# Software-Defined Networking Architecture for NoC-based Many-Cores

Marcelo Ruaro, Henrique Martins Medina, Alexandre M. Amory, Fernando Gehm Moraes
School of Technology, PUCRS University - Av. Ipiranga, 6681, Porto Alegre, Brazil
{marcelo.ruaro, henrique.medina}@acad.pucrs.br, {alexandre.amory, fernando.moraes@pucrs.br}

***Abstract*—The Software-Defined Networking (SDN) is a communication paradigm adopted in computer networking. The SDN assumes simple and programmable routers, removing the control logic from the routers' level, and assigning it to a high-level controller (software), which is responsible for defining the path of the communication flows at run-time. The controller can implement different communication rules to define the paths, as Quality-of-Service (QoS), fault-tolerance, and security. Many-cores may adopt the SDN paradigm due to its advantages: reduced hardware complexity, high reusability, and flexible management of communication policies. However, the challenge to apply the SDN may be the overhead for defining the paths in software against hardware-based approaches. The goal of this paper is to show that SDN can be a viable alternative for NoC management in many-core systems. This work proposes a generic SDN architecture for many-cores, detailing both the hardware and software designs. We compare the quality of the proposal with a state-of-the-art search path mechanism (hardware implemented), in a QoS case-study providing Circuit-Switching (CS) for applications. Results show that the SDN paradigm achieves similar performance than the hardware-based technique regarding path length. Hardware implemented mechanisms present a reduced latency to establish the paths. As the path establishment occurs once for each application flow, results show that the search path latency of SDN is not an actual drawback, as it could be expected.**

***Index Terms*—SDN (Software-Defined Networking); Many-Core; NoCs; MPN (Multiple Physical Networks); Communication management.**

## I. Introduction

Software-Defined Networking (SDN) [1] is a computer network paradigm that has as the central concept the routers' simplification. The diversity of routers available on the market made the process to configure and manage a network difficult, motivating the SDN development. Thus, SDN was conceived assuming simple architectures, moving the control logic from the router to a high-level manager, called Network Controller, implemented in software. With this paradigm, routers act as simple forwarding units, programmed by the controller at run-time according to network policies defined by the user or the network status.

The same scenario occurs today in the context of many-core systems. NoC designs commonly adopt large buffers, several virtual channels, and complex arbitration/routing schemes [2] to meet the applications' requirements. The complexity of current NoCs motivate us to explore SDN applied to many-core systems, with potential advantages to reduce the NoC cost (area and power [3]) concomitantly with a flexible management (e.g., QoS policies defined by software). Also, SDN can provide better reusability because routers are generic and simple hardware components, configured by software.

The path between any communicating pair in the system requires the configuration of the routers belonging to the path. Thus, the adoption of configurable routers incurs on the adoption of Circuit-Switching (CS), because it would be unfeasible to configure all routers in a given path for each packet injected into the NoC. As CS reserves the routers in a path, the NoC must adopt virtual channels [4], or Multiple Physical Networks (MPN) to enable simultaneous connections. According to the literature, MPNs has smaller area and power compared to virtual channels [2] [5].

The *goal* of this paper is to demonstrate that NoCs may adopt the SDN paradigm, with simple and programmable routers. To achieve this goal, this paper compares the connection establishment quality w.r.t a state-of-the-art CS search path mechanism called Parallel-Probe [6], which is hardware implemented. The SDN drawback is the latency to establish connections since the search path mechanism is software implemented. Such latency is evaluated, as well as the area required to support MPN.

Our *contribution* is a generic and flexible architecture to implement the SDN paradigm in many-core systems. The proposed SDN can enable flexible policies related to the communication management, as QoS (by allocating dedicated paths to the high priority flows), security (by deviating the traffic from secure regions [7]), fault tolerance (by reprogramming the already established paths at run-time). All these features can be implemented simultaneously and independently as a rule inside the NoC Controller, which manages the SDN-based communication on the chip.

## II. Related Works

Recent works address the SDN paradigm in many-core systems, as shown in Table I.

**TABLE I:** Related works on SDN architectures for Many-Core SoCs.

| Work | Impl. Details | RTL Validation | SDN-Controller |
|---|---|---|---|
| [8]-2014 | Few details | No | one per router |
| [9] [10]-2015,2016 | Arch. organization | Yes | one per system |
| [11]-2016 | Only router details | Yes | NA |
| This Work | Arch. organization and implementation | Yes (VHDL, SystemC) | one per system |

Cong et al. [8] propose a SDNoC architecture where the control plane is deployed as a distributed unity at each router. The routers' control plane exchange messages to implement the communication management protocol and to define the path for the flows. That work presents few details related to the architecture and no RTL validation. Sandoval et al. [9] propose an SDN organization with three layers: operating system, network operating system, and infrastructure. The work assumes routers that can have the routing algorithm defined by the SDN controller. Flows that are not managed by the SDN controller use the XY routing algorithm. The work [10] evaluated the configuration time for several routing algorithms, implementing them in the SDN controller. Results showed that the performance of the SDN to configure the routers varies according to the routing algorithm and the injection rate. For congested scenarios, worst results were obtained with adaptive routing algorithms. Scionti el al. [11] propose the SDN architecture to explore dynamic changes in the network topology. Each Processing Element (PE) has specific instructions to control the network topology by software, including switch off the links which are not used. The SDN paradigm is implemented by these specific instructions and not by an SDN Controller.

This work covers two gaps observed in the literature. The 1st one is a comprehensive SDN architecture, describing the hardware and software layers. The 2nd one is the SDN evaluation against a state-of-the-art hardware method for defining the paths.

## III. SDN Architecture Overview

Figure 1 presents the layered SDN organization. The application layer contains the users' applications. An application can be described

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

as a graph where nodes represent tasks, and the edges the communication flows. Tasks exchange data by using a communication protocol as MPI or open-MP. The middleware layer contains the embedded Operation System (OS) and the NoC-Controller ($NC$). The OS runs at each PE of the system. It abstracts the physical resources to the applications' tasks, providing the communication primitives and task scheduling. The $NC$ implement the SDN services to the OS. The OS can request to the $NC$ to define a communication path between a source and target PE. The $NC$ handles path requests from the OS, searches the path according to some predefined policy and notifies to the OS the path establishment result (success or failure). The bottom layer contains the physical network components.

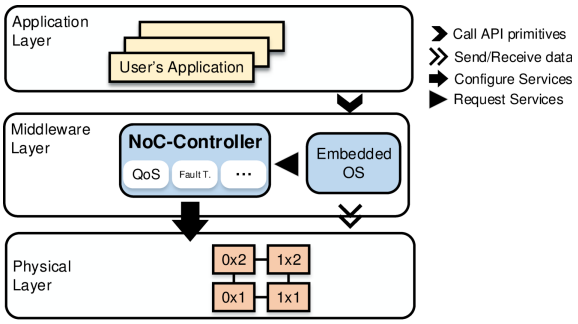


**Fig. 1:** Layered view for the SDN paradigm in a many-core organization.

**Definition 1.** $R$ - packet switching router.

**Definition 2.** $S_R$ - configurable SDN router.

**Definition 3.** *MPN* - multiple physical networks, corresponding to multiple, independent and parallel networks, consisting of many simple networks operating independently [2].

Figure 2(a) shows a standard many-core architecture, with PEs connected to $R$s (Def. 1). Figure 2(b) presents the SDN architecture, with a $NC$ managing the connection between $S_R$s (Def. 2). Figure 2(c) shows the integration of the SDN architecture to the many-core architecture. The communication architecture presented in Figure 2(c) corresponds to MPNs (Def. 3), with one packet switching (PS) network and a set of SDN networks. The PS network is used for management packets and to transmit data packets when there is no path between two PEs. It also has the role of configuring the $S_R$s.

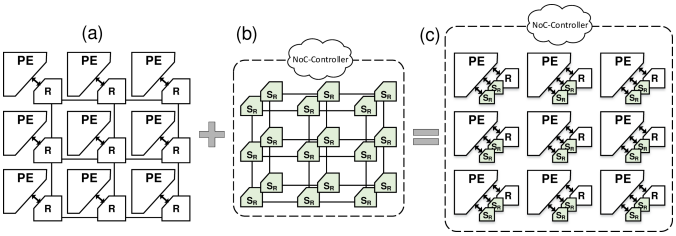


**Fig. 2:** (a) Standard NoC-based many-core architecture, (b) proposed SDN-based architecture (c) integration of the SDN in a NoC-based many-core architecture.

## IV. SDN Router Architecture and Configuration

According to the SDN paradigm, the $S_R$ should act as forwarding unit. To reduce area, input buffers are replaced by simple Elastic Buffers (EBs) [12] - Figure 3(a). The EBs retains data for one clock period, avoiding long wires, ensuring a reduced clock period. EBs also enable to reduce the silicon cost compared to a two-slot FIFO [12], once EBs need only one master-slave flip-flop instead of two.

In addition to the five EBs at the input ports, a $S_R$ contains two crossbars, to connect the upstream and downstream signals between input and output ports, configured by the Input Reservation Table (IRT) and Output Reservation Table (ORT), respectively. Each table is a 5-entry array (number of input ports) with 3 bits at each slot (enabling to store six states: E, W, N, S, L, Free). In Figure 3(b), the North input port is forwarding data to the East output port. The configuration interface programs the IRT and ORT tables. This interface is the *key* feature to make the router simple, avoiding logic for routing and arbitration modules. After configuring the $S_R$ routers, data is transmitted through CS.

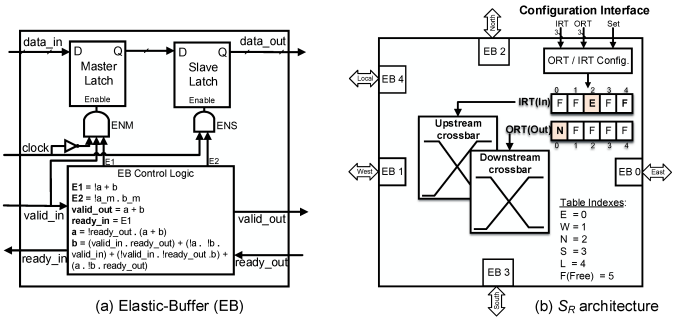


**Fig. 3:** (a) EB architecture. (b) $S_R$ implementation.

Figure 4 presents the main blocks of the PE (Local Memory, Network Interface (NI), CPU, and routers) and the process to configure a $S_R$. The SDN configuration is independent of the PE architecture once the configuration process does not include the NI. The $NC$ sends through the PS network a configuration packet to program the IRT/ORT tables. Each configuration packet has 3 flits: *header*, with the target address and a flag specifying that the packet must be consumed by a given $S_R$ and not by the NI; *payload size*, which is always 1; *configuration*, with 3 fields: input port, output port, SDN network number. It is not necessary to clear the IRT/ORT tables because the configuration process is managed by software. A new connection request releases the previous connection.

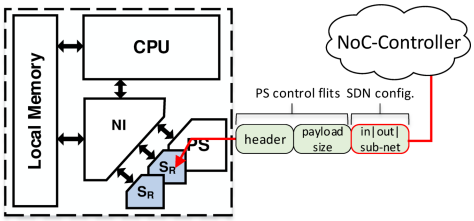


**Fig. 4:** PE architecture and configuration process of a $S_R$.

## V. Software Architecture

The software architecture concerns the implementation of the $NC$, which handles path establishment requests generated by the OS. As the $NC$ is decoupled from the OS (Figure 1), it can also handle path requests from other system's components. Algorithm 1 presents the pseudo-algorithm of the $NC$.

The algorithm continuously observes for new path requests (lines 1 and 2). If there is a request (line 3), the $NC$ calls the SEARCH-PATH algorithm at line 4. The role of the SEARCH-PATH algorithm is to define a path between a *source* and a *target* PE, implementing the *control logic* of the network (removed from the router to make it simple) according to a given path definition policy. The SEARCH-PATH algorithm returns the $path[]$, which consists of an array composed of the path routers' addresses, and the selected sub-net. If the path is valid ($path[] \neq \emptyset$), the $NC$ configures each $S_R$ of the path by sending the configuration packet (line 6). Next, at line 7, the $NC$ sends a *ack* message to the OS (requester). If the path cannot be defined ($path[] = \emptyset$), the algorithm sends a *nack* to the requester at line 9.

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

**Algorithm 1** NOC-CONTROLLER

```
Input: source address, target address
1: while TRUE do
2:     path_request ← read_path_request()
3:     if path_request = VALID then
4:         path[], subnet ← SEARCH-PATH(source, target)
5:         if path[] ≠ Ø then
6:             configure_SDN_routers(path[], subnet)
7:             send_ack_to_requester(subnet)
8:         else
9:             send_nack_to_requester()
10:        end if
11:    end if
12: end while
```

## VI. Experimental Setup

The case-study adopted to evaluate de SDN architecture is QoS provision at the communication level (guaranteed throughput). The goal is to provide at run-time support to establish CS for real-time applications flows. The SDN is compared to the *Parallel-Probing* (PP) method [6].

### A. Many-Core Architecture Overview

A clock-cycle accurate RTL model describes the many-core [13] (SystemC and VHDL implementations). Applications and OS are described in C language, compiled from C code and executed over the cycle-accurate models. The PE adopts the architecture presented in Figure 4 and 5(b). The many-core is divided into clusters. A cluster is a set of slave PEs ($S_{PE}$) managed by a cluster manager PE ($M_{PE}$). The $S_{PE}$ executes the applications' tasks. The $M_{PE}$ executes management routines, as dynamic task mapping and task reclustering (run-time reshaping of a cluster). Figure 5(a) presents a 6x6-3x3:3 many-core instance (system dimension: 6x6, cluster size: 3x3, 3 CS sub-nets).

To support run-time CS, the $M_{PE}$ is also responsible for requesting CS establishment. When an application enters into the system, the $M_{PE}$ request connections to the CS-Controller for each CTP (Communicating Task Pair). The application is allowed to execute when all its CTPs were handled by the CS-Controller.

The CS-Controller is a $NC$ specialized for communication QoS, mapped at the most central $S_{PE}$. It handles the CS requests from the $M_{PE}$ managing the process of CS establishment between a CTP. Two versions of the CS-Controller were implemented, one for the SDN method and the other one for the PP method.

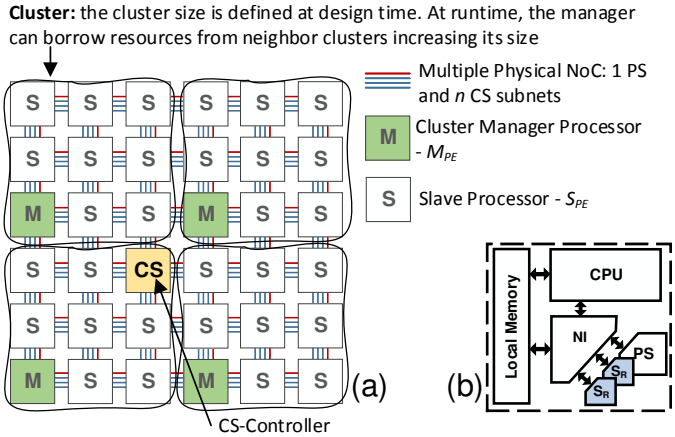


**Fig. 5:** Many-core architecture (a) with a hierarchical organization and MPN; (b) PE architecture.

### B. SDN Implementation

The SDN implementation follows the proposal presented in Sections III to V. The version of the CS-Controller addressing the SDN implementation uses the Hadlock's algorithm [14] to implement the SEARCH-PATH algorithm, which is a routing algorithm originally used in VLSI synthesis tools. This algorithm can find the shortest path in a 2D mesh network within a polynomial time of $O(n^2)$.

### C. Parallel-Probe (PP) Implementation

The CS-Controller addressing the PP implementation adopts the algorithm proposed by Liu et al. [6]. The PP method adopts a dedicated NoC responsible for finding the paths. As shown in Figure 6(a), each PE receives a PP router, connected to neighbors PP routers and locally to the $S_R$s (note that in this case the $S_R$s are configured by the PP router instead the CS-Controller). Figure 6(b) presents an example of the search method, with router 1 being the source and router 9 the target. The PP method finds the shortest path by propagating a wave of probes, which floods the PP network and unveils the shortest path by selecting the first probe to reach the target. When the first probe reaches the target, a backtracking process starts, releasing the other pre-allocated paths, and setting up the current path by configuring the $S_R$s. The PP method enables to find the shortest path within a constant setup time of 3*D+6 clock cycles, where D is the Manhattan distance between the source and target PP routers.

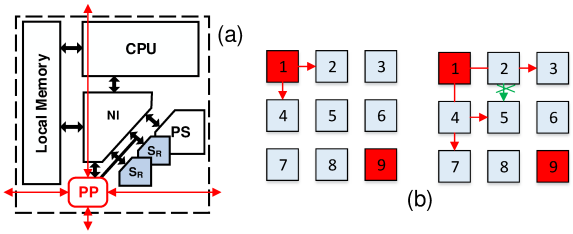


**Fig. 6:** (a) PE architecture including the Parallel-Probing router. (b) Example of PP algorithm [6].

The CS-Controller acts as a synchronizer in the PP implementation. As in the SDN implementation, the CS-Controller receives CS requests. These requests are stored in a FIFO because the PP network handles one propagation at a time. If there is a request in the FIFO, the CS-Controller handles it, by sending a message to the source $S_{PE}$ to start the PP method. The message contains the target address and the sub-net that the $S_{PE}$ should use (the sub-net is selected according to the sub-net utilization, selecting the less used sub-net). The $S_{PE}$ starts the PP propagation by configuring its PP router. When the propagation reaches the target, the backtracking process starts. During the backtracking, the pre-allocated $S_R$s not belonging to the path are released, and the $S_R$s belonging to the path are configured using the programming interface (Section IV). When the backtracking reaches the source PP router, it interrupts the $S_{PE}$. If the search fails, the $S_{PE}$ tries the next sub-net, until finding a path. When this process finishes, the $S_{PE}$ sends a message to the CS-Controller, reporting success or failure. As in the SDN implementation, after the search path process, the CS-Controller sends a *ack/nack* to the $M_{PE}$ (the CS requester).

## VII. Experimental Results

This section evaluates the connection establishment quality, the latency to setup connections, and the MPN and PP silicon area.

### A. Performance Evaluation

Table II presents the results, addressing many-core sizes from 36 to 256 PEs (1st col., system size-cluster size), with three CS sub-net configurations: 4, 6, 8 (2nd col.). Each $S_{PE}$ executes simultaneously two tasks. The evaluated scenarios execute several benchmarks instances (DTW, JPEG decoder, MPEG-2 decoder, VOPD) in such a way to have all $S_{PE}$s executing 2 tasks (system occupation equal to 100%), with the goal to stress the CS infrastructure. The 3rd column

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

**TABLE II:** PP and SDN evaluation, path length and connection time, for 6x6 to 16x16 many-core systems. Success rate: $(min\ hops\ +\ non\ min\ hops)/N\#\ paths$.

| System size: Cluster size | N# of SDN sub-nets | N# Paths | Method | Avg hops | Hops | | | Success Rate (%) | Connec. time (clock cycles) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | min | non min | not found | | Avg | Std dev | Max |
| 6x6-3x3 | 4 | 70 | PP | 2.25 | 65 | 2 | 3 | 95.71 | 450 | 237 | 1,248 |
| | | | SDN | 2.24 | 64 | 2 | 4 | 94.29 | 2,840 | 1,663 | 8,637 |
| | 6 | | PP | 2.31 | 66 | 4 | 0 | 100.00 | 435 | 252 | 1,800 |
| | | | SDN | 2.20 | 70 | 0 | 0 | 100.00 | 2,946 | 2,036 | 16,792 |
| | 8 | | PP | 2.20 | 69 | 1 | 0 | 100.00 | 377 | 214 | 1,800 |
| | | | SDN | 2.20 | 70 | 0 | 0 | 100.00 | 2,913 | 2,103 | 16,284 |
| 8x8-4x4 | 4 | 127 | PP | 2.57 | 113 | 9 | 5 | 96.06 | 467 | 265 | 1,252 |
| | | | SDN | 2.60 | 119 | 3 | 5 | 96.06 | 3,765 | 3,111 | 25,810 |
| | 6 | | PP | 2.48 | 122 | 5 | 0 | 100.00 | 420 | 215 | 1,801 |
| | | | SDN | 2.40 | 127 | 0 | 0 | 100.00 | 5,084 | 5,324 | 27,793 |
| | 8 | | PP | 2.45 | 124 | 3 | 0 | 100.00 | 401 | 199 | 1,511 |
| | | | SDN | 2.40 | 127 | 0 | 0 | 100.00 | 5,166 | 5,645 | 29,465 |
| 12x12-4x4 | 4 | 328 | PP | 3.03 | 269 | 28 | 31 | 90.55 | 501 | 311 | 1,247 |
| | | | SDN | 3.02 | 273 | 21 | 34 | 89.63 | 5,094 | 5,570 | 61,557 |
| | 6 | | PP | 2.85 | 304 | 20 | 4 | 98.78 | 443 | 288 | 1,835 |
| | | | SDN | 2.70 | 323 | 1 | 4 | 98.78 | 4,772 | 3,392 | 42,024 |
| | 8 | | PP | 2.76 | 318 | 9 | 1 | 99.70 | 406 | 243 | 2,431 |
| | | | SDN | 2.67 | 327 | 0 | 1 | 99.70 | 4,468 | 2,514 | 26,077 |
| 16x16-4x4 | 8 | 623 | PP | 2.62 | 608 | 12 | 3 | 99.52 | 423 | 289 | 2,486 |
| | | | SDN | 2.62 | 620 | 0 | 3 | 99.52 | 5,453 | 3,522 | 48,542 |

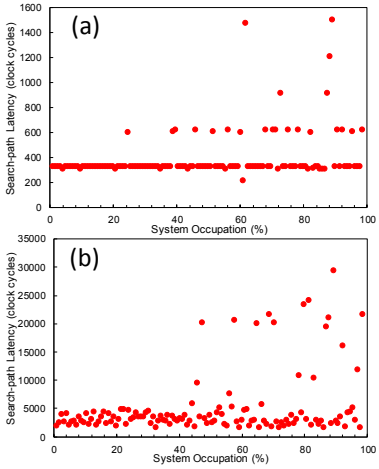


**Fig. 7:** Search path latency for PP (a) and SDN (b) - 8x8-4x4:8 system size.

presents the number of paths to connect, which is a function of the selected benchmarks, i.e., the total number of CTPs. The 4th column corresponds to the method: PP or the proposed SDN

The 5th column presents the *avg hops*, which corresponds to the average distance between CTPs. The SDN and PP methods present similar results (difference smaller than 5%), showing the effectiveness and scalability of the proposed software method compared to the hardware method. The reduced average number of hops is due to the mapping heuristic, which maps communicating tasks near to each other [15]. Next, the table presents the number of minimal, non-minimal, and not found paths. The SDN slightly overcomes the PP when evaluating the path length, since from 6 CS sub-nets all found paths were minimal for 6x6 and 8x8 systems, 98.5% for a 12x12:8 system, and 99.52% for a 16x16:8 system. The column *not found* is related to non-established paths (remember that when there is no path for a given CTP, the PS NoC is used). As expected, smaller number of CS sub-nets induces a larger number of failures. The 9th column corresponds to the success rate. For small to medium systems sizes, 6 sub-nets were enough to find all paths. For large system sizes (12x12 and 16x16), 8 CS sub-nets enabled to route more than 99% of the paths. Summarizing, the SDN method has a similar success rate to establish CS connections compared to the hardware implementation, with a slight advantage related to the path length (higher number of minimal paths).

The last 3 columns compare the latency to search the CS paths. The PP latency presents a small variation (small standard deviation values). On the other hand, the SDN latency tends to increases with the system size. The highest average latency was 5,453 and 501 clock cycles ($cc$) for SDN and PP, respectively. This is expected since the comparison occurs between software (SDN) and hardware (PP) implementations. If we assume CS connections established at the beginning of the application execution, with connections staying active during the application lifetime, the SDN search-path latency only impacts on the application startup. For example, consider a system running at 500 MHz ($T = 2ns$), an average latency equal to 5,000 $cc$, and an application with 10 CTPs. The total latency would correspond to 100 $\mu s$, and would not be noted by the end user. We argue that SDN can be a viable option for communication management, with reduced area and management flexibility, features that hardware-centric techniques are not able to provide.

Graphs in Figure 7 detail the search path latency (Y-axis) as a function of the system occupation (X-axis), for scenario 8x8-4x4:8. All other experiments present similar behavior. As mentioned, the PP latency presents a small variation (Fig. 7(a)). The average search path latency is 401 $cc$, and only 5 paths (3.9%) presents a latency higher than 630 $cc$. The latency increases when the available paths become scarce, inducing the search mechanism to explore alternative CS sub-nets. SDN presents a more significant variation in the search path latency (Fig. 7(b)), due to the features of the Hadlock's algorithm, which increases the search space according to the failures to set a given path. The latency stays below 5,000 $cc$ for 82.7% of the paths. As in the PP method, the SDN achieves worst-case latency when the system occupation increases, reaching 29,465 $cc$ in the worst-case.

### B. Area Evaluation

Consider a PS router configured as follows: 32-bit flit width, 8-flit buffer depth, round-robin arbitration, XY routing, no virtual channels. The area of this router (28 nm SOI technology @ 1GHz) is 10,021 $\mu m^2$. As reported in the literature, the adoption of 2 virtual channels (VCs) almost doubles the router silicon area of the PS router [16]. One 32-bit flit width $S_R$, as detailed in Figure 3, requires 2,011 $\mu m^2$. As the current work adopts 16-bit flits to reduce the MPN area, the $S_R$ requires 1,291 $\mu m^2$. Thus, the 1PS-8CS MPN has a silicon area equivalent to a 2-VC packet switching NoC. Such result demonstrates the low cost to adopt MPNs compared to TDM-based NoCs [2].

The PP router area is 800, 962 and 1,130 $\mu m^2$ for 4, 6, and 8 CS sub-nets respectively. Thus, the overhead of the PP router is equivalent to one $S_R$.

## VIII. Conclusion

This work investigated the pros and cons of the SDN paradigm, evaluating the proposal in a cycle-accurate many-core model, filling a lack in the literature by proposing a generic SDN architecture, addressing hardware and software implementation details. The reference hardware implementation (PP) enables fast connection establishment (small latency), with a small area overhead. Comparing the proposed SDN to the PP, we observe a similar path quality (i.e., average number of hops), with a slight improvement in the number of minimal paths, and higher latency. The higher latency is not an actual drawback since the latency only impacts on the application startup (in the order of $\mu s$). The advantages of adopting SDN include simple hardware architectures, reusability and management flexibility, features not available in hardware-centric approaches.

This work evaluated SDN for communication QoS. Future work includes the proposition and evaluation of other communication management policies using the SDN paradigm herein proposed, as fault-tolerance and security.

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

## ACKNOWLEDGEMENT

Author Fernando Gehm Moraes is supported by FAPERGS (17/2551) and CNPq (302531/2016-5), Brazilian funding agencies.

## REFERENCES

[1] Y. Jarraya, T. Madi, and M. Debbabi, "A Survey and a Layered Taxonomy of Software-Defined Networking," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 1955–1980, Fourthquarter 2014.

[2] 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, Dec 2013.

[3] M. Shafique and S. Garg, "Computing in the Dark Silicon Era: Current Trends and Research Challenges," *IEEE Design & Test*, vol. 34, no. 2, pp. 8–23, April 2017.

[4] S. Liu, Z. Lu, and A. Jantsch, "Costs and Benefits of Flexibility in Spatial Division Circuit Switched Networks-on-chip," in *NOCs*, 2015, pp. 1–8.

[5] E. Carara, F. Moraes, and N. Calazans, "Router Architecture for High-performance NoCs," in *SBCCI*, 2007, pp. 111–116.

[6] 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.

[7] M. M. Real, P. Wehner, V. Migliore, V. Lapotre, D. Göhringert, and G. Gogniat, "Dynamic spatially isolated secure zones for NoC-based many-core accelerators," in *ReCoSoC*, 2016, pp. 1–6.

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

[9] R. Sandoval-Arechiga, J. L. Vazquez-Avila, R. Parra-Michel, J. Flores-Troncoso, and S. Ibarra-Delgado, "Shifting the Network-on-Chip Paradigm towards a Software Defined Network Architecture," in *CSCI*, 2015, pp. 869–870.

[10] 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.

[11] A. Scionti, S. Mazumdar, and A. Portero, "Software defined Network-on-Chip for scalable CMPs," in *HPCS*, 2016, pp. 112–115.

[12] G. Michelogiannakis and W. J. Dally, "Elastic Buffer Flow Control for On-Chip Networks," *IEEE Trans. on Computers*, vol. 62, no. 2, pp. 295–309, Feb 2013.

[13] P. GAPH Group, "HeMPS web-site," http://www.inf.pucrs.br/hemps/, 2017, [Online; accessed 30-October-2017].

[14] F. O. Hadlock, "A shortest path algorithm for grid graphs," *Networks*, vol. 7, no. 4, pp. 323–334, 1977.

[15] 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. 1–10.

[16] A. Mello, L. Tedesco, N. Calazans, and F. Moraes, "Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC," in *SBCCI*, 2005, pp. 178–183.