliminating the interference between different flows, and ensuring high throughput with communication predictability.

Three main CS designs stand-out in NoCs [1]: (*i*) TDM, Time Division Multiplexing, link sharing in predefined time slots; (*ii*) SDM, Spatial Division Multiplexing, dynamic allocation of a set of wires for a given flow; (*iii*) MPN, Multiple Physical Networks, set of independent sub-networks (*subnets*) allocated at runtime.

The main CS design challenge, regardless of the adopted technique, is the path management at runtime, finding CS paths, and dynamically allocating the CS resources. The majority of techniques available in the literature adopt hardware-based management [2], which has as main advantage a reduced connection setup time [3]. However, it present drawbacks:

- *Hardware complexity*: dedicated hardware components to find paths, establish and release connections [3].
- *Fixed objective*: the path search algorithm follows a specific objective, as find the shortest path [4]. The lack of flexibility of hardware approaches inhibits the simultaneous exploration of multiple goals. For example, consider the search for shortest paths, load distribution, and fault tolerance simultaneously.

Hardware-based CS management is not suitable for large MCSoC, which increasingly request for reduced physical complexity and energy consumption. Moreover, *multi-objective resource management* is fundamental in complex systems, an ability that hardware-based management can hardly achieve.

The Software-Defined Networking (SDN) [5] is a concept that moves the communication management from the hardware level to the software level. In SDN, routers become programmable hardware units, with the ability to change its connections according to commands sent by an SDN Controller. This approach simplifies the on-chip communication architecture since it promotes a generic and straightforward communication design paradigm [2] [6], and also leveraging to self-adaptivity and multi-objective due to the high-level knowledge of the communication resources by the Controller.

The *goal* of this work is to exploit a state-of-the-art distributed SDN (D-SDN) architecture [7] to achieve multi-objective management. The original *contribution* of this work is a self-adaptive framework (SELF-SDN), able to ensure *QoS* for soft real-time (RT) flows and *fault-tolerance* at the link and router level.

II. Related Work and Motivation

Table I reviews SDN related works applied to MCSoCs. This research subject is recent, demonstrating an interest in adopting this paradigm to simplify the communication infrastructure. It is possible to classify SDN proposals in three categories (first column of Table I): (i) adoption of a centralized SDN (C-SDN) control [4], [6], [8]–[11]; (ii) distributed NoC control implemented inside each router or operating system (OS) [12], [13]; (iii) distributed SDN control (D-SDN), similar to C-SDN but with each Controller in charge of a cluster of resources instead to the whole system [7].

978-1-7281-9625-1/20/$31.00

TABLE I
RELATED WORKS ON SDN FOR NOCS AND MCSOCS (*generic*: PROPOSAL NOT CONCERNED WITH A SPECIFIC OBJECTIVE).

Classification	Work	Objective	Model	Summary
Centralized	Ellinidou et al. [6] 2019	Security	Mininet	Secure SDN configuration protocol of switches that implement the communication in a System-on-Package environment
	Kostrzewa et al. [4] 2018	QoS	RTL	Controller manages the NoC packet injection aiming to fulfill QoS of applications
	Sandoval et al. [8] 2016	generic	Noxim	Explore the SDN concept in NoCs, presenting a layered organization and performance experiments related to the NoC performance
	Berest. et al. [9] 2017	generic	OMNET++	Propose an SDN framework along with performance evaluations of the applications' execution time (no QoS bound)
	Fathi et al. [10] 2017	generic	RTL	Explores the SDN into NoC with experiments related to NoC performance
	Ruaro et al. [11] 2019	QoS	RTL	SDN Controller manages CS connection at runtime
Distributed (OS or Router level)	Scionti et al. [12] 2018	Power	RTL	SDN implemented as an instruction set architecture that allows the OS to control the network topology by partially or fully switching off unused links
	Cong et al. [13] 2014	generic	Noxim	Explore the SDN into NoC with experiments related to NoC performance
Distributed (Cluster-based)	— et al. [7] 2019	generic	RTL	Propose a cluster-based distributed SDN management with a distributed SDN synchronization protocol among clusters
	This work	*QoS Fault-tolerance*	*RTL*	*Multi-objective management framework based on distributed SDN*

Most works apply SDN without being concerned with a specific objective (*generic* in the third column of Table I). Other works adopt a single objective, like QoS [4], [11], security [6], and power [12]. A multi-objective SDN proposal is a gap observed in the literature, tackled in this work.

Note that the SDN paradigm differs from existing software-based management techniques already explored in NoCs, e.g., dynamic TDM [14] or SDM allocation [15]. Such techniques are focused on specific goals and do not support the dynamic change of its path search rules according to runtime constraints.

The motivations for electing SDN paradigm as a suitable technology to be adopted in multi-objective management for MCSoCs are the following [2], [4], [6]:

1) *hardware complexity reduction*: straightforward CS router design only requiring configuration support and packet forwarding, without the need for routing and arbitration;
2) *management flexibility*: SDN allows changing and updating of policies that define paths at runtime without the need to redesign routers;
3) *multi-objective management*: SDN allows multi-objective path management due to its software implementation;
4) *self-awareness*: SDN leads to a self-aware communication infrastructure due to the knowledge of the status of each router at a high-level of abstraction.

Despite the above benefits, SDN on MCSoCs presents the challenge related to the higher path setup latency compared to hardware-based approaches. Such a challenge is inherent to software-based approaches, with a path setup latency typically two orders of magnitude higher compared to hardware [2]. However, a fast setup latency is *not* a requirement considering the frequency of CS establishment. The SDN approach may control the CS establishment, if executed once at the beginning of the application execution, or when a monitoring process detects a QoS constraint violation (as detailed in Section III-B). The NoC infrastructure to support reserved paths requires a rich path diversity at a low cost. This work adopts a lightweight MPN with one PS subnet and a set of CS subnets. The use of MPNs is due to its smaller area and power compared to TDM and SDM [1].

Additionally, this work adopts D-SDN management that helps to reduce the path search latency compared to C-SDN. Cores and routers are grouped into clusters, with one SDN Controller per cluster. For local paths (i.e., intra-cluster paths), each Controller defines paths independently and in parallel to other SDN Controllers. For global paths (i.e., inter-cluster paths), Controllers cooperate in a synchronized way to manage the path establishment.

Thus, three strategies help to overcome the SDN challenges for MCSoCs: (*i*) small frequency of path setup to avoid the delay to setup connections; (*ii*) adoption of MPN, achieving low physical cost and rich path diversity [1]; (*iii*) a distributed management, enabling at the same time a global view of the NoC, and parallelism to setup intra-cluster paths.

III. SELF-SDN MANAGEMENT

This Section details the contribution of this work, the SELF-SDN management. Subsection III-A presents the MCSoC architecture supporting a D-SDN approach. Section III-B presents the SELF-SDN framework, resulted from the integration between D-SDN, MCSoC components, and multi-objective and self-adaptive management.

A. Distributed SDN Architecture for MCSoCs

Figure 1(a) overviews the MCSoC architecture supporting D-SDN [7]. It contains a set of Processing Elements (PEs) interconnected by an MPN NoC. Each PE has a CPU, a local memory, a Network Interface (NI), a PS router, and a set of CS routers. Figure 1(a) presents a 6x6 MCSoC instance, with four 3x3 clusters. Each cluster has an SDN Controller, running in a given PE.

Each router in a PE belongs to an independent network, named *subnet*, and there are no connections between subnets. Best-effort flows and management data share the PS subnet.

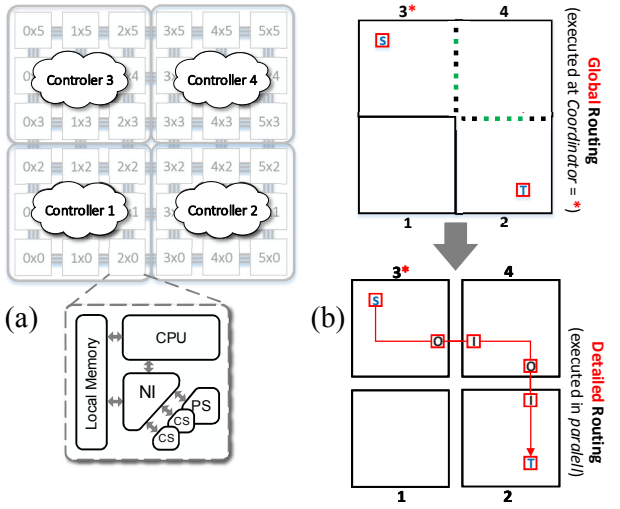

Fig. 1. (a) D-SDN architecture for MCSoCs, adapted from [7], (b) Hierarchical approach of the D-SDN protocol for global paths. Green dots in the global routing represent CS routers with available links.

Each PS router has input-buffers (usually 8-flit depth), credit-based control flow, and wormhole packet switching. A CS router has a configuration interface with the NI and an internal crossbar to connect an input port to an output port. The CS router costs, on average, 25% of the area and power of a PS router with the same flit width [2]. To create a path, the SDN Controller sends a configuration packet through the PS subnet to each CS router in the path. The NI handles the configuration packet, extracting its information and configuring the respective CS router of a given subnet.

Each cluster Controller locally creates intra-cluster paths. Global paths (inter-cluster paths) require a synchronization protocol among Controllers. Figure 1(b) presents an overview D-SDN protocol for finding global paths. The method is inspired in VLSI routing algorithms [16], starting with a global routing, responsible for finding the clusters where the path will traverse, followed by a detailed routing, which is performed in parallel by each Controller, and search paths inside each cluster.

Global paths require a consistent global state of the network. Thus, the Controller of the source PE becomes a temporary *coordinator* of the path establishment (cluster 3 in Figure 1(b)), managing the following phases of the protocol:

Phase 1 - Consistency. The coordinator achieves a global view of the network, exchanging messages with other Controllers. All Controllers send abstracted information of their cluster's borders status to the coordinator. The border status consists of the links' availability of CS routers located at cluster borders.

Phase 2 - Path search. The path coordinator executes the global routing (Hadlock algorithm at cluster level [16]), selecting the clusters that the path will traverse. Figure 1(b) overviews the process. After selecting the clusters (clusters 3, 4, and 2), the coordinator sends a message to the involved Controllers to execute in parallel the detailed routing. The detailed routing searches the input (I) and output (O) routers of the cluster, and a path between I and O (Hadlock algorithm at router level). When a Controller finishes the detailed routing, it sends the result to the coordinator, informing about path success or failure. If one of the Controllers fails in the detailed routing, the coordinator either can choose to re-execute the global routing using a different subnet, or it concludes that such a path cannot be created. If successful, the protocol advances to the last phase.

Phase 3 - Path Configuration. The path coordinator transmits the order to the involved Controllers to configure its CS routers. When all Controllers reply the configuration to the coordinator, it concludes the global path protocol, releasing all Controllers to execute new path searches.

The path release has the advantage of dismissing a synchronization protocol and the physical configuration of CS routers. As the CS routers allocation status is kept at the software level, each Controller only needs to update the status of a router as free.

B. Multi-objective and Self-adaptive Management

This Section presents the SELF-SDN framework, depicted in Figure 2. Each cluster has a dedicated PE for management purposes, named M_{PE}, and a set of PEs executing user's tasks.

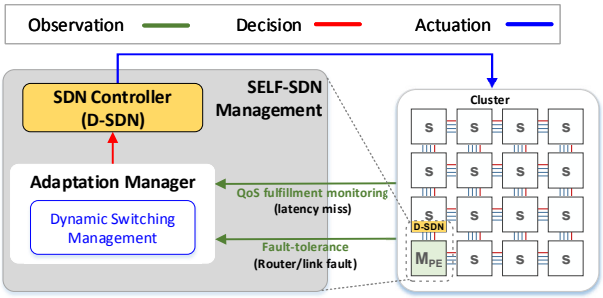

Fig. 2. Overview of the proposed SELF-SDN management.

The SELF-SDN management adopts the ODA (Observe, Decide, Act) paradigm [17]. The ODA paradigm comprises a loop that is always aware of the system status. It is generic and can be adapted to different many-core architectures. The left part of Figure 2 presents the SELF-SDN components and the right part of the Figure the cluster view, with the M_{PE} and the SDN Controller (Controller). The Controller is implemented as a high priority task that executes in a different PE from M_{PE}, allowing the Controller to execute in parallel with M_{PE}. Thus, the management processes inside M_{PE} keep running while the Controller is working configuring paths into CS routers.

SELF-SDN takes decisions according to observation messages. Green arrows in Figure 2 represent the following observation messages:

1) *Latency miss*: message sent by the OS of each PE reporting a communication latency violation;
2) *Fault notification*: message sent by a fault detection mechanism reporting a permanent fault in links or routers. The fault detection mechanism is out of the scope of this work, examples of methods available in

the literature include invariance checking at router level [18] and cyclic redundancy check (CRC) at the link level [19].

The Adaptation Manager (**AM**) decides the moment to trigger adaptations according to its awareness about the system and application provided by the observed information.

Quality-of-Service Adaptations. QoS is provided to soft real-time tasks. Communication between two real-time tasks forms a communicating task pair – **ctp** = $\{s, r\}$, with a sender task s sending messages to a receiver task r.

The SELF-SDN adopts the following threshold parameters to control QoS of a ctp (each ctp has thresholds defined according to the application constraints):

1) ***latency_TH***: maximum message latency, in clock cycles, for a given ctp.
2) ***latency_miss_TH***: maximum number of latency misses for a given ctp. Inversely proportional to the AM reaction time.

When an r task enters into the system, it has available the system call *SetLatencyContraint(sender_id, lat_th)*, where $sender\_id$ is the s ID, and lat_th is the *latency_TH* of the ctp. Such system call sets the *latency_TH* into the OS where the r task is running. An r task can invoke at any moment the *SetLatencyContraint()*, allowing the task to informs its QoS constraints at different communication loads and with different s tasks.

During system execution, the OS monitors the latency (based on the message's timestamp) of each received message addressed to r. If the latency is higher than *latency_TH*, the OS increments a latency miss counter for that ctp. When the counter reaches the value defined in *latency_miss_TH* the OS sends a latency miss to SELF-SDN and resets the counter.

The default communication mode between tasks is PS. When the SELF-SDN receives a latency miss message, it invokes the Controller to set a CS connection for the affected ctp. As the Controller abstracts the communication management for AM, the AM only executes a path request to the Controller and returns to its monitoring and decision activities.

The Controller receives the path request and, based on its knowledge of the MPN status, sets the path for the affected ctp. If the s task of the ctp is running in a different cluster than the r task, the D-SDN executes the protocol for global paths. Otherwise, the path is local, and the Controller searches and defines the path locally independent from the other Controllers.

Fault-tolerance Adaptations. A fault detection mechanism detects a permanent fault in a link or router, signalizes the fault to the NI, which sends the fault notification to the SELF-SDN.

If the faulty link or router is not assigned to any path, the SELF-SDN executes the following actions according to the fault location: (*i*) PS router: the PE is removed from the PE set since its CPU cannot communicate using PS; (*ii*) CS router or CS link: the fault notification makes the Controller avoid the faulty router by future paths; (*iii*) link: isolated by wrappers.

On the other side, if the fault breaks a current path of a ctp, the method considers the fault location and if there is a "broken" packet:

- fault location: (*i*) PS router: isolation of the router by wrappers, and use of adaptive routing; (*ii*) CS router: isolation of the router, with the Controller using a different subnet for the same path, or if not possible, searching and defining a new path; (*iii*) link: isolation of the port connected to faulty link;
- "broken" packet: the target PE detects an incomplete packet reception, requesting retransmission to the source PE. Before the retransmission, occurs the computation of the new path according to the fault location.

IV. Experimental Results

This Section first evaluates QoS (Section IV-A) and fault tolerance (Section IV-B) objectives individually. Section IV-C evaluates both objectives simultaneously. Section IV-D discusses the path search impact of SELF-SDN.

The work was implemented using the MCSoC plataform described in [20]. A SystemC-RTL description models the hardware. The software is modeled in C code (*mips-gcc* cross-compiler, version 4.1.1, optimization $O2$). The CPU is a MIPS processor running at 100 MHz. The C-SDN used for comparison is similar to D-SDN but with one "cluster" equal to the system size, making the C-SDN Controller always compute local paths.

A. Self-adaptive QoS Case Study

This experiment evaluates the capability of SELF-SDN to provide QoS by dynamically establishing CS for ctps, and also compares D-SDN to C-SDN. Figure 3(a)-(b) show the test-case mapping used in the experiments, in an 8x8 system, partitioned in four 4x4 clusters. Eight ctps run in the system, two at each cluster. Each ctp has an S task transmitting packets to a R task (blue arrows), with a $latency\_TH = 5.500$ clock cycles (cc). The $latency\_miss\_TH$ was set to 2 violations, aiming the AM to ignore random latency misses while keeps a fast reaction time.

Disturbing flows start at 15,000 cc, represented by red arrows. Figure 3(a) shows the D-SDN mapping, assuming one D-SDN Controller per cluster. Figure 3(b) shows the C-SDN mapping, with one C-SDN Controller.

Figure 3(c)-(d) presents the packet latency for each ctp. Initially, the ctps communicate using PS. When disturbing flows start, the latency of the ctp increases due to the interference on the PS subnet. The ctps start to generate latency miss messages to SELF-SDN, which decides to invoke the Controller to setup CS. After the CS setup, all ctps return to the expected latency since they are now using CS.

Comparing Figure 3(c)-(d), it is observable that D-SDN outperforms C-SDN. The D-SDN latency penalization time for each ctp is reduced, on average, 49.6% against C-SDN. This occurs because, in D-SDN, the management is performed in parallel, which increases the reaction time. Consequently, the total number of latency misses decreases in 67% with D-SDN

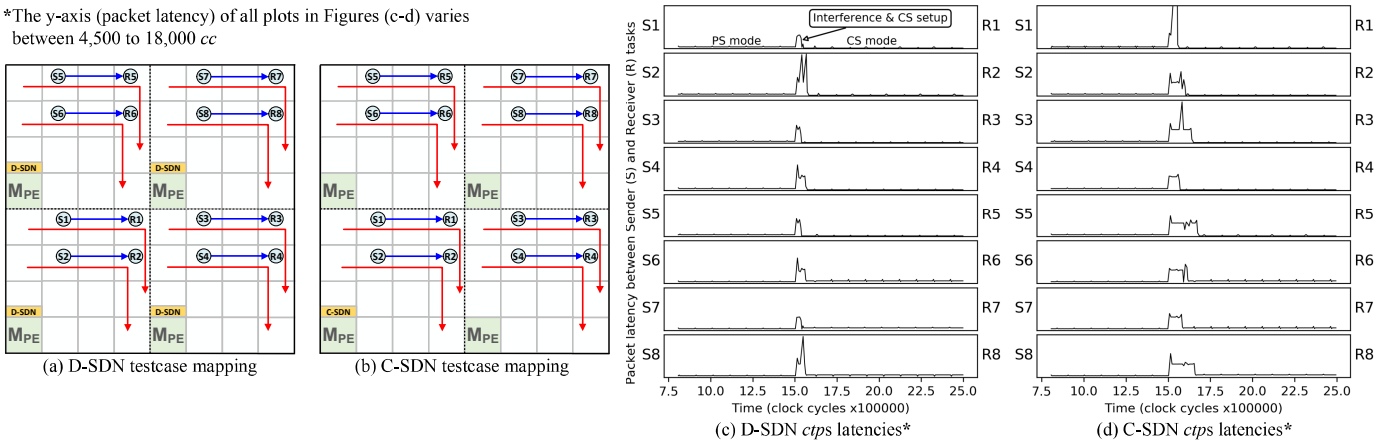

Fig. 3. Self-adaptive QoS support using dynamic CS establishment, for D-SDN and C-SDN.

(29 latency misses) when compared to C-SDN (88 latency misses).

This experiment shows that both approaches, C-SDN and D-SDN, may provide QoS for RT *ctp*s. The clear advantage of the D-SDN approach is scalability for large scale systems since it reduces the CS setup time, especially when there are multiple CS requests.

B. Fault-Tolerance Evaluation

The methodology of this experiment comprises a 3x3 cluster, with a synthetic 4-task application (p) mapped on it, and assuming all CS routers on PEs 1x2 and 1x1 as faulty at 3 ms. When the Controller receives the fault messages, it just excludes the faulty CS router from path search. Figure 4(a) presents the p mapping, where blue arrows denote the communication between tasks, without faults. SELF-SDN invokes the SDN Controller to establish CS connections to all flows at 5 ms, with the new flows represented by red arrows.

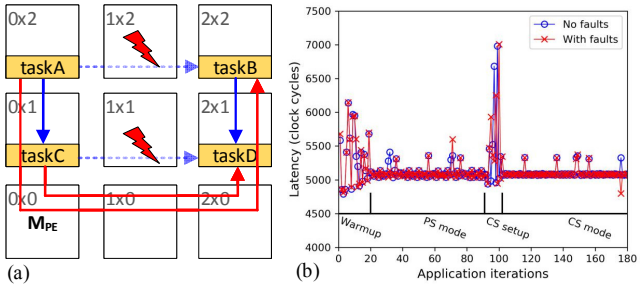

Fig. 4. (a) Mapping and communication flows of a synthetic application with faults and without faults (dotted paths). (b) Application iteration latency with faults and without faults.

Figure 4(b) presents the latency iteration measured at task D. This experiment results in the following periods: warm-up (iterations 0-20), PS mode (iterations 21-89), CS setup (90-105), and CS mode (106-186). The PS communication mode results in an average iteration latency of 5,102 *cc* for both scenarios. In the CS setup period, the scenario without faults has an increase in the latency of 4.5% against 5.4% of the scenario with faults. Such an increase of 0.9% comes from the higher computational complexity to find new CS paths due to the faults in the CS routers. After the CS establishment, the iteration latency is similar for both scenarios. Despite simple, this experiment confirms the effectiveness of the D-SDN Controller to deal with faulty routers.

C. Multi-objective Case Study

This experiment evaluates the multi-objective adaptation, handling QoS and fault-tolerance simultaneously. Figure 5(a) shows the MPEG benchmark mapping, where blue arrows represent the communication between MPEG tasks. Red arrows represent disturbing flows, and the two red rays represent faulty CS routers. Figure 5(b) presents the MPEG iteration latency measured at the *output* task. The latency threshold ($latency\_TH$) defined for each iteration corresponds to 58,000 *cc*, with $latency\_miss\_TH = 2$. The communication between tasks starts in PS mode. As there is no disturbing traffic, the NoC can meet $latency\_TH$. At 800,000 *cc* disturbing flows start, generating an increase in the iteration latency, and consequently, triggering a CS setup for the two affected *ctp*s ($input \rightarrow ivlc$, $iquant \rightarrow idct$). At 936,116 *cc*, both *ctp*s communicate using CS mode, restoring its QoS constraints. At 1,800,000 *cc*, faults occur at CS routers in the PEs executing $ivlc$ and $idct$ tasks, which makes SELF-SDN switch the affected *ctp*s to PS mode. As the disturbing traffic is still active in PS subnet, the iteration latency increases, generating latency misses and making the SELF-SDN to setup CS again for the same *ctp*s. As the D-SDN Controller was notified about the faulty CS routers, it sets CS using different subnets than previously. After 1,984,991 *cc*, both affected *ctp*s meet the QoS constraint latency by using CS.

Intervals $t1$ and $t2$ in Figure 5(b) corresponds to the SELF-SDN reaction time to restore QoS and recovery from faults. We evaluate the reaction time of the D-SDN and the C-SDN. Although both approaches can restore QoS and faults, the D-SDN delay (interval to search and configure a path) was, on average, 1,475 *cc*, resulting in a reduction of 43.2% on average compared to C-SDN.

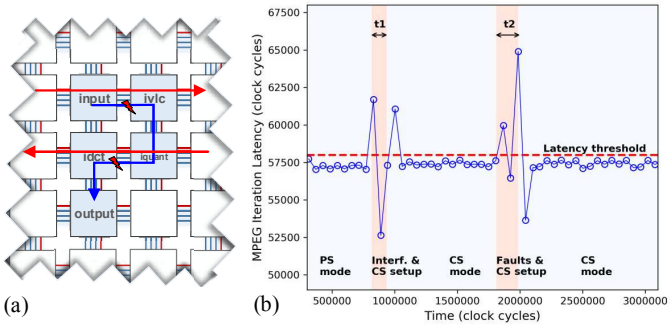

Fig. 5. (a) MPEG mapping: blue arrows represent the MPEG flows, red arrows the disturbing flows, red ray faulty CS routers; (b) MPEG iteration latency with multi-objective management.

The reason for explaining this difference comes for the larger search space of the C-SDN approach, which requires larger data structures, increasing its path search complexity.

D. SDN Delay Discussion

The SDN delay in searching and configuring paths varies according to the following characteristics: (i) path search algorithm; (ii) rules of the path search (minimal, energy-efficient, load balancing, etc.); (iii) centralized or distributed control; (iv) path setup frequency. Additionally, the computation of a path in a faulty region takes longer than in a faulty-free region (as exploited in Figure 4). Thus, we argue that SELF-SDN adopts suitable path search approach for large-scale MCSoC due to the following design principles:

1) Adoption of the Hadlock algorithm, ensuring shortest paths with reduced memory and computational complexity [16].
2) The D-SDN approach reduces the path search time since it has a reduced search space compared to C-SDN and can, most of the time, work in parallel with other Controllers.
3) The path setup frequency is performed when necessary faced to QoS degradation or faults and remains active during the ctp lifetime due to the rich path diversity of MPN [5].

ACKNOWLEDGEMENT

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. Fernando Gehm Moraes supported by FAPERGS (17/2551-0001196-1 and 18/2551-0000501-0) and CNPq (302531/2016-5).

V. CONCLUSION AND FUTURE WORK

This work showed the dynamism of the SELF-SDN framework based on D-SDN to meet multiple objectives simultaneously. The QoS evaluation showed that SELF-SDN management reduces latency misses due to the distributed SDN execution. It is worthwhile to mention that task mapping heuristics usually try to group communicating tasks close to each other to minimize congestion and latency. Thus, most paths between communicating tasks are local, favoring parallelism of D-SDN. The fault-tolerance evaluation demonstrated that SDN creates at runtime new paths, not requiring any further protocol or hardware support to deal with faulty-routers. The fault-detection mechanism notifies the SELF-SDN manager, which updates D-SDN Controllers to avoid faulty routers or links when computing new paths. The evaluation of QoS and fault-tolerance in a single scenario showed that SELF-SDN has the self-awareness of the system and application characteristics, which, combined with the high-level communication knowledge of the SDN approach, allows it to make efficient decisions and adaptations at runtime.

Future work includes extending management objectives by addressing power and energy (turning off unused CS routers) and security (allowing only authorized tasks to use a given CS subnet).

REFERENCES

[1] Y. J. Yoon, N. Concer, M. Petracca, and L. P. Carloni, "Virtual Channels and Multiple Physical Networks: Two Alternatives to Improve NoC Performance," *IEEE Trans. on CAD of ICs and Systems*, vol. 32, no. 12, pp. 1906–1919, 2013.

[2] M. Ruaro, H. M. Medina, and F. G. Moraes, "SDN-Based Circuit-Switching for Many-Cores," in *ISVLSI*. IEEE, 2017, pp. 385–390.

[3] S. Liu, A. Jantsch, and Z. Lu, "Parallel probing: Dynamic and constant time setup procedure in circuit switching NoC," in *DATE*, 2012, pp. 1289–1294.

[4] A. Kostrzewa, S. Tobuschat, and R. Ernst, "Self-Aware Network-on-Chip Control in Real-Time Systems," *IEEE Design Test*, vol. 35, no. 5, pp. 19–2, 2018.

[5] Y. Oktian, S. Lee, H. Lee, and J. Lam, "Distributed SDN controller system: A survey on design choice," *Computer Networks*, vol. 121, pp. 100–111, 2017.

[6] S. Ellinidou, G. Sharma, T. Rigas, T. Vanspouwen, O. Markowitch, and J. Dricot, "SSPSoC: A Secure SDN-Based Protocol over MPSoC," *Security and Communication Networks*, vol. 2019, no. 4869167, pp. 1–11, 2019.

[7] M. Ruaro, N. Velloso, A. Jantsch, and F. G. Moraes, "Distributed SDN Architecture for NoC-Based Many-Core SoCs," in *NOCS*, 2019, p. 8.

[8] R. Sandoval-Arechiga, R. Parra-Michel, J. L. Vazquez-Avila, J. Flores-Troncoso, and S. Ibarra-Delgado, "Software Defined Networks-on-Chip for multi/many-core systems: A performance evaluation," in *ANCS*, 2016, pp. 129–130.

[9] K. Berestizshevsky, G. Even, Y. Fais, and J. Ostrometzky, "SDNoC: Software defined network on a chip," *Microprocessors and Microsystems*, vol. 50, pp. 138–153, 2017.

[10] A. Fathi and K. Kia, "A Centralized Controller as an Approach in Designing NoC," *International Journal of Modern Education and Computer Science(IJMECS)*, vol. 9, no. 1, pp. 60–67, 2017.

[11] M. Ruaro, A. Jantsch, and F. G. Moraes, "Self-Adaptive QoS Management of Computation and Communication Resources in Many-Cores SoCs," *ACM Transaction on Embedded Computing Systems*, vol. 18, no. 7, pp. 1–24, 2018.

[12] A. Scionti, S. Mazumdar, and A. Portero, "Towards a Scalable Software Defined Network-on-Chip for Next Generation Cloud," *Sensors*, vol. 18, no. 7, pp. 1–24, 2018.

[13] L. Cong, W. Wen, and W. Zhiying, "A configurable, programmable and software-defined network on chip," in *WARTIA*, 2014, pp. 813–816.

[14] R. Stefan, A. Neja, and K. Goossens, "Online Allocation for Contention-free-routing NoCs," in *INA-OCMC*, 2012, pp. 13–16.

[15] A. Leroy, D. Milojevic, D. Verkest, F. Robert, and F. Catthoor, "Concepts and Implementation of Spatial Division Multiplexing for Guaranteed Throughput in Networks-on-Chip," *IEEE Trans. on Computers*, vol. 57, no. 9, pp. 1182–1195, 2008.

[16] N. Sherwani, *Algorithms for VLSI Design Automation*, 3rd ed. Springer, 2005.

[17] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal, "A generalized software framework for accurate and efficient management of performance goals," in *EMSOFT*, 2013, pp. 1–10.

[18] A. Prodromou, A. Panteli, C. Nicopoulos, and Y. Sazeides, "NoCAlert: An On-Line and Real-Time Fault Detection Mechanism for Network-on-Chip Architectures," in *MICRO*, 2012, pp. 60–71.

[19] A. Vitkovskiy, V. Soteriou, and C. Nicopoulos, "A Dynamically Adjusting Gracefully Degrading Link-Level Fault-Tolerant Mechanism for NoCs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 8, pp. 1235–1248, 2012.

[20] M. Ruaro, L. Caimi, V. Fochi, and F. Moraes, "Memphis: a framework for heterogeneous many-core socs generation and validation," *Design Automation for Embedded Systems*, vol. 23, no. 3, p. 103–122, Aug 2019.