A Monitored NoC with Runtime Path Adaptation

Edson Moreno, Thais Webber, César Marcon, Fernando Moraes, Ney Calazans
PUCRS University, Computer Science Department, Porto Alegre, Brazil
{edson.moreno, cesar.marcon, fernando.moraes, ney.calazans}@pucrs.br

***Abstract*—Networks-on-chip (NoCs) are already a common choice of communication infrastructure for complex systems-on-chip (SoCs) containing a large number of processing resources and with critical communication requirements. A NoC provides several advantages, such as higher scalability, efficient energy management, higher bandwidth and lower average latency, when compared to bus-based systems. Experiments with applications running on NoCs with more than 10% of bandwidth usage show that most of a typical message latency refers to buffered packets waiting to enter the NoC, while the latency portion that depends on packets traversing the NoC is often negligible. This work proposes a *Monitored NoC* called MoNoC, which is based on a monitoring mechanism and on the exchange of high-priority control packets. Practical experiments show that our fast adaptation method enables transmitting packets with smaller latencies, by using non-congested NoC areas, which reduces the most significant part of message latency.**

***Keywords*: NoC, monitoring, routing methods, adaptive control.**

I. INTRODUCTION AND RELATED WORK

NoCs constitute a communication infrastructure that offers a high degree of redundant paths. However, the overuse of some communication resources in them may increase the overall average message latency and compromise applications' execution time. This performance issue motivates the use of traffic monitoring and adaptive routing approaches.

Monitoring mechanisms may serve as a base to fulfill requirements such as latency minimization [1], throughput increase [2], energy consumption reduction [3], fault-tolerance [4], and thermal distribution [5]. Additionally, NoC monitoring schemes may be centralized [6] [7] or distributed [8] [9] [1]. Usually, the global view of centralized monitoring allows better results when applied to small NoCs. Nevertheless, as the NoC size increases, traffic load and latency for monitoring packets can significantly penalize this approach. Besides, it forms a hotspot region around the processing element (PE) dealing with monitoring packets, which can easily overload this PE. Due to this scalability issue, most NoC-based systems adopt distributed approaches for monitoring.

A monitoring architecture may be implemented in three ways: (*i*) using a lightweight dedicated network that operates in parallel with the main network [10]; (*ii*) sharing the main network, through the use of higher priority control packets [7]; (*iii*) or using distributed agents that detect local traffic behavior to infer and/or forecast the global system behavior [11]. The present work adopts the second way, because the use of control packets with higher priority usually leads to the smallest area overhead. Here, the target PE monitors packets and the source PE picks one of a set of pre-computed paths. Routing is deterministic and adaptive at the same time. During packet transmission the source-target path does not change, characterizing a deterministic routing. Upon receiving a control packet signaling congestion, a new path is adopted. As the path for different packets may change, routing is also adaptive. The *Monitored NoC* (MoNoC) adopts distributed monitoring, using high-priority packets to reduce the average latency at runtime in congested scenarios. The basis of the approach is a method for runtime path adaptation to avoid congested NoC regions.

II. MONOC ARCHITECTURE

Conceptually, MoNoC is a generic communication infrastructure in the sense that it is topology agnostic. It comprises mechanisms to perform traffic monitoring and policies for enabling the efficient use of NoC resources with the goal of achieving a fair workload distribution. This work assumes the use of a mesh topology, due to the nature of the routing algorithms used to pre-compute paths. MoNoC includes four main components: (*i*) routers and links; (*ii*) intra monitors; (*iii*) inter monitors; (*iv*) network interfaces (NIs).

The MoNoC architecture adopts the use of virtual channels and *source routing*. At design time, application tasks are mapped to PEs. For each communicating pair, the source PE computes a set of possible paths to each target PE. Path computation uses deadlock-free adaptive algorithms like [12].

Each router input port employs two virtual channels: (*i*) a ***data lane***, responsible for data packets transmission; (*ii*) a ***control lane***, responsible for control packets transmission. By default, the control lane has higher priority than the data lane.

The *Intra Monitor (***IAM**) is a non-intrusive circuit responsible for monitoring the use of each output port, and for transmitting this information when requested by a control packet. The *Inter Monitor* (**IRM**) on the other hand, executes end-to-end transmission rate computation between communicating pairs. It is also a non-intrusive module, but implemented at the NI. This rate is computed based on a parameterizable *Monitored Time Slice* (**MTS**). The IRM is employed only when a monitored communication is required. Otherwise, it remains computing link usage, which will eventually be requested. When establishing a monitored communication is, the master is the source PE, and the monitoring module name is *mstMonitor*. The slave is the target PE, and the monitoring module name is *slvMonitor*.

III. RUNTIME PATH ADAPTATION

Distinct communicating pairs may use multiple shared paths, which may overload communication resources and produce congestion. To overcome this, designers normally employ two strategies: (*i*) consider the traffic load of the entire network to compute an optimal distribution of all paths, which is carried out at design time, because it constitutes a time-consuming task; (*ii*) evaluate at runtime the current network status to take routing decisions.

MoNoC merges both design time and runtime strategies. For each communicating pair, a set of possible paths that link source and target PEs is computed at design time. At runtime,

whenever the required transmission rate is violated, MoNoC adapts the entire path of this communicating pair, selecting the less congested one among those computed at design time.

In the context of MoNoC, a *contract* refers to a transmission rate that a communicating pair requires. To keep this contract, MoNoC provides mechanisms to permanently observe the traffic load of each flow. It offers adaptive and non-adaptive communication approaches. When a non-adaptive communication is requested, no contract is necessary. However, if an adaptive communication is requested, a contract must be established for the communicating pair, and the transmission rate is monitored, to provide adaptability if necessary. Figure 1 illustrates with a message sequence chart the set of messages applied to the *contract session opening* of an adaptive communication.

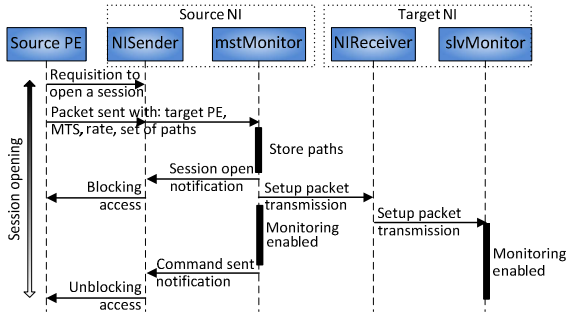

Figure 1 – Message sequence chart for a contract session opening.

To establish the contract, the source PE requests a session opening to the *NISender*, the NI module responsible for injecting packets into the NoC. Next, the source sends four items to the *mstMonitor*: (*i*) the target PE address; (*ii*) the MTS; (*iii*) the rate to transmit data during a given MTS, i.e. the *Agreed Contract* (AC) rate; (*iv*) a set of alternative paths. *mstMonitor* dispatches a setup packet to the *NIReceiver*, the NI module that receives packets from the NoC, which in turn requires the monitoring of the transmission rate by the *slvMonitor*. After reception of the setup packet by the *slvMonitor*, the session is opened. Then, every packet exchanged between source and target PEs is monitored, at both sides, to verify the maintenance of the requested rate. By default, the first path type used in communication is the XY routing path. The *contract* can only be cancelled by the source PE, which transmits a setup packet, releasing the IRM at the target NI.

A. Path Adaptation Protocol

A communication path between PEs may be adapted if the contract is violated. *slvMonitor* observes the transmission rate, by counting the number of received flits at each MTS. The target NI notifies the source NI in case of non-compliance. The source NI is responsible for firing the path adaptation procedure. Figure 2 presents the path adaptation protocol, which considers the following parameters: (*i*) the *Agreed Contract* rate, **AC**; (*ii*) the *Current Reception Rate*, **CRR**, computed at the *slvMonitor*; (*iii*) the *Average Injection Rate*, **AIR**, computed at the *mstMonitor*. Note that given this organization, path adaptation is fully transparent to PEs.

Once a contract is established as defined in Figure 1, the target NI starts the monitoring processes. At the end of each MTS period a comparison between CRR and AC occurs. If a violation is detected (CRR<AC), the *slvMonitor* freezes the monitoring process and notifies the violation to the *mstMonitor* (event 1, in Figure 2). When the source NI receives this notification, it tests AIR against AC. If the AIR < AC, this means that the source PE is injecting packets at lower rates (event 2). In this case, no adaptation is required, and the source NI notifies the target NI that the observed violation does not represent a violation at the NoC level, and monitoring processes at the target PE can continue. However, if the AIR at the source PE is being respected, but the target PE detects a violation, this means that the path is congested (event 3). This situation fires path adaptation and the source NI stops the *mstMonitor*, transmitting a set of control packets to the target PE, using the set of available paths (event 4).

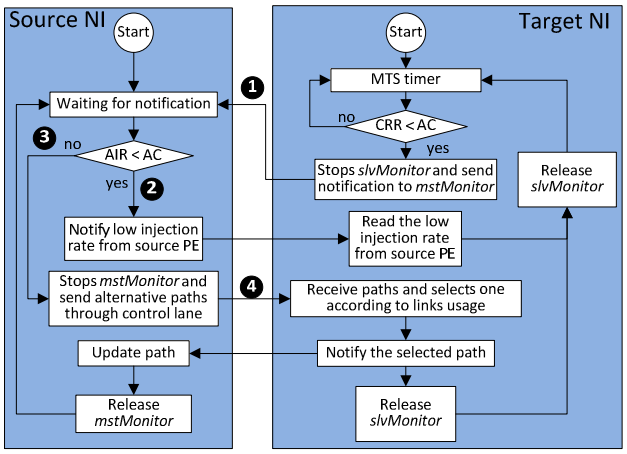

Figure 2 – Path adaptation protocol.

The Intra Monitors (IAMs) compute the average link usage at each output port. The source NI injects a control packet for each alternative path and at each hop of the path data computed by the IAMs is added to the control packet.

Figure 3 presents an example of the path selection procedure with two paths available between source and target PEs. Each control packet reads the average link usage of each involved IAM, computing the average and maximum rate in the path. After receiving all proposed paths (the number of paths to receive is defined during the setup process - Figure 1), the *slvMonitor* selects the one with the lowest peak and average rate of channels occupancy. In this example, assume the selected path is Route A. After path selection, the *slvMonitor* notifies the source PE the selected path, and monitoring is re-enabled at both sides. After path updating, data packets are transmitted using this new path.

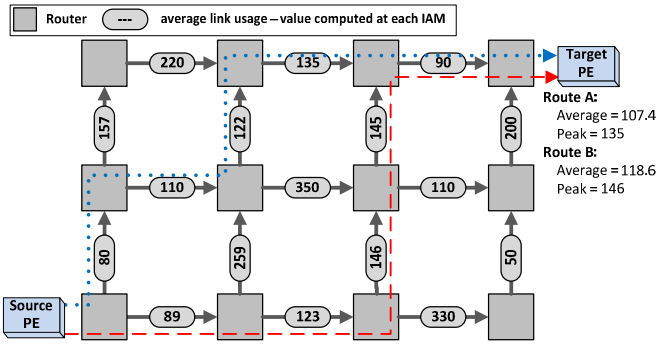

Figure 3 – Path selection example. Path A presents the lowest peak and average rate of channels occupancy.

B. *Path Adaptation Cost*

Figure 4 illustrates an example of path adaptation cost computation, in clock cycles (CC). The values come from a MoNoC RTL simulation, with synthetic traffic. The simulation evaluates the main steps of the adaptation process: (*i*) session opening; (*ii*) path adaptation; and (*iii*) session closing.

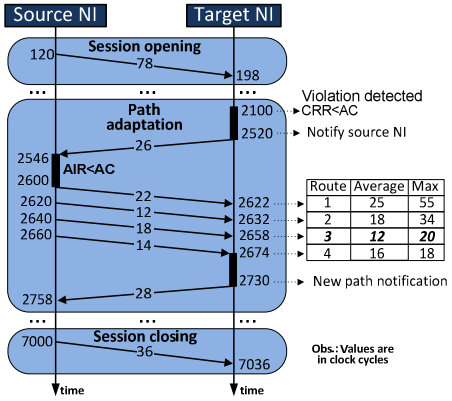

Figure 4 – Example of the path adaptation process computation.

Session opening takes 78 CC, from the moment when the source PE requests to open a session, until when monitoring is enabled at the target NI (Figure 1). The path adaptation phase starts when the target NI detects a reception rate smaller than the agreed rate, CRR<AC. (time: 2,100). The source NI receives the notification (at time 2,546), and verifies that AIR<AC, characterizing a congestion in the path (event 3 in Figure 2). In this example, 4 paths are available for path adaptation. The source NI sends 4 control packets, using four different paths. The target NI receives the fourth control packets at time 2,674, selecting the third path as the one reaching the target NI with smallest contention. The source NI receives the notification of the selected path at time 2,758. From this moment on, all packets are transmitted using the new path. Finally, at time 7,000 the source NI sends a notification message to close the session. The session closing process takes 36 CC.

The source PE cannot communicate with the target PE from the reception of the path adaptation request (at time 2,546) until the path update moment (at time 2,758). This represents only 212 CC, a *remarkable* result only possible due to the use of an all-hardware implementation. From the point of view of PEs, the impact of path adaptation on the performance of a given application is irrelevant; a few hundred CC typically represents the time it takes to execute only a small amount of assembly language instructions at PEs.

IV. RESULTS

A. *Results for Static Traffic Scenarios*

A static traffic scenario contains a set of flows sending packets during the entire simulation. A *disturbing* traffic is a flow that interferes in the path of the *Communicating Pair under Evaluation* (CP_{ev}). Figure 5 illustrates the setup of the experiments presented in this Section, with four disturbing flows applied to the path of the CP_{ev}.

Figure 6 presents the first experiment, executed using the XY routing algorithm, and without path adaptation. The Y-axis graph represents the *application latency* of the CP_{ev} while the Y-axis is the packet number (250 packets were injected by the Source PE of the CP_{ev}). Each curve corresponds to a given injection rate of the disturbing flows according to the channel transmission capacity (i.e. from 10% to 20%; from 20% to 30%; from 30% to 40%; and from 40% to 50%). Even when the disturbing traffic has a small injection rate (from 10% to 20%), congestion leads to application latencies of up to 4,000 CC. For disturbing traffics with higher injection rates, the latency can reach more than 20,000 CC.

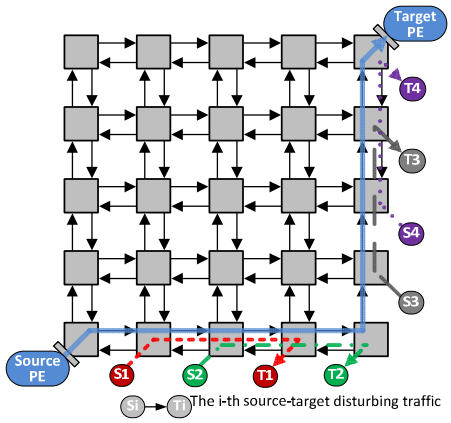

Figure 5 – Experiment with four disturbing flows (S1-T1 to S4-T4) concurring to use resources in the same path as the CP_{ev}.

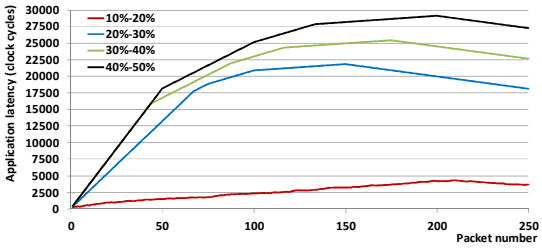

Figure 6 – Application latencies obtained within a static traffic scenario without path adaptation.

The second experiment adopts path adaptation in the CP_{ev}. Figure 7 details the network and application latencies achieved in the first 50 packets for a disturbing traffic with an injection rate from 10% to 20%. The other curves are omitted because they lead to similar results only quantitatively distinct.

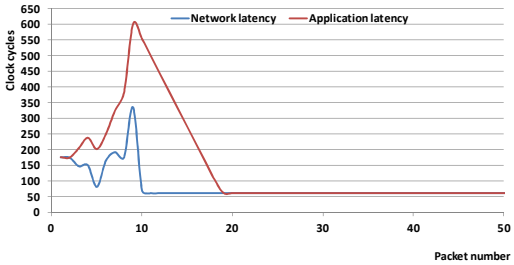

Figure 7 – Application latencies obtained within a static traffic scenario, with path adaptation, for the first 50 packets.

The effect in the latency was expected, since any path not using XY routing by CP_{ev} finds non-congested links. Relevant data to analyze here include *peak latency* and *reaction time.* The *peak latency* defines how long the source PE stays stalled due to the path adaptation process. This process sends packets with higher priority in the network, which causes the latency peak, and induces transient congestion on other data packets.

In accordance with the result of section III.B, the network latency increases less than 300 CC. The impact on the application latency is higher, due to packet buffering at the NI. *Reaction time* defines how long it takes to restore the latency to minimal values. In this experiment, once congestion detection occurs, after less than 10 packets, the network latencies were restored, and after 15 packets, the application latency achieved minimal values. These experiments demonstrate the effectiveness of the path adaptation approach. It is capable of changing the path in a short time, restoring latencies to minimal values.

B. *Results for Dynamic Traffic Scenarios*

These experiments apply the path adaptation when traffic changes at runtime. Figure 8 shows examples of scenarios used in these experiments, which employs a CP_{ev} and several disturbing flows varying: (*i*) the transmission rate of the disturbing traffic, (*ii*) the amount of transmitted packets. Each disturbing flow Si-Ti is active in different moments of the simulation, possibly concurring for resources with the CP_{ev}.

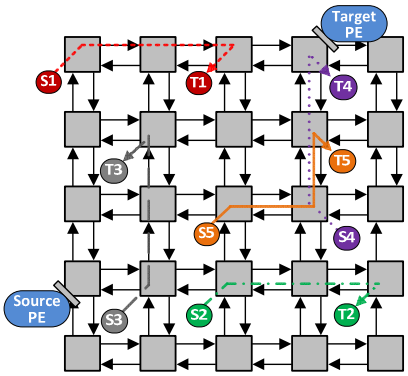

Figure 8 – Experiment setup with dynamic interference traffic examples (Si-Ti) concurring with the CP_{ev} communication.

Figure 9 presents the application latency with and without adaptation, using disturbing traffics with an injection rate ranging from 20% to 30% of the channel bandwidth, and with disturbing traffic changes occurring at each 50 packets sent to the same destination. The disturbing flows all start after the hundredth packet is transmitted by the source PE.

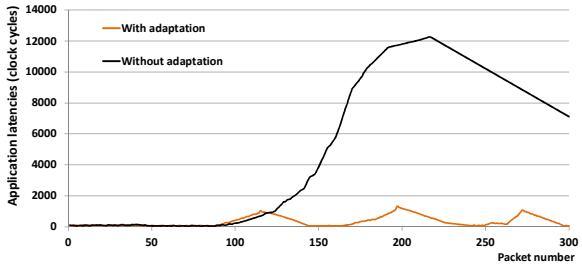

Figure 9 – Application latency with and without adaptation for a dynamic traffic scenario.

Without adaptation, application latency grows beyond 12,000 CC. When applying adaptation, it is possible to observe three peaks in the application latency, corresponding to the moments when path adaptation is performed. The highest application latency observed here was 1,300 CC, i.e. a result around 10 times smaller than that achievable in the non-adaptive scenario. This second experiment demonstrates the benefits of the hardware implementation, which leads to small reaction times that enable restoring small latencies even when disturbing flows appear in several places at once.

V. CONCLUSIONS

This paper presented a NoC mechanism for choosing paths at runtime based on traffic monitoring schemes and protocols. The runtime adaptive routing mechanism is part of MoNoC, a NoC adapted to simultaneously provide distributed traffic monitoring and QoS. MoNoC uses virtual channels to distribute the communication network status, with high priority control packets, but with low communication volume, and these only in very specific moments, which do not compromise bandwidth or NoC latency. At the application level, developers define the required communication rate, and a set of possible paths between each communicating pair. All adaptation is implemented at the NoC network interfaces. Therefore, there is little performance impact at the operating system level as well as at the application level. This approach is alternative to methods that employ software monitoring, leading to smaller reaction times. Experiments exploring static and dynamic traffic scenarios showed that MoNoC is capable to ensure optimal routes as soon as it detects a fail in the contracted rate. Typically, after few packets minimal latency is restored.

ACKNOWLEDGMENT

The authors acknowledge the support granted by CNPQ under processes 472126/2013-0, 302625/2012-7, 310864/2011-9 and 552699/2011-0, by CAPES, under process 708/11, and by FAPERGS under grants 11/1445-0, 12/1777-4 and Docfix SPI 2843-25.51/12-3.

REFERENCES

[1] Ancajas, D.; Chakraborty, K.; Roy, S. **Proactive aging management in heterogeneous NoCs through a criticality-driven routing approach**. In: *DATE*, pp. 1032-1037, 2013.

[2] Matos, D.; Concatto, C.; Kologeski, A.; Carro, L.; Kastensmidt, F.; Susin, A.; Kreutz, M. **Monitor-adapter coupling for NOC performance tuning**. In: *SAMOS*, pp. 193-199, 2010.

[3] Chen, X.; Xu, Z.; Kim, H.; Gratz, P.; Hu, J.; Kishinevsky, M.; Ogras, U. **In-network Monitoring and Control Policy for DVFS of CMP Networks-on-Chip and Last Level Caches**. In: NOCS, pp. 43-50, 2012.

[4] Radetzki, R. **Fault-Tolerant Differential Q Routing in Arbitrary NoC Topologies**. In: EUC, pp. 33-40, 2011.

[5] Huang, Y.; Chou, K.; King, C.; Tseng, S. **NTPT: On the end-to-end traffic prediction in the on-chip networks**. In: DAC, pp. 449-452, 2010.

[6] Yazdi, M.; Modarressi, M.; Sarbazi-Azad, H. **A Load-Balanced Routing Scheme for NoC-Based Systems-on-Chip**. In: *DMEMS*, pp. 72-77, 2010.

[7] Zhao, J.; Madduri, S.; Vadlamani, R.; Burleson, W.; Tessier, R. **A Dedicated Monitoring Infrastructure for Multicore Processors**. *IEEE Transactions on VLSI Systems*, 19(6), pp. 1011-1022, 2011.

[8] Rahmani, A.; Vaddina, K.; Latif K.; Liljeberg, P.; Plosila, J.; Tenhunen, H. **High-Performance and Fault-Tolerant 3D NoC-Bus Hybrid Architecture Using ARB-NET Based Adaptive Monitoring Platform**. *IEEE Transactions on Computers*, PrePrints, 14p., 2013.

[9] Garbade, A.; Weis, S.; Schlingmann, S.; Fechner, B. **Impact of Message Based Fault Detectors on Applications Messages in a Network on Chip**. In: *Euromicro*, pp. 470-477, 2013.

[10] Mak, T.; Lam, K.P.; Cheung, P.; Luk, W. **Adaptive Routing in Network-on-Chips Using a Dynamic Programming Network**. *IEEE Transactions on Industrial Electronics*, 58(8), pp. 3701-3716, 2011.

[11] Al Faruque, M. A.; Ebi, T.; Henkel, J. **AdNoC: Runtime Adaptive Network-on-Chip Architecture**. *IEEE Transactions on VLSI Systems*, 20(2), pp. 257-269, 2012.

[12] Hu, S.; Yat-sen S.; Lin, X. **A Symmetric Odd-Even Routing Model in Network-on-Chip**. In: ICIS, 2012, pp. 457-462.