PreNoc: Neural Network based Predictive Routing for Network-on-Chip Architectures

Michel A. Kinsy, Shreeya Khadka and Mihailo Isakov
Adaptive and Secure Computing Systems (ASCS) Laboratory
Department of Electrical and Computer Engineering
Boston University

ABSTRACT

In this paper, we introduce a neural network based predictive routing algorithm for on-chip networks which uses anticipated global network state and congestion information to efficiently route network traffic. The core of the algorithm is a multi-layer neural network machine learning approach where the inputs are level of occupancy of virtual channels, average latency for a particular router to be selected for route computation, the probability of virtual channel allocation, and the probability of winning switch arbitration at the crossbar. The algorithm lends itself to both node routing and source routing. To evaluate the **PreNoc** routing algorithm, we simulate both synthetic traffic and real application traces using a cycle-accurate simulator. In most test cases, the proposed approach outperforms current deterministic and adaptive routing techniques in terms of latency and throughput. The hardware overhead for supporting the new routing algorithm is minimal.

CCS Concepts

•**Hardware** → **Interconnect;** *Network on chip;*

Keywords

Network on chip; NoC; Adaptive Routing; Artificial Neural Network; Predictive Routing

1. INTRODUCTION

The exponential growth in transistor count has led to the integration of more processors onto a single chip. Computer systems with several distinct CPU cores on a single die are now standard with commercially available multicore designs containing 64 or more cores. Multicore and many-core systems raise new sets of design challenges, one of them being the communication infrastructure between the cores. Initially, buses were used in establishing communication between the cores and system components, but bus-based communication techniques do not scale well beyond 16 cores. As an alternative, network-on-chip (NoC) architectures have been proposed.

Routing algorithms for NoC architectures can be generally classified into oblivious and adaptive. With oblivious routing, which includes deterministic routing algorithms as a subset, the path followed by a packet is statically determined. This method allows each node in the network to make its routing decisions independently from the others. By virtue of this distributed aspect, oblivious routing, such as dimension order routing, enables simple and fast router designs, and is widely adopted in early NoC-based systems.

However, oblivious algorithms perform poorly if the network state changes, as in the case of transient or permanent faults, or if the application has certain communication patterns, such as bursty data transfers. The performance reduction occurs because application and network state information are not used for computing routes. Alternatively, an adaptive routing algorithm can use state information, for example network congestion, to adjust to application phases and network changes when making routing decisions. With its dynamic load balancing, adaptive routing theoretically outperforms oblivious routing. However, in the past, adaptive routing schemes have faced the difficult challenge of balancing adaptiveness with router complexity. The vast majority of these algorithms have never been implemented because of the complexity of the router hardware that they require.

The challenge of designing adaptive routing algorithms which achieve good network load balancing, relatively short paths for packets, and modest router architecture complexity, is still relevant. Furthermore, in current multicore and many-core designs large portions of silicon are effectively "dark"– either idle for long periods of time or significantly underclocked at the nominal operating voltage. This has made adaptive routing more relevant, since oblivious routing is inadequate for dealing with the network state changes. Esmaeilzadeh et al. [4] predict that over 50% of chips will not use the ITRS scaling at 8nm technology node. Taylor [12] estimates that in few years, designs may be 93.75% dark.

In this work, we present a predictive network-on-chip routing algorithm, named **PreNoc**, that uses a neural network based learning technique to route packets for better performance and power efficiency. To achieve the best performance, adaptive routing algorithms generally need global knowledge of the current network status. However, dynamically obtaining a global and instantaneous view of the network is often impractical due to changing propagation delays. As a result, adaptive routing algorithms in practice

GLSVLSI '17, May 10 - 12, 2017, Banff, AB, Canada

ISBN 978-1-4503-4972-7/17/05…$15.00

DOI: http://dx.doi.org/10.1145/3060403.3060406

have relied on local knowledge, which limits their effectiveness. The proposed algorithm mitigates this problem by first learning the congestion and arbitration delays in the network, then predicting the future network states and making routing decisions without requiring real-time network state information.

2. RELATED WORK

Ma et al. [10] introduced DBAR - Destination-Based Adaptive Routing. The algorithm uses both local and global network information in estimating network congestion levels. In [3], Ebrahimi et al. presented an adaptive routing algorithm based on minimal and non-minimal paths. Using local and global congestion information, the algorithm is able to estimate the latency associated with each output channel on the routing path to the destination switch. To avoid some the pitfalls associated with global network load balancing, Gratz et al. [6] proposed RCA - Regional Congestion Awareness: a lightweight adaptive routing scheme that makes path selection decisions based on the aggregated congestion information from adjacent routers. The **PreNoc** algorithm can be applied effectively to routing techniques by utilizing regional and global congestion information. Cai et al. [1] proposed a congestion prediction algorithm. It uses a hamming network to compute the link buffer congestion, finds the worst congested nodes, and then adapts to avoid congested nodes. With DyXY routing [9], packets are sent to either X or Y direction depending on the congestion conditions. It uses local information, such as the current queue length of the corresponding input port of neighboring routers to decide on the next hop. Nilsson et al [2] proposed a proximity congestion awareness technique which can be used for uniform load distribution. Information from the neighboring routers is sent from one router to its neighbors in all directions. Ramakrishna *et al.* [11] in their Global Congestion Awareness (GCA) scheme used global link state and congestion information to adapt routing to the network state. Farahnakian et al. [5] introduced a congestion aware dual-reinforcement routing algorithm where a two-way learning approach allows the source node to learn the best way to get to the destination node and from the destination node back to source node. These works have informed and influenced the design of **PreNoc** algorithm but with a key difference in path selection. In the proposed routing algorithm, path selection is done in a predictive manner as opposed to the general reactive way.

3. PREDICTIVE ROUTING ALGORITHM

To support the adaptive routing algorithms, NoC-based systems need to implement mechanisms to monitor and collect the network state data. Adaptive routing algorithms analyze this state information, for example network congestion, to make their routing decisions. In this work, we use neural networks to make routing decisions. Due to the development of more sophisticated and efficient machine learning techniques, practical neural network based adaptive routing algorithms have become more feasible.

To present **PreNoc** algorithm, the following standard definitions of network graphs are used.

DEFINITION 1. *Given a network graph $G(V, E)$, where a directed edge $(u, v) \in E$ has capacity $c(u, v)$ and buffer space $b(u, v)$. The capacities $c(u, v)$ are the available bandwidths on the edge and buffer spaces $b(u, v)$ are available buffer spaces associated with input port (u, v).*

There is a set of $N(N-1)$ source and destination pairs in the network. The set of possible flows is $F = \{f_1, f_2, ..., f_n\}$. $f_i = (s_i, t_i, d_i)$, where s_i and t_i are the source and destination, respectively, for pair i, and d_i is the demand. In this work the assumption is $s_i \neq t_i$ The flow f_i is the set of all data transfers from source s_i to destination t_i.

DEFINITION 2. *The amount of flow f_i along edge (u, v) is $f_i(u, v)$. A route is a path p_i from s_i to t_i for a flow i. Edges along this path will have $f_i(u, v) > 0$, other edges will have $f_i(u, v) = 0$. If $f_i(u, v) > 0$, then route p_i will use both bandwidth and buffer space on the edge (u, v).*

The objective function of the routing algorithm is to (1) minimize the maximum channel load (MCL) across all network links to avoid premature network saturation, and (2) control the average path length to maximize network throughput and minimize power usage.

3.1 Network State Estimation

The **PreNoc** algorithm is a hybrid approach that uses both *node routing* and *source routing*. The network state estimation is performed in a distributed fashion using local and global information. Resource sharing conflicts are at the root of network congestion, premature saturation and lower network throughput. In wormhole routers, the routing operation takes four steps or phases; namely, ① routing (RC), ② virtual channel allocation (VA), ③ switch allocation (SA), and ④ switch traversal (ST). Resource sharing conflict may arise in three of those four steps: at the buffer read and route computation level, at the virtual-channel allocation level, and at the switch arbitration level. Figure 1 shows the conventional router architecture and the three contention points, 1 through 3.

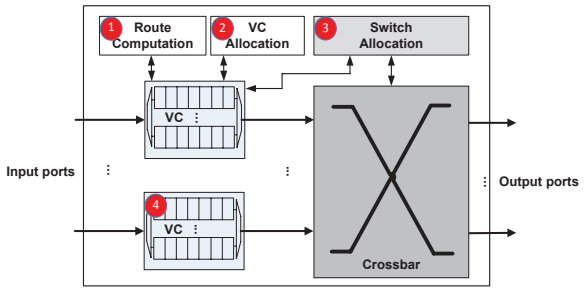


Figure 1: Conventional wormhole router and resource sharing conflict points [1-3].

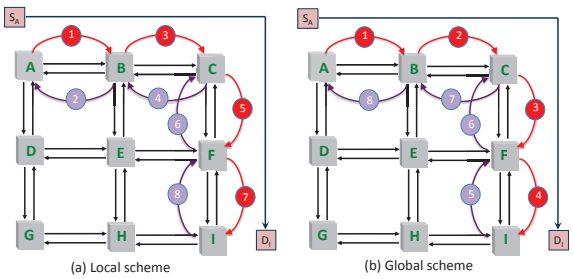


Figure 2: Cost computation protocols: local (a) and global (b).

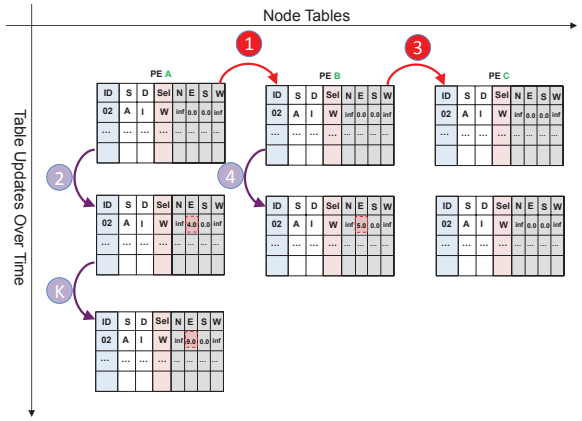


Figure 3: Routing cost tables for nodes A, B, and C and local cost updates scheme.

Each router node executes its own version of the **PreNoc** algorithm and stores the "best" learned routing costs to the other nodes in the network in its *routing cost table*, shown in Figure 3. The table entries are flow *id*s, source node, destination node, cost estimates per direction and the "best" cost to reach destination. Having the flow *id*s and source node in the table allows assigning different costs to the same direction depending on the flow *id* or source node. This schema represents the upper-bound size requirement for *routing cost table*. It takes into account the effects of head-off-line blocking. The table can be substantially reduced by (1) removing the source node and flow *id* distinctions – leading to $n-1$ entries per table, (2) limiting the learning range to some k hops and finally (3) applying routing constraints to the table size – for example XY-routing.

In the *node routing* mode, a node only directly calculates the routing cost to reach its adjacent neighbors, then uses the estimated costs of those neighbors to make its global prediction. The computation propagates forward to the destination then backward to the source node. Figure 2 (a) shows the different phases and Figure 3 shows the corresponding *routing cost table* updates. Initially, costs in all the tables for all the valid directions are set to zero. Although, they could be configured to some predetermined values learned off-line. To route a packet from **A** to **I**, in phase **1**, node **A** selects to go through its east link to **B**. In this illustrative example, the actual cost for routing the packet from **A** to its neighbor **B** is 4.0 which is sent from **B** to **A** as part of the routing credit message. Node **A** updates its routing cost through the east link (**E**) from 0.0 to 4.0 in phase **2**. In the following phases, node **B** forwards the packet to node **C** at a cost of 5.0 and updates its *routing cost table*. When node **A** later computes its eastbound cost, it will be 9.0.
For the *source routing*, *routing cost table* updates happen in the reverse order initiated at the destination node. It uses the global cost computation approach where the true routing cost of the packet does not arrive at node **A** until phase **8**, shown in Figure 2 (b). Although the global cost can be viewed as being less-adaptive, its routing cost estimates are exact in the beginning of the learning process.

3.2 Cost Computation

The cost computation, denoted by $C_j(f(s_i,t_i,d_i))$, is a weighted sum of the degree of goodness of considered paths from node j to destination t_i with nearest neighbors more heavily weighted than distant neighbors. The one hop cost calculation is shown in **Equation** 1 and has four key variables: (1) a_{uv_q} is the average route computation latency for VC q at the input port, p_{uv}, (2) p_{uv_q} the probability of getting through VC allocation without stall, (3) p_{uv_X} is the probability of winning arbitration at the crossbar without stall and (4) $b(u,v)$ represents the level of occupancy of virtual channels associated with the link (u,v). The weight parameters w_a, w_q, w_X and w_b measure the degree of impact of contentions 1 to 3 and amount of free buffer space at the input port. The parameters are also dynamic as they are learned at runtime via the neural network, and provide the predictive part of the algorithm. When the destination node is more than one hop away, **Equation** 2 is used to get the aggregated cost to the destination. In the equation below k represents the number of hops away from j.

$$C_j(f(s_i,t_i,d_i)) = w_a * a_{ij_q} + w_q * \frac{1}{p_{ij_q}} + w_b * b(i,j) + w_X * \frac{1}{p_{ij_X}} \quad (1)$$

$$\begin{aligned} C_{[jk]}(f(s_i,t_i,d_i)) &= w_1 * C_{j+1}(f(s_i,t_i,d_i)) \\ &+ w_2 * C_{j+2}(f(s_i,t_i,d_i)) \\ &... + w_k * C_{j+k}(f(s_i,t_i,d_i)) \\ &Where \;\; w_1 \; through \; w_k \; are \; weights \\ &And \; w_1 \le w_2 \le ... \le w_k. \end{aligned} \quad (2)$$

Algorithm 1 calculates the local routing cost and Algorithm 2 performs the global recursive computation. These two algorithms are integrated to derive the hybrid local and global routing cost estimation.

Initialization
Temporary calculated cost $T_{cost} = 0$;
Best routing cost $B_{cost} = \infty$;
Select direction dir = NULL ;
for $j \in i_{neighbors}$ **do**
  $T_{cost} = C_j(f(s_k,t_k,d_k))$;
  if $T_{cost} < B_{cost}$ **then**
    $B_{cost} = T_{cost}$;
    $dir = (i,j)$;
  end
end

Algorithm 1: Local Routing Cost Estimation - ψ_l

3.3 Deadlock Freedom

Since the the algorithm can be used for both minimal and non-minimal routing, it is not deadlock-free by default. For deadlock freedom, the network channel dependency graph can be made acyclic by prohibiting certain turns [8]. The process of making the CDG acyclic essentially consists of removing certain routes from all possible routes in the on-chip network to guarantee deadlock-free routing. For the network performance, the routing algorithm still needs to select a route (or an optimal route) from all the allowed routes under a given Acyclic CDG (ACDG). Routing table entries at each router could be set to (1) the maximum number of edges that a packet could take in the ACDG (for example, 2-entry tables for 2-D mesh networks using dimension-order routing like XY or YX) or (2) the maximum number of edges that a packet could take in the original CDG with some default table entry value for the restricted routes generated by the

Initialization
Source node src_{node};
Destination node dst_{node};
Temporary calculated cost $T_{cost} = 0$;
Best direction cost $D_{cost} = \infty$;
Best path cost $P_{cost} = 0$;
Select path = $\emptyset$;
Current node $cur_{node} = src_{node}$;
while *(cur_{node} != dst_{node})* **do**
  $i = cur_{node}$;
  $next_{hop}$ = NULL;
  for $j \in i_{neighbors}$ **do**
    $T_{cost} = C_j(f(s_k, t_k, d_k))$;
    if $T_{cost} < D_{cost}$ **then**
      $D_{cost} = T_{cost}$;
      $next_{hop} = j$;
    end
  end
  $path = path \cup (i, next_{hop})$;
  P_{cost} += D_{cost};
  $D_{cost} = \infty$;
  $cur_{node} = next_{hop}$;
end
Algorithm 2: Global Routing Cost Estimation - ψ_g

ACDG (for example 4-entry tables for 2-D mesh networks using dimension-order routing like XY or YX).

3.4 Predictive Model Using Neural Network

The **PreNoc** algorithm is based on the conventional hidden layers *artificial neural network* (ANN). Backpropagation is used to train the ANN in predicting the optimal path for a given flow. The process involves feed-forward computation and backpropagation from the output layer to the input layer with weight adjustments. Although, the ANN is configured to output a routing direction, its main function is to train the weight parameters w_a, w_q, w_X and w_b for the cost computation.

The input parameters to the input layer are $b(u, v)$, a_{uv_x}, where x is a VC at port (u, v)), p_{uv_x} and p_{uv_X}. For a 2D-Mesh network, there are four nodes on the output layer. Each node represents the direction for the next hop; and its output value is the routing cost associated with that direction. The weights between the layers are randomly initialized. In our experimental setup, weights are set to small values in the range of $[-0.5, 0.5]$; the learning rate and the momentum are set to 0.15 and 0.9, respectively, and the default *Sigmoid* function is $f(x) = \frac{1}{1+e^{(-l_j)}}$, where l_j is the layer node j. In active training, after each route selection (r_s), a preferred route (r_p) decision is fed back into the ANN. On the output layer the total prediction error is calculated based the *mean square error* (MSE): $E = \frac{\sum_{i=1}^{m}(r_{p_i} - r_{s_i})^2}{m}$, where m is 4 for the 2D-Mesh. We use standard backpropagation with partial derivatives.

3.5 Predictive Routing Approach

The **PreNoc** approach is a predictive scheme and defers from previous adaptive routing techniques. The routing is done in phases and in a iterative fashion. First, there is a learning phase where routers collect network state information, learn and forecast communication patterns. Second, there is a monitoring phase to validate the information learned in the first phase. In the final phase, routing tables are updated. For a user-defined period of time or number of system cycles, no learning or modification of routing paths is done. Routing during this period, named called *Oblivious Routing Phase* (ORP), is done based on the previously learned best routes and oblivious to current transient network conditions. A the end of the ORP, a new validation phase is initiated to test the robustness of routes under current network conditions. If routing performance is above the permissible threshold, the system goes into another ORP without any change to the routing tables. In the case of a degrading performance, the system performs a new learning phase, followed by a validation phase and an ORP. Figure 4 shows an illustrative view of the phases.

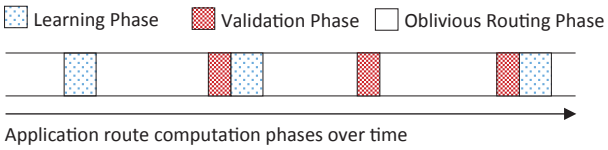


Figure 4: Phase-based hybrid adaptive-oblivious routing approach.

This intermittent aspect of the system allows for (1) the learning phase to not ping pong, (2) the learning logic to converge and stop when no learning is needed, and (3) to disable or power-gate the learning logic. In general, training of the neural networks can take a fair amount of time. Therefore, the initial training and calibration of weights can be performed offline in software to alleviate some of the latency. The online training is ongoing with user-defined break periods or ORPs.

4. MICRO-ARCHITECTURE DETAILS

The conventional wormhole router micro-architecture is modified to include three additional functional units: (1) *Router States Monitor* unit that keeps track of flits, nodes, etc. (2) a hardware engine for executing the ANN algorithm, and (3) modification of the credit message to transport router state information. Figure 5 shows the architecture of the new router.

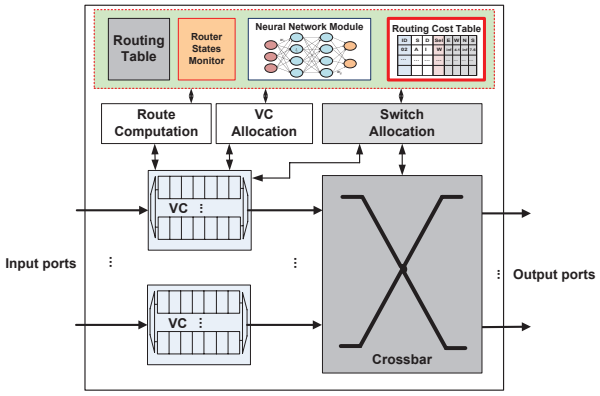


Figure 5: **PreNoc** Micro-Architecture.

Figure 6 highlights the information pieces added to the credit message. In the **PreNoc** architecture, the credit message carries: (1) the conventional credit information (CR) and then (2) the number of free buffer spaces (FB) at the port $log_2(bv \times pv)$-bit where bv is the number of buffer spaces per VC and pv is the number of VCs per port, (3) the arrival time (AT) of each header flit at the particular node

in terms of cycle time is recorded and stored, (4) Latencies are calculated based on the arrival time, route computation, virtual channel allocation and switch allocation. The number of bits used for each of these message components is architecture dependent. In the prototype, 32 bits are used. Instead of sharing the network link bandwidth with program data, a secondary bufferless network is created to route the augmented credit messages. This credit network sends two types of messages: one contains the credit information when the header and the body flits are passing through the router and the second has the routing state information when the tail flit passes through the router. Tables shown in Figure 6 are stored in the *Router States Table*. Learning module including the ANN function is power-gated and can be power-down when not learning.

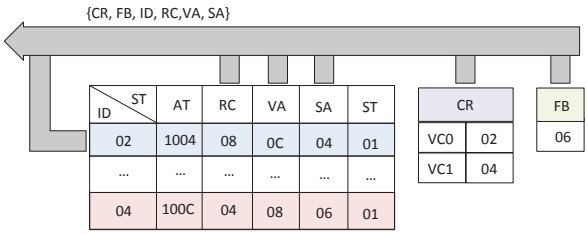


Figure 6: Augmented credit message: state table (ST), credit (CR), free buffer spaces (FB), (ST), header flit arrival time (AT), route computation latency (RC), virtual channel allocation latency (VA), and switch allocation latency (SA).

4.1 Hardware Implementation

To accurately estimate the hardware overhead and complexity of the **PreNoc** router architecture, an FPGA implementation on a Xilinx Virtex7-XC7VX980T FPGA device was performed. The router has 4 virtual channels and 8 slots per virtual channel. Table 1 shows the FPGA synthesis results. The per router register and look-up table (LUT) usage overheads are 9.2% and 9.1%, respectively.

Resource	Conventional	**PreNoc** Router	%
Regs	370829	404945	9.2
LUT	321582	350845	9.1

Table 1: FPGA implementation resource utilization.

There is no change to the critical path in the proposed modification to the router architecture. As in the conventional router, the arbitration logic controls the clock frequency. In both architectures, the clocking speed is 111.83 MHz.

5. EVALUATION

The experimental setup uses *BookSim*, a C++ based cycle-accurate on-chip network simulator [7], to implement and 8x8 2D-Mesh NoC with 4 VC per input ports, 6 buffer spaces in each VC and 8 flits per packet with an exponential distribution. To test the efficiency of our design, flows from both synthetic benchmarks and real applications are used. The synthetic benchmarks, TRANSPOSE, BIT REVERSE, and TORNADO are useful for evaluating specific communication patterns. The real application *H2.64* allows us to test for more general and uneven traffic patterns. For the H.264 application, flow bandwidths are derived from profiling results. Several video streams are run and profiled in deriving the flow demands. The injection rate variations for H.264 workload correspond to frame rates of 4 frames/second to 30 frames/second. For the comparative study, deterministic and other adaptive routing algorithms are implemented: (1) DOR routes packets first in x-direction to the correct column, then in y- direction to the destination node, (2) xy_yx: paths are switched randomly between xy and yx routes to split the traffic, and (3) Adaptive (Adap): it selects between xy and yx routes based on the least congested X or Y neighbor at the source. Simulation runs for 10000 cycles for a warm-up and executes for 100000 cycles for 10 runs. The neural network training takes on average one third of the execution cycles across all benchmarks.

Figures 7a and 7b show the throughput and average latency performance results for the different routing algorithms under the TRANSPOSE benchmark application. The **PreNoc** algorithm outperforms the DOR routing algorithm by 2.25×, the xy_yx algorithm by 1.5× and the adaptive routing by 1.2× in terms of throughput. Figures 7c and 8a present the performance results for the BIT REVERSE benchmark. The performance gab between the **PreNoc** algorithm and the other routing algorithms is smaller. The **PreNoc** algorithm outperforms the adaptive routing algorithm by 8.7% at high injection rates. Even for the less predictable TORNADO benchmark where the routing algorithms have comparable throughput performance, the **PreNoc** approach shows a better saturation point. This performance trend is validated by the H.264 Encoder application, shown in Figures 9a and 9b . The **PreNoc** approach shows a greater aptitude to create routing path diversity and avoid premature network saturation. The **PreNoc**'s performance is very stable and consistently outperforms all the other routing approaches both deterministic and adaptive in terms of both throughput and latency.

6. CONCLUSIONS

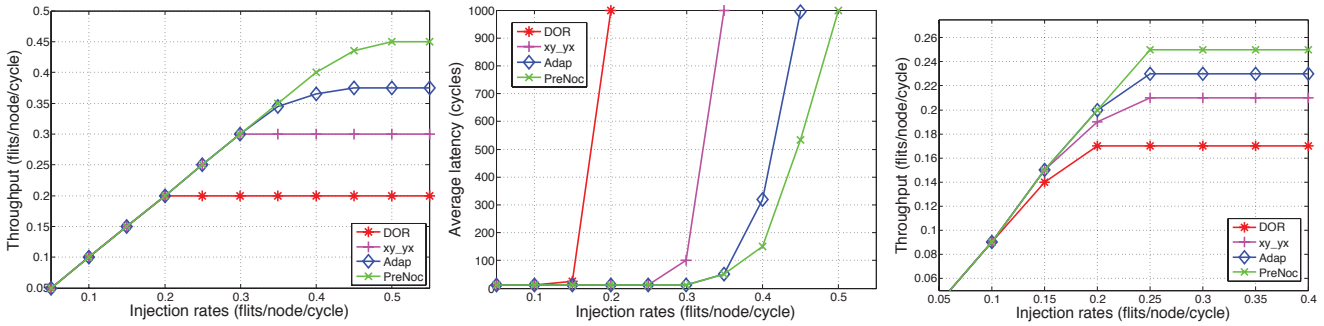
Using different network state information, the **PreNoc** algorithm is able to estimate routing costs and perform low-latency routing of traffic. A router architecture is created to support the algorithm. The hardware overhead for new router is 9.2% when compared to a conventional wormhole design. Overall the **PreNoc** algorithm generally outperforms deterministic and other adaptive routing algorithms in terms of both average packet latency and network throughput. The proposed technique performs especially well at high injection rates.

7. REFERENCES

[1] H. Cai, Y. Y. Yang, F. Qu, J. Wu, and B. Wang. *Congestion Prediction Algorithm for Network on Chip.* Telkomnika, Vol. 11, No. 12, pp. 7392-7398, 2013.

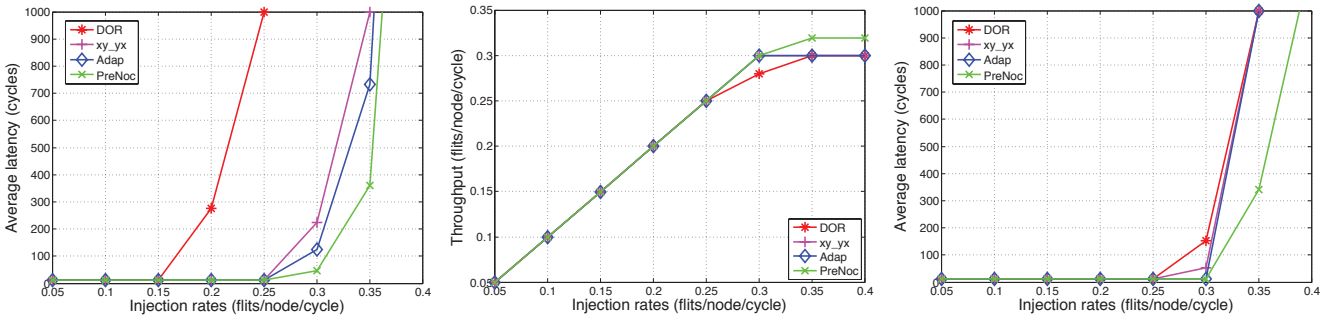
[2] J. O. E. Nilsson, M. Millberg and A. Jantsch. *Load Distribution with the Proximity Congestion Awareness in a Network-on-Chip.* In Proceedings of the Design Automation and Test in Europe Conference, pp. 1126-1127, December 2003.

[3] M. Ebrahimi, M. Daneshtalab, F. Farahnakian, J. Plosila, P. Liljeberg, M. Palesi, and H. Tenhunen. Haraq: Congestion-aware learning model for highly adaptive routing algorithm in on-chip networks. In *NOCS*, pages 19–26. IEEE, 2012.



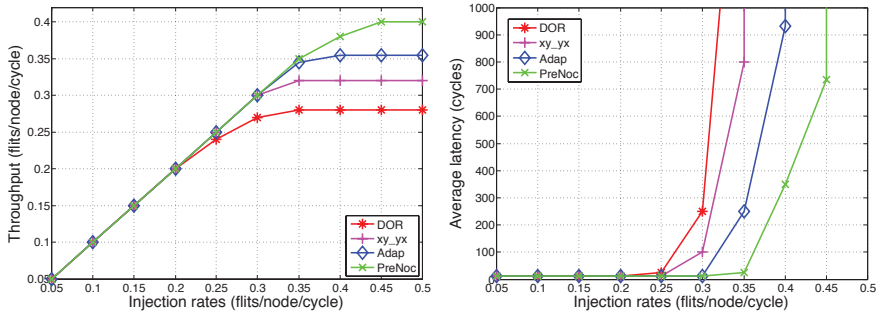
(a) Throughput results for TRANSPOSE. (b) Latency results for TRANSPOSE. (c) Throughput results for BIT REVERSE.

Figure 7: TRANSPOSE and BIT REVERSE synthetic benchmarks results.



(a) Latency results for BIT REVERSE. (b) Throughput results for TORNADO. (c) Latency results for TORNADO.

Figure 8: BIT REVERSE and TORNADO synthetic benchmarks results.



(a) Throughput results for H2.64 ENCODER. (b) Latency results for H2.64 ENCODER.

Figure 9: H2.64 ENCODER benchmark results

[4] H. Esmaeilzadeh, E. Blem, R. St Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 365–376. IEEE, 2011.

[5] F. Farahnakian, M. Ebrahimi, M. Daneshtalab, J. Plosila, and P. Liljeberg. Adaptive reinforcement learning method for networks-on-chip. In *Embedded Computer Systems (SAMOS), 2012 International Conference on*, pages 236–243, July 2012.

[6] P. Gratz, B. Grot, and S. Keckler. Regional congestion awareness for load balance in networks-on-chip. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 203–214, Feb 2008.

[7] N. Jiang, J. Balfour, D. U. Becker, B. Towles, W. J. Dally, G. Michelogiannakis, and J. Kim. A detailed and flexible cycle-accurate network-on-chip simulator, 2013.

[8] M. Kinsy, M. H. Cho, K. S. Shim, M. Lis, G. Suh, and S. Devadas. Optimal and heuristic application-aware oblivious routing. *Computers, IEEE Transactions on*, 62(1):59–73, Jan 2013.

[9] Q. Z. M. Li and W. B. Jone. *DyXY - A Proximity Congestion-Aware Deadlock-Free Dynamic Routing Method for Network on Chip*. DAC pp. 849-852, 2006.

[10] S. Ma, N. Enright Jerger, and Z. Wang. Dbar: An efficient routing algorithm to support multiple concurrent applications in networks-on-chip. *SIGARCH Comput. Archit. News*, 39(3):413–424, June 2011.

[11] M. Ramakrishna, P. V. Gratz, and A. Sprintson. Gca: Global congestion awareness for load balance in networks-on-chip. In *2013 Seventh IEEE/ACM International Symposium on Networks-on-Chip (NoCS)*, pages 1–8, April 2013.

[12] M. B. Taylor. A landscape of the new dark silicon design regime. *Micro, IEEE*, 33(5):8–19, 2013.