A Monitoring and Adaptive Routing Mechanism for QoS Traffic on Mesh NoC Architectures

Leonel Tedesco
FACIN-PUCRS
Av. Ipiranga, 6681 - 90619-900
Porto Alegre – BRASIL
leonel.tedesco@pucrs.br

Fabien Clermidy
CEA-LETI-MINATEC
38054 Cedex 9
Grenoble - FRANCE
fabien.clermidy@cea.fr

Fernando Moraes
FACIN-PUCRS
Av. Ipiranga, 6681 - 90619-900
Porto Alegre – BRASIL
fernando.moraes@pucrs.br

ABSTRACT
The development of MPSoCs targeting embedded systems with a dynamic workload of applications constitutes an important challenge. The growing number of applications running on these systems produces a considerable utilization of resources, implying a high demand of computation and communication in the different MPSoC parts. The heterogeneity of processing elements brings to the application traffic a dynamic and unpredictable nature, due to the variability on data injection rates. NoCs are the communication infrastructure to be used in such systems, due to its performance, reliability and scalability. Different strategies may be employed to deal with traffic congestion, such as adaptive routing, buffer sizing, and even task migration. The goal of this work is to investigate the use of adaptive routing algorithms, where the path between source and target PEs may be modified due to congestion events. The major part of the state of art proposals have a limited view of NoCs, since each NoC router takes decisions based on few neighbors' congestion status. Such local decision may lead packets to other congested regions, therefore being inefficient. This paper presents a new method, where congestion analysis considers information of all routers in the source-target path. This method relies on a protocol for QoS session establishment, followed by distributed monitoring and re-route to non-congested regions. The set of experiments present results concerning performance and amount of time spent by packets on routers when the proposed method is applied.

Categories and Subject Descriptors
B.7.1 [Integrated Circuits]: Types and Design Styles – advanced technologies, algorithms implemented in hardware, VLSI (very large scale integration).

General Terms
Design, Experimentation, Measurement, Performance, Theory, Verification.

Keywords
Networks on Chip, Traffic Monitoring, Dynamic Routing, QoS.

CODES+ISSS'09, October 11–16, 2009, Grenoble, France.

1. INTRODUCTION
Present embedded applications can be characterized by high levels of complexity and size, beyond strict performance constraints. It is also observed technological advances in terms of the number of transistors that it is possible to integrate on a single chip, which can be reach 100 billion [1]. This fact enables the growing on the capacity of MPSoCs architectures, which are adequate to treat the intense computational demand of the current embedded systems.

Application traffic that occurs on typical MPSoCs environments includes data transfers between processors, memories, specific units, being characterized in terms of modeling by multimedia streams, control signals, and data blocks transfers [2]. This traffic also has flows with different QoS levels requirements, which is usually specified in terms of throughput, latency, and deadlines.

It is possible to increase the overall throughput on MPSoCs using Networks-On-Chip (NoC) [3]. Scalability, reusability, parallelism, and fault tolerance are characteristics that justify its use, replacing traditional forms of on-chip interconnect, as busses or point-to-point architectures. The design complexity, application performance requirements, and silicon area constraints constitute some challenges to adapt MPSoC applications on NoC architectures.

NoCs are generally built targeting a specific application, where data are generated at periodic time intervals and traffic behavior can be previewed. On the other side, the amount of computation and communication in emerging MPSoCs incurs in systems with a considerable variability and unpredictability on the overall communication throughput, bringing a dynamic nature to these. Changes on systems constraints and user operation are events that may appear at MPSoCs runtime. Fault tolerance aspects must also be considered [4].

Some approaches on the NoC literature target the treatment of unpredictable events and performance increasing at run time. They have in common the use of monitoring units, which inform some entity that abnormality in the network traffic has occurred. Monitors may be use to control the injection rate, as proposed in [5] and [6]. The attribution of dynamic priority for QoS packets based on router delivering rates is adopted by [7].

The goal of this work is the use of adaptive routing algorithms to treat network traffic at runtime. Based on some condition on network traffic, NoC routers are configured to change their routing decisions. The major part of state of the art proposals considers local traffic monitoring, that is, each router analyses the condition of its immediate neighbors to perform routing. The disadvantage of these approaches is the absence of a global view

of the current traffic, once a routing decision for a given flow may lead its traffic to other congested region. The present work proposes an algorithm that uses information collected for a given path to perform traffic routing. Monitoring data are transmitted on available bits present on packets. Notifications of network conditions inform source traffic units that it is necessary the modification on path to the target router.

The remaining of the paper is organized as follows. Section 2 presents the state of art on adaptive routing algorithms. Section 3 presents the protocol adopted for session establishment and Section 4 focus on the proposed dynamic routing algorithm. Section 5 presents the experimental setup and results. Section 6 summarizes the contributions of this work.

2. RELATED WORK

Li et al. [8] present the architecture and modeling of the DyXY router. This router provides adaptive routing and assures deadlock-free and livelock-free guarantees, once each packet is transmitted along the shortest path between a given pair of source-target nodes. For each packet to be routed, is verified the alignment on X or Y axis with the target router. If there is no alignment, the packet is adaptatively routed according to stress values, which represent the congestion condition of the router. Buffer fillings on the neighboring routers configure stress values. A history buffer is used to help the input arbiter to select input requests for routing.

Hu et al. [9] present the Dyad routing algorithm, which combines the advantages of deterministic and minimal adaptive routing algorithms. Routing decisions are based on congestion values, which are represented by the input buffers occupation. Congestion flags information are exchanged between neighbor routers. If the network is not congested, the DyAD router works on deterministic mode, else, the adaptive mode is activated.

Lofti-Kamran et al. [10] elaborate the BARP routing algorithm. Each router on the NoC considers its state of congestion, where two levels of threshold are monitored. Threshold levels correspond to the number of packets on the input buffer. According to the threshold reached, packets are sent to critical groups of routers, which have to change their packets output port, in order to avoid congested regions. It is a different approach compared to DyAD and DyXY, once the traffic is monitored in routers that are different to those which modifies their routing processing. Deadlock avoidance is guaranteed with the use of different virtual channels for each traffic direction. Many monitoring messages to groups of routers may be generated, which increases the overall traffic load on the NoC. In addition, there is a need at some level to control packets ordering at the target routers.

Al Faruque et al. [11] propose an adaptive on chip communication scheme, which dynamically allocates paths from a given flow to meet QoS requirements. In the presented method, the intermediate routers makes decisions in a local way depending on the available bandwidth in each direction to the neighboring routers and on the distance between current and target routers. This availability is evaluated according to an algorithm called wXY, which is based on the weight of a path between a given current and destination node. The obtained results presented increasing on individual buffers utilization and decreasing on the total number of buffers allocated.

Gratz et al. [12] propose the Routing Congestion Awareness Algorithm technique (RCA), which informs congestion monitoring values in parts of the network beyond adjacent routers, bringing a global view of congestion. Congestion information is propagated across a specific network, separated to the one used for application data transmission. The number of active requesters for output ports was chosen as the preferential metric of congestion. Each router analyses the congestion information received to define which is the output port for its current flow. The efficiency of the approach is indicated by reduction on latency and increase on throughput compared to local congestion strategies for routing.

The analysis of congestion on neighbors' routers presents a limited view of the overall network traffic condition by each router. This approach is suitable for traffic workloads with a high degree of locality, where nodes communicate with those that are closer to themselves. In traffic patterns with lower locality, the local monitoring may lead a given to traffic to other congested areas. This work explores the use of an adaptive routing algorithm based on monitoring information on traffic paths.

3. PROTOCOL FOR MONITORING PROCESS

This Section presents the protocol used to transmit monitoring data information and notification of congestion occurrences. Two classes of packets are defined. DATA packets carry application data and traffic monitoring information (direction: source→ target). ALARM packets carry congestion information, used to change the path of the DATA packets (direction: target→source).

This work adopts *source routing*. Figure 1 presents the packet structure. The header flit of DATA packets carries the path to the target, and *monitoring field*s. The path to the target router is specified in the *path to the target* field, which assumes for each router the values {N,S,E,W}. When the current direction is the opposite of the last direction taken, it means that the local port is reached. The *monitoring fields* contain: (i) `ident`, router address responsible to collect the congestion status; (ii) `ocup`, congestion status of the router specified in the field `ident`.

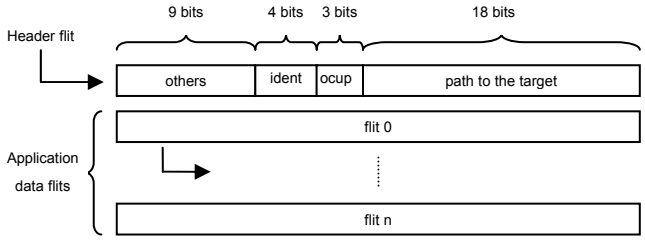


Figure 1 – DATA packet structure.

ALARM packets, presented in Figure 2, are back-propagated from the target to the source, containing one flit. The 18 less significant bits are used to indicate the path to target, as in the DATA packets. The field *congestion information* carries the path congestion status, where each bit indicates if a hop on the path is congested or not (limiting the maximum number of hops between source to target to 10). This packet is then used in the source PE

to define a new path to the target. The computation of the new path to the target is explained in detail in Section 4.

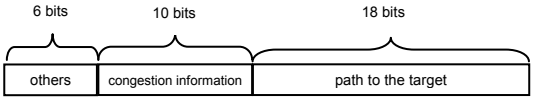


Figure 2 – ALARM packet structure.

It is important to note that `others` fields may be used to store information about begin/end of packets or, begin/end of messages, for example, or even packets priority. This type of information is normally used by NoC structures to execute flow control schemes.

The protocol for congestion registering works as follows. The traffic initiator opens a monitoring session with a first DATA packet. This packet configures each hop with its unique identification on the path. At the target side, a **TCT** (*Target Congestion Table*) is initialized with size equal to the number of hops of the flow being initialized. This table will store traffic statistics for each hop of the flow. The following DATA packets are sent with the `ident` field configured with the router address responsible to store its congestion level information.

Consider the source-target traffic pair 1-7 in Figure 3, starting the flow 'X'. The first DATA packet assigns to the `ident` field the value '1'. At each hop in the path, this value is incremented, and stored in the field identification of the Congestion Table. In the example, the Congestion Table of router 2, is initialized with flow 'X', when the DATA packet with `ident`=2 arrives at router 2.

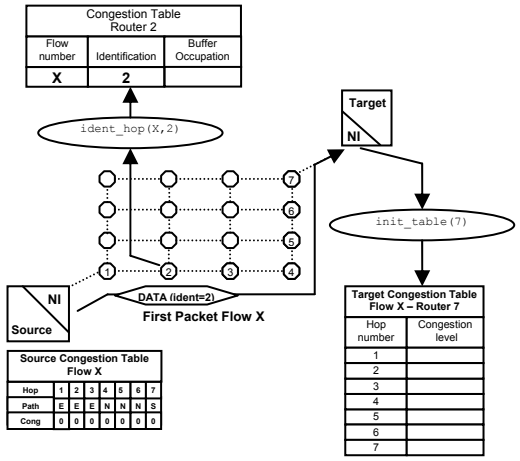


Figure 3 – Monitoring Session Establishment.

After path initialization, when an intermediate router receives a DATA packet, it verifies the `ident` field, and if its matches with the router address, `ocup` information must be filled. This field is filled with the value of the parameter adopted for QoS evaluation. This parameter can be, for example, the amount of used buffer slots, the output data rate or the average time spent by packets to traverse the router. If there is no match, the router updates its congestion table.

The example of Figure 4 illustrates a request to capture the mean occupation of the congestion state of router 3 (`ident`=3). The mean buffer occupation (BO) of the input port receiving flow 'X' is 2, value forwarded to the target router by means of the `ocup` packet field. When the DATA packet arrives at the target router, the content of the `ocup` field is inserted in the TCT. The TCT, sized according to the number of hops in the path, is addressed by the `ident` field. As shown in Figure 4, the table has 7 entries, since there are 7 hops in the path of flow 'X'. Note that the third entry on the table is updated. The target router, using the information stored at the TCT, has a global knowledge of the path being used by DATA packets. Such global knowledge enables the implementation of new routing heuristics. This is the main difference of our proposal to the major part of state of the art heuristics, which considers only neighbours routers.

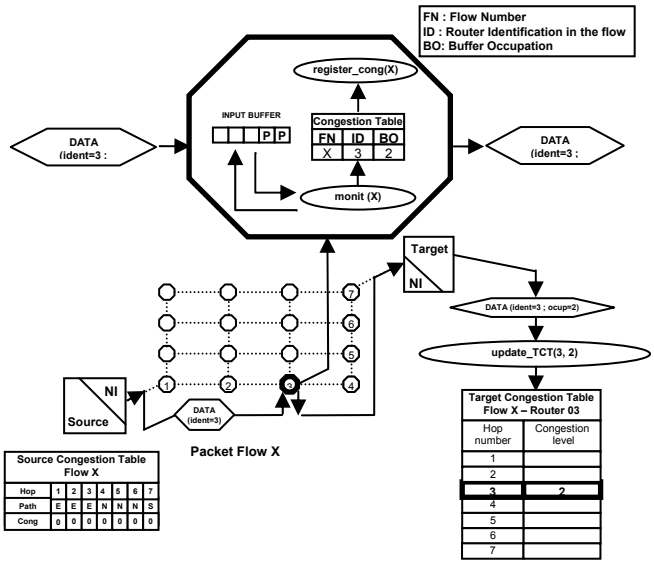


Figure 4 – Registering monitoring values.

4. CONGESTION DETECTION AND NEW PATH COMPUTATION

Periodically, the NI compares the TCT congestion levels to a given threshold, which is previously defined according to the QoS requirements of the application that generates the packet flow. If one or more values are greater than the threshold value, an ALARM packet containing the address of the congested routers is back-propagated to the source router. When the source router receives the ALARM packet, a new path is computed and used for the next DATA packets.

As an example, consider the target NI in Figure 5. It is possible to observe that a congestion level equals to 5 is reached at routers 3 and 5. At this moment, a function `gen_alarm` is invoked. This function configures a new ALARM packet, which will indicate the 2 congested routers.

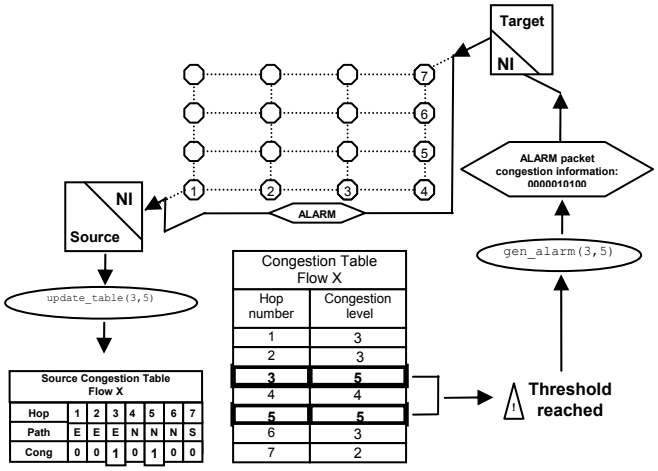


Figure 5 – Congestion Detection.

This ALARM packet is detailed in Figure 6. As routers 3 and 5 are congested, bits 3 and 5 of the *congestion information* field are asserted. The source NI receive this packet and update the `cong` field of *Source Congestion Table*. Note in Figure 6 values '1' in hops 3 and 5.

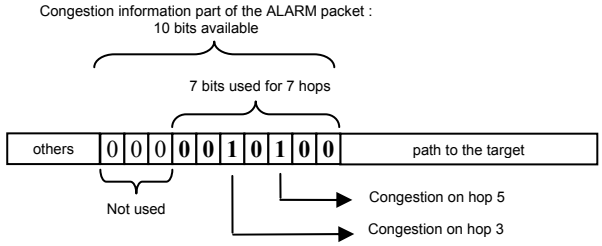


Figure 6 – An ALARM packet example.

The definition of the new path follows two assumptions: (*i*) minimal routing, and therefore all paths have the same number of hops to the target; (*ii*) the new path should use, if it is possible, routers near to the oldest path, to minimize the impact of the new flow in other existing flows.

The proposed heuristic to compute the new path to the target uses two vectors to store the congestion states. The X and Y direction vectors store the congestion state for each NoC column and line, respectively. Figure 7 shows the relationship between these vectors in a 5x5 mesh.

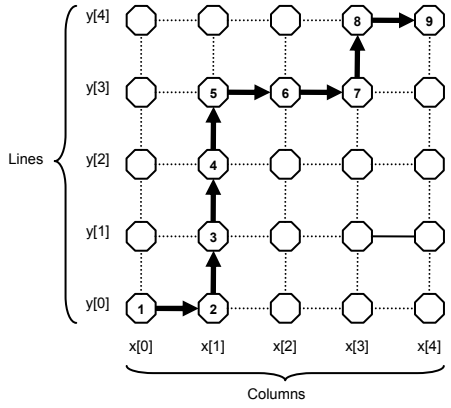


Figure 7 – Example of a path in a 5x5 mesh.

Table 1 presents an example of source congestion table, with hops '3', '6', 7' and '8' congested.

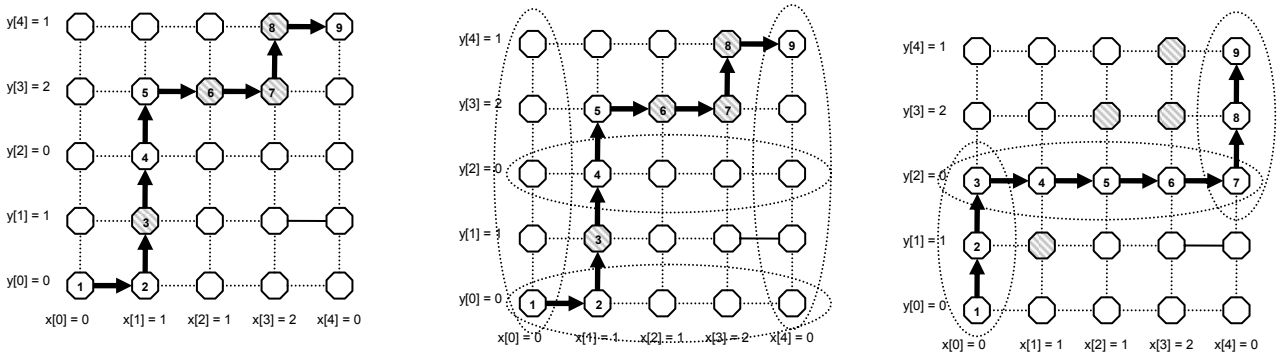
Table 1 – An example of Source Congestion Table.

Source Congestion Table Router 00									
Hop	1	2	**3**	4	5	**6**	**7**	**8**	9
Path	E	N	**N**	N	E	**E**	**N**	**E**	**W**
Cong	0	0	**1**	0	0	**1**	**1**	**1**	0

Figure 8(a) presents the congested routers shadowed and Figure 8 (b) presents the lines and columns that are congestion free. Vectors X and Y are filled by counting the number of congested routers at each column and line, respectively. The resulting vectors of the example presented in Figure 8 (a) are:

$$X = [0 ; 1 ; 1 ; 2 ; 0]$$

$$Y = [0 ; 1 ; 0 ; 2 ; 1]$$



(a) (b) (c)

Figure 8 - (a) Congestion points highlighted; (b) Congestion free lines and columns highlighted; (c) New path to the target computed.

Figure 9 presents a pseudo-algorithm which describes the method used to define the new path to target. The algorithm receives as arguments three parameters, as stated in line 1: (*i*) `path`, which contains the original path to the target; (*ii*) `nhops`, the number of hops between source and target of traffic; (*iii*) `congestion`, a vector which describes the congestion state of the path.

```
1: compute_new_path(path,nhops,congestion){
2:  //searching for horizontal and vertical directions
3:  for (i=0 ; i<nhops ; i=i+1){
4:   if ((path[i]==EAST) OR (path[i]==WEST)){
5:     x_direction <- path[i];
6:     size_dirx    <- size_dirx+1;
7:   }
8:   else if ((path[i]==NORTH) OR (path[i]==SOUTH)){
9:     y_direction <- path[i];
10:     size_diry    <- size_diry+1;
11:   }
12:   //configuration of vectors vet_x and vet_y
13:   if (congestion[i]!=0){
14:     x[size_dirx] <- x[size_dirx]+1;
15:     y[size_diry] <- y[size_diry]+1;
16:   }
17:  }
18:
19:  //the biggest distance is horizontal or vertical?
20:  if (size_dirx > size_diry)
21:    biggest <- size_dirx;
22:  else
23:    biggest <- size_diry;
24:
25:  for(i=0 ; i<biggest ; i=i+1){
26:    // first positions of x and y vectors
27:    if (i==0){
28:      //first hop congested->change first direction
29:      if ((x[i]!=0) AND (y[i]!=0)){
30:        if (path[i] == x_direction)
31:          add_new_path (y_direction);
32:        else
33:          add_new_path (x_direction);
34:        //search will begin by x or y?
35:        for (j=1 ; first_found == false ; j=j+1){
36:         if ((x[j]==0) AND (y[j]!=0)){
37:          first <- y;
38:          first_found <- true;
39:         }
40:         else if ((x[j]!=0) AND (y[j]==0)){
41:          first <- x;
42:          first_found <- true;
43:         }
44:        }
45:      }
46:      else if ((x[i]==0) AND (y[i]!=0)){
47:        for (j=1 ; first_found == false ; j=j+1){
48:         if ((x[j]==0) AND (y[j]!=0)){
49:          first <- y;
50:          first_found <- true;
51:         }
52:         else if ((x[j]!=0) AND (y[j]==0)){
53:          first <- x;
54:          first_found <- true;
55:         }
55:        }
56:      }
57:      else if ((x[i]==0) AND (y[i]!=0)){
58:        first <- y;
59:      }
60:      else if ((x[i]!=0) AND (y[i]==0)){
61:        first <- x;
62:      }
63:    }
64:    else
65:      build_remaining(i, first);
66:  }
67:}
```

Figure 9 – Pseudo-algorithm for the new path definition.

The loop specified between lines 3 and 17 computes the `x_direction` and `y_direction` variables, which are the horizontal and vertical directions of the path. Once minimal routing is used, these directions are the ones which will constitute the new path to target. The `size_dirx` and `size_diry` variables define the distances in x and y coordinates, respectively, to the target router. The lines 14 and 15 present the computation of each cell of the vectors X and Y.

The proposed heuristic to define the new path to target seeks in an *alternatively* way in both X and Y vectors, the first index equal to '0'. The loop that executes this search is the one between lines 25 and 66. Lines 19-23 defines the value of the `biggest` variable, which controls the execution of the loop, being the size of the largest between X and Y vectors. Between the lines 27 and 63 is executed the code to compute the value for the variable `first`, which denotes the first vector to be analyzed. To obtain the value of `first`, the first cell of X and Y vectors is verified.

If `x[0]` and `y[0]` are different to zero (lines 29-45), it means that the first hop of the routing path is already a congested one. In this case, the new first hop direction is other than the original one. That is, if the old direction is in *x* direction, the new one will be on the *y* direction, and vice-versa. The definition of the variable `first` will be decided by the loop between lines 35 and 44, which analyzes the remaining cells of both vectors. If the first zero is found in X vector, the y direction will the first to be adopted. This same loop is executed if `x[0]` and `y[0]` are equal to zero (lines 46-56). Lines 57-62 consider the remaining conditions, that is, when the first cells of X and Y stores different values between them.

Once the variable `first` is defined, it is time to seek the remaining part of X an Y vectors. This is done by the function `build_remaining`, and it is shown in Figure 10. This function receives as parameters the current cell being searched and the value of `first`.

When the first '0' is found (that may occurs on lines 4, 11, 20 or 27), the source router is connected to router index having this value. If the first '0' is found in the X vector, a horizontal connection is implemented; otherwise a vertical connection is created when this '0' is found in the Y vector. The horizontal or vertical connections are done by the function `add_new_path`, which concatenates the routing direction to the variable `new_path`, that stores the new routing path for the flow. The next step is to repeat this process for the next '0' value (which may belong to a X or Y direction vector), creating a new partial path between the previous router to present one.

At the end of the process, if the target router is not reached, the path must be completed until it. Note that the target router does not present congestion, since it is receiving the flow. In this way, that partial path always reach the line or column of the target router, being the complete path obtained with a vertical or a horizontal segment.

```
1: build_remaining(i, first){
2:  if (first==x){
3:    if (i<num_dirx){
4:      if (vet_x[i]==0){
5:         for (j<-x_actual ; j<i ; j<-j+1)
6:           add_new_path(new_path,x_direction)
7:         x_actual<-i;
8:      }
9:    }
10:   if (i<num_diry){
11:     if (vet_y[i]==0){
12:        for (j<-y_actual ; j<i ; j<-j+1)
13:          add_new_path(new_path,y_direction)
14:        y_actual<-i;
15:     }
16:   }
17: }
18: else{
19:   if (i<num_diry){
20:     if (vet_y[i]==0){
21:        for (j<-y_actual ; j<i ; j<-j+1)
22:          add_new_path(new_path,y_direction)
23:        y_actual<-i;
24:     }
25:   }
26:    if (i<num_dirx){
27:      if (vet_x[i]==0){
28:        for (j<-x_actual ; j<i ; j<-j+1)
29:          add_new_path(new_path,x_direction)
30:        x_actual<-i;
31:     }
32:   }
33:  }
34:}
```

Figure 10 – Pseudo-algorithm to seek the remaining part of X and Y vectors.

Consider the vectors X and Y of the example illustrated in Figure 7. As shown in Figure 11, the first '0' is found at y[2]. Thus, it will be added on the new path 2 routings on the North direction. The next '0' found is in the position x[4]. In this way, it will be added 4 East directions on the path. The Figure shows that the target is not already reached. Therefore, the path will be completed with 2 routings on the North direction and 1 on the Local direction. This is coherent with the congestion-free path principle, once these two routing on North direction are made on a column without congestion points. Figure 8(c) illustrates the new path to the target.

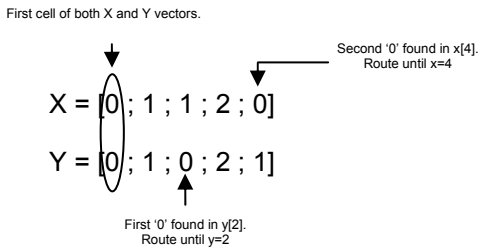


Figure 11 - Illustration of new path computation taking as reference congestion vectors X and Y.

To prevent deadlock, the turns in the path obey the turns allowed in partial-adaptive routing algorithms, as west-first. To be completely adaptive, 2 virtual channels can be used, being the first one using west-first routing, and the second one a symmetric one, as east-first.

Before the adoption of the new path, a last DATA packet is sent in order to clean the congestion tables belonging to each intermediate router of the old path. The new path is thus initialized with a DATA packet to open a new routing session, identifying the routers of this path. Considering that minimal routing is adopted, the TCT on the target remains with the same size.

5. EXPERIMENTAL SETUP

The set of experiments use the asynchronous NoC platform presented in [13]. Implementation of the methods presented was made in SystemC TLM. This communication structure has the main following main features: (*i*) source routing; (*ii*) wormhole packet switching; (*iii*) 32-bit fixed size bits; (*iv*) end-to-end credit based flow control. Each packet is composed of a header flit and, depending on the packet type, a number of flits, which carry application and configuration data. The clock frequency used for simulation is 500MHz.

5.1 TRAFFIC MODELING

In this work, one spatial traffic model was defined to evaluate the proposed routing method. The objective is to generate congestion areas during simulation, and verify the adaptivity of the routing method.

Three types of traffic generators and receptors are defined, as shown in Table 2 (for generators) and Table 3 (for receptors). *QoS traffic flows* are those which have performance requirements. Each unit that generates QoS traffic executes the method of dynamic routing presented in Section 4. QoS traffic receptors evaluate if it is necessary the sending of ALARM packets, in order to notice the generators about the congested areas. *Disturbing traffic flows* are those which produce hot-spot areas, and are used to cause perturbation on the QoS flows. The PEs which generates and receives disturbing flows do not execute the method for dynamic routing presented. *Noise traffic flows* are those used as background traffic, and do not make considerable perturbations on the other flows.

Table 2 – Types of generated flows.

Label	Node description
Q_{Gfn}	QoS traffic generator of the flow number *fn*.
D_{Gfn}	Disturbing traffic generator of the flow number *fn*.
N_{Gfn}	Noise traffic generator of the flow number *fn*.

Table 3 – Types of received flows.

Label	Node description
Q_{Rfn}	QoS traffic receptor of the flow number *fn*.
D_{Rfn}	Disturbing traffic receptor of the flow number *fn*.
N_{Rfn}	Noise traffic receptor of the flow number *fn*.

Figure 12 illustrates the spatial traffic distribution for the simulated flows for the set. The **CPU** activates each PE to start data generation/reception. Nodes labeled as **R** indicates a router with no generator/receptor PE attached. Considering that data routing follows initially the **XY** algorithm, it is possible to observe the hot spot areas, where the higher number of flows compete for resources, being potential congestion points. Therefore, QoS flows have to avoid this area in case of congestion. In the Figure, the hot-spot area is highlighted.

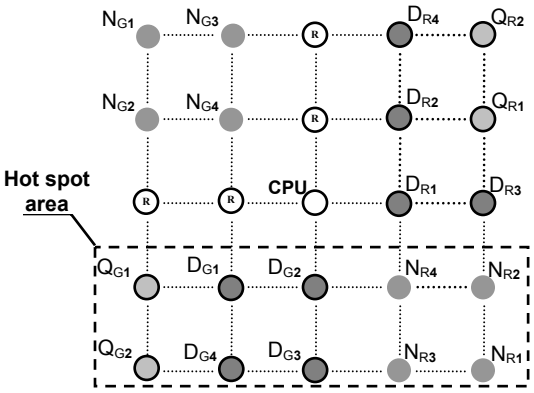


Figure 12 – Spatial traffic scenario adopted.

Each **QoS** traffic generator sends **8000** flits. The average injection rate for this traffic is **1.2 Gbps**. **Noise** traffic generators inject each one 2000 flits, at a rate which varies between **1.5 Gbps and 2.0 Gbps**. **Disturbing** traffic flows are generated at rates which vary between **1 Gbps and 1.4 Gbps**. Three scenarios are evaluated according to th number of flits generated by disturbing flows, being described in Table 4. The following parameters that configure disturbing flows are:

- **NF**: Number of generated flits;
- **LP**: Percentage of simulation time spent when the last packet of the flow is transmitted, i.e., the end of the disturbing flow w.r.t the QoS flow.

Table 4 – Disturbing traffic flows configuration.

Initiator	Feature	Scenario 1	Scenario 2	Scenario 3
D_{G1}	NF	2250	4250	5750
	LP	45%	65 %	85 %
D_{G2}	NF	2250	4250	5750
	LP	45 %	65 %	85 %
D_{G3}	NF	2250	4250	5750
	LP	30 %	45 %	60 %
D_{G2}	NF	2250	4250	5750
	LP	40 %	60 %	75 %

5.2 QoS MONITORING PARAMETERS

This section presents the parameters taken as reference for QoS monitoring for each QoS flow. Each intermediate router monitors each flit for each routing direction, registering three parameters: the moment when the flit enters the router, the flit source, and the moment when the flit leaves the router. With these values, the amount of time the flit spent inside the router is computed.

At regular intervals, the router updates the statistics of the QoS flows. When the router is requested to register the congestion status, it fills the *ocup* field (Figure 4) with the average amount of time spent by the QoS flit flows in the router.

At the target side, each QoS receptor periodically monitors its TCT (Target Congestion Table) in order to detect congestion on the path (as shown in Figure 5). In the conducted experiments, a flow with an average flit time on a router above **4 ns** is considered a congested point. Therefore, this is the requirement for the QoS flows. This value is two times the optimal value for flit times in routers. This optimal value of 2 ns is obtained in a previous simulation, where QoS flows are not disturbed by concurrent flows

5.3 LATENCY EVALUATION

Latency is obtained for each flit, being the time spent between its transmission and reception at the target node. This analysis considered the latency only for the QoS flows, generated by Q_{G1} and Q_{G2} (Figure 12). Table 5 presents the average latency values obtained and their standard deviation, in ns, for the defined scenarios.

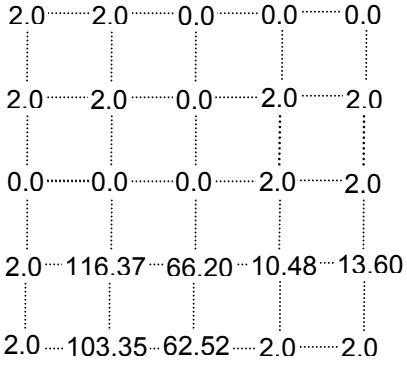
With the above results, it is possible to observe that both the QoS flows have reduced their average latency in the three scenarios experienced. The reduction on the average latency is similar for the two QoS flows. However, the standard deviation on latency performed by Q_{G1} presents the higher reduction. This is justified by the possibilities out of the hot spot area that can be explored to route this flow, where a lower degree of concurrence is found.

Table 5 – Latency values for QoS flows.

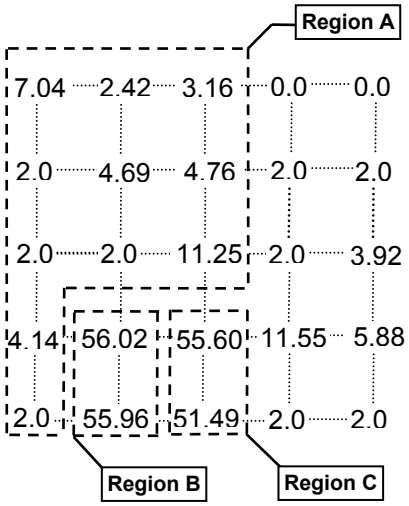
Latency (AV: average; SD: standard deviation) in ns			
Flow	**Without the proposed method**	**With the proposed method**	**Reduction (-) or Addiction (+)?**
Scenario 1			
$Q_{G1} \rightarrow Q_{R1}$	AV: 25.80	AV: 22.31	AV: - 13.53%
(7 hops)	SD: 39.31	SD: 25.62	SD: **- 34.83%**
$Q_{G2} \rightarrow Q_{R2}$	AV: 31.12	AV: 26.87	AV: - 13.66%
(9 hops)	SD: 39.93	SD: 28.77	SD: - 27.95%
Scenario 2			
$Q_{G1} \rightarrow Q_{R1}$	AV: 29.07	AV: 20.40	AV: - 29.82%
(7 hops)	SD: 49.75	SD: 29.04	SD: **- 41.63%**
$Q_{G2} \rightarrow Q_{R2}$	AV: 35.66	AV: 26.01	AV: - 27.06%
(9 hops)	SD: 50.52	SD: 35.91	SD: - 28.92%
Scenario 3			
$Q_{G1} \rightarrow Q_{R1}$	AV: 32.34	AV: 22.40	AV: - 30.74%
(7 hops)	SD: 58.16	SD: 32.76	SD: **- 43.67%**
$Q_{G2} \rightarrow Q_{R2}$	AV: 40.19	AV: 28.91	AV: - 28.07%
(9 hops)	SD: 58.89	SD: 42.73	SD: - 27.44%

5.4 FLIT SPENT TIME

The second metric evaluated is the time spent by flits in routers (named here *flit spent time*). For this, three traffic regions were defined: A, B and C. Each region limits a group of routers with similar values of average flit spent times. These regions are illustrated in Figure 13(b). The presented values (in *ns*) are those which were obtained for the simulation of Scenario 2.



(a)



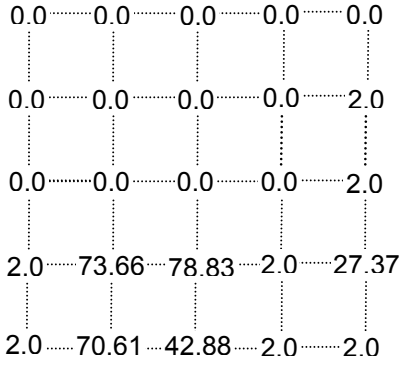
(b)

Figure 13 – Average flit times in routers for each NoC region, for all flows. In (a), the values when the proposed method for dynamic routing is not adopted. In (b) is presented the values when the method proposed is adopted.

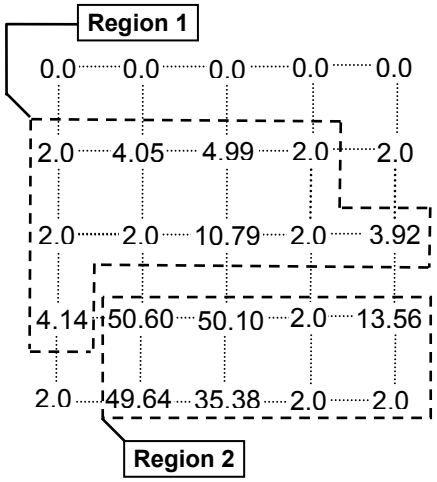
Table 6 presents the average values of flit times for the regions presented in Figure 13. The region B presents the major reduction compared to the other ones. It is expected, considering that the changes in the routing path involve mainly this region, which is closed to Q_{G1} and Q_{G2}. The routers which belong to region A offer alternative paths to a number of packets that would pass through congested regions, caused by disturbing traffic. If the method is applied, there occurs a major exploration of this region, which was not previously used by the flow generated by Q_{G1} and Q_{G2}, as shown in Figure 13(a). This fact justifies the increasing on the values of flit times in this region.

Table 6 – Flit times in routers for the specified regions.

Average flit times in routers for Scenario 2 (ns)			
Region	**Without the proposed method**	**With the proposed method**	**Reduction (-) or Addiction (+)?**
A	2.00	4.13	+ 106.50%
B	109.89	55.99	- 49.04%
C	64.36	53.55	- 16.80%



(a)



(b)

Figure 14 – Flit times in routers for the QoS flows. Values are presented in (a) when dynamic routing is not adopted and (b) when the dynamic routing method proposed is adopted. Regions 1 and 2 are highlighted.

Specifically for the QoS flows, it is even possible to specify regions according to their flit times in routers (Figure 14(b)). Note that the occupation of region 1 occurs when the dynamic routing method proposed is adopted. Table 7 presents the average values of flit times in regions 1 and 2. It is shown that it leads to an occupation increasing of 89.50% in this region, compared to the values obtained when only the XY routing is executed. The region 2 has its utilization reduced in 31.88%.

Table 7 – Flit times in regions used by QoS flows (ns).

Region	**Without the proposed method**	**With the proposed method**	**Reduction (-) or Addiction (+)?**
1	2.00	3.79	+ 89.50%
2	37.67	25.66	- 31.88%

6. CONCLUSIONS AND FUTURE WORK

The original contribution of this research work is a new method for adaptive dynamic routing to be used in networks on chip. This

method takes routing decisions based on the congestion path of each QoS flow, bringing to the routing algorithm a global view of the path being routed, not only neighbors routers status, as the state of the art proposals. Results present reduction on the flit latency average and standard deviation, important parameter for QoS flows, and smaller flit times on hot-spot routers. The major reductions occur with the flow with lower degree of concurrence on the alternative paths.

The proposed approach will be compared to other approaches in a near future. Other future works include: (1) analysis of the benefits when some routers of a path are selected to perform monitoring, instead of all routers, strategy that it is adopted today; (2) extension of the proposed routing algorithm, considering the weight of congestion (in the present implementation, a new path is considered free if its lines and columns have cong values equal to zero); (3) include new metrics for congestion beyond packet times in routers, e.g., throughput; (4) perform experiments using other traffic patterns and real traffic scenarios.

7. ACKNOWLEDGMENTS

This research was supported partially by CNPq (Brazilian Research Agency), projects 300774/2006-0 (PQ) and 471134/2007-4 (Universal).

8. REFERENCES

[1] Borkar, S. “Thousand Core Chips: a Technology Perspective”. In: DAC’07, pp. 746-749. *DOI= http://doi.acm.org/10.1145/1278480.1278667*.

[2] Tedesco, L. et al. "Application Driven Traffic Modeling for NoCs". In: SBCCI’06, pp. 62-67. *DOI= http://doi.acm.org/10.1145/1150343.1150364*.

[3] Benini, L.; De Micheli, G. “Networks on Chips: a New SoC Paradigm”. IEEE Computer, v.35(1), 2002, pp. 70-78. *DOI= http://doi.ieeecomputersociety.org/10.1109/2.976921*.

[4] Al Faruque, M.A. et al. “ROAdNoC: Runtime Observability for an Adaptive Network on Chip Architecture”. In: ICCAD’08, pp. 543-548. *DOI= http://doi.acm.org/10.1145/1509456.1509577*.

[5] Ogras, U. Y.; Marculescu, R. “Analysis and Optimization of Prediction-Based Flow Control in Networks-on-Chip”. ACM Transaction on Design Automation of Electronic Systems, v.13(1), 2008, article 11, 28p. *DOI= http://doi.acm.org/10.1145/1297666.1297677*.

[6] Manolache, S. et al. “Buffer Space Optimisation with Communication Synthesis and Traffic Shaping for NoCs”. In: DATE’06, pp. 1-6.

[7] Mello, A. et al. “Rate-based Scheduling Policy for QoS Flows in Networks on Chip”. In: VLSI-SOC 2007, pp. 140-145.

[8] Li, M. et al. “DyXY - A Proximity Congestion-aware Deadlock-free Dynamic Routing Method for Network on Chip”. In: DAC’06, pp. 849-852. *DOI= http://doi.acm.org/10.1145/1146909.1147125*.

[9] Hu, J.; Marculescu, R. “Dyad – Smart routing for networks on chip”. In: DAC’04, pp. 260-263. *DOI= http://doi.acm.org/10.1145/996566.996638*.

[10] Lotfi-Kamran, P. et al. “BARP-A Dynamic Routing Protocol for Balanced Distribution of Traffic in NoCs”. In: DATE’08, pp. 1408-1413. *DOI= http://doi.acm.org/10.1145/1403375.1403716*.

[11] Al Faruque, M. A. et al. “Run-time Adaptive On-chip Communication Scheme”. In: ICCAD’07, pp. 26-31.

[12] Gratz, P. et al. “Regional Congestion Awareness for Load Balance in Networks-on-Chip”. In: HPCA’08, pp. 203-214. *DOI= http://dx.doi.org/10.1109/HPCA.2008.4658640*.

[13] Lattard, D. et al. “A Reconfigurable Baseband Platform Based on an Asynchronous Network-on-Chip”. IEEE Journal Of Solid State Circuits, v. 43(1), 2008, pp 223-235. *DOI= 10.1109/JSSC.2007.909339.*