Prediction-based Flow Control for Network-on-Chip Traffic

Umit Y. Ogras and Radu Marculescu
Department of Electrical and Computer Engineering
Carnegie Mellon University, Pittsburgh, PA, USA
e-mail: {uogras,radum}@ece.cmu.edu

ABSTRACT

Networks-on-Chip (NoC) architectures provide a scalable solution to on-chip communication problem but the bandwidth offered by NoCs can be utilized efficiently only in presence of effective flow control algorithms. Unfortunately, the flow control algorithms published to date for macronetworks, either rely on local information, or suffer from large communication overhead and unpredictable delays. Hence, using them in the NoC context is problematic at best. For this reason, we propose a predictive closed-loop flow control mechanism and make the following contributions: First, we develop traffic source and router models specifically targeted to NoCs. Then, we utilize these models to predict the cases of possible congestion in the network. Based on this information, the proposed scheme controls the packet injection rate at traffic sources in order to regulate the total number of packets in the network. Evaluations involving real and synthetic traffic patterns show that the proposed controller delivers a superior performance compared to the traditional switch-to-switch flow control algorithms.

Categories and Subject Descriptors
B.4 [**Hardware**]: Input/output and data communications.

General Terms
Algorithms, Performance, Design.

Keywords
Multi-processor systems, networks-on-chip, flow control, congestion control.

1. INTRODUCTION

Network-on-Chip (NoC) architectures have been proposed to address the communication problems generated by the increasing complexity of the single chip systems [1,4,9]. While the NoC architectures offer substantial bandwidth and concurrent communication capability, their performance can significantly degrade in the absence of an effective flow control mechanism. Such a control algorithm tries to avoid resource starvation and congestion in the network by regulating the flow of the packets which compete for shared resources, such as links and buffers [2,5].

In the NoC domain, the term flow control was used almost exclusively in the context of switch-to-switch [4,7,8,11,18] or end-to-end [16] transport protocols. These protocols provide a smooth traffic flow by avoiding buffer overflow and packet drops. However, the flow control can also regulate the packet population in the network by restricting the packet injection to the network [2][1]. This is precisely the main objective of this paper. To the best of our knowledge, this is the first study which addresses the congestion control problem in the NoC domain.

1.1. Overall approach

Switch-to-switch flow control algorithms, such as on-off, credit-based and ack/nack mechanisms, regulate the traffic flow *locally* by exchanging control information between the neighboring routers. These approaches have a small communication overhead, since they do not require explicit communication between source/sink pairs. However, the switch-to-switch flow control does not regulate the actual packet injection rate directly at the traffic source level. Instead, it relies on a backpressure mechanism which propagates the availability of the buffers in the downstream routers to the traffic sources. Consequently, before the congestion information gets the chance to reach the traffic sources, the packets generated in the meantime can seriously congest the network.

End-to-end flow control algorithms, on the other hand, conserve the number of packets in the network by regulating the packet injection rate right at the source of messages. For example, in window-based algorithms, a traffic source can only send a limited number of packets before the previously sent packets are removed from the network. However, the major drawback of end-to-end control algorithms is the large overhead incurred when sending the feedback information [2]. Besides this, the unpredictable delay in the feedback loop can cause unstable behavior as the link capacities increase [12]. Since this is very likely to happen in NoCs, such algorithms are not directly applicable to regulate the NoC traffic.

Starting from these considerations, in this paper, we propose a predictive flow control algorithm which enjoys the simplicity of the switch-to-switch algorithms, while directly controlling the traffic sources, very much like the end-to-end algorithms. Towards this end, we first present a novel router model based on state space representation, where the state of a router is given by the amount of flits already stored in the input buffers. Using this model, each router in the network predicts the *availability* of its input buffers in a *k-step* ahead of time manner. These availability values are computed via an aggregation process using the current state of the router, the packets currently processed by the router, and the availability of the immediate neighbors. Since all predictions are based on the data the routers receive directly from their immediate neighbors, the computations are decentralized and no global data exchange is required. Moreover, we note that the availability information computed at time n is obtained by aggregating the availability of the immediate neighbors at time $n-1$. This information,

1. This function is also referred as congestion control. However, following the convention in [2] and [6], we do not make such a distinction.

in turn, reflects the state of the routers situated two hops away, at time $n-2$, and so on so forth. Therefore, due to the aggregation process the *local* predictions actually reflect the global view of the network. Finally, the traffic sources utilize the availability of the local router to control the packet generation process and avoid excessive injection of packets in the network.

1.2. Related work and paper contribution

From a flow control perspective [5,6], the majority of work presented in the NoC domain relies on the switch-to-switch flow control [4,7,8,11,16,18]; this is primarily due to the large overhead incurred by the end-to-end flow control algorithms. The work presented in [16] employs the end-to-end flow control for guaranteed service in addition to the basic link-level control. A comparison of the overhead of flow control algorithms can be found in [15].

Congestion control is well studied for macronetworks [2,6,12,17]. In [12], the authors develop a decentralized control system, where the sources adjust their traffic generation rates based on the feedback received from the bottleneck links. A predictive explicit-rate control mechanism is presented in [17]. The authors consider a single bottleneck node and infinite buffering resources. The sources adjust their traffic rates using the congestion information received from the bottleneck node via control packets.

Our approach is different from the previously mentioned work in a number of ways. First, our technique is computationaly light since it relies on *local* data transfers, similar to the basic switch-to-switch flow control. At the same time, due to the aggregation performed at each router, the information exchanged between the switches actually reflects the global view of the network. Furthermore, since the predictions reflect the state of the network k steps ahead in time, the packet sources across the network can sense a possible congestion situation early on and then adapt in order to avoid excessive packets injection to the network.

We note that, in real applications, the *best effort* (or non-real time) and *guaranteed service* (or real time) traffic coexist. The proposed framework is a convenient way to control the *best effort* traffic such that it makes best use of network bandwidth without sacrificing the bandwidth allocated to the *guaranteed service* traffic.

1.3. Paper organization

Section 2 and Section 3 present the traffic source and router models, respectively. In Section 4, we propose a flow control mechanism for NoCs. Experimental results appear in Section 5. Finally, Section 6 concludes the paper.

2. SYSTEM AND TRAFFIC MODELS

2.1. System model and basic assumptions

We assume that the network nodes are populated with processing and storage elements (referred to as PEs) but we do not make any assumption about the underlying network topology. The nodes communicate by exchanging packets across the network. Since we consider wormhole routing, the packets are divided into flits. The length of a packet (S) is measured by the number of flits it contains. For convenience, the flit size is assumed to be equal to the physical channel width (W).

In order to avoid packet loss, a basic link-level ON-OFF flow control mechanism is implemented at the routers [5]. The proposed predictive control technique works together with this link-level mechanism to control directly the behavior of the traffic sources.

2.2. Traffic source model

Since the traffic injection rate into the network is the main knob for source control, an accurate model of the input traffic is necessary. Such a model will not only show how the input traffic can be handled by the actual design, but also describe its impact on the packet delay in the network. Towards this end, we observe that the NoC nodes can be in two different states:

OFF STATE: The PE is either processing data or waiting for new data. While in this state, the PE does not generate traffic (hence the name OFF) as shown in Figure 1.

ON STATE: The PE injects packets to the network so the traffic source and its corresponding state are referred to as ON. In this state, the source injects packets in a bursty manner until the message is completely transmitted.

Let the discrete time stochastic process $\lambda(t),\ t \in Z^+$ denote the instantaneous flit injection rate at time t. The cumulative traffic volume generated up to time t (denoted by $V(t)$) is given by:

$$V(t) = V(t-1) + \lambda(t),\ \ V(0) = 0,\ t \in Z^+ \qquad (1)$$

In the ON state, the flit injection rate $\lambda(t)$ is constant and equal to channel width, *i.e.* $\lambda_{ON} = W$ *bits/sec*. If a header flit is injected to the network at time t_0, one can see that $\lambda(t_0 + \Delta) \neq 0$ *for* $0 < \Delta < S$, where S is the packet size in flits. Similarly, when the PE is in the OFF state, one can get an idea of how much longer the OFF state will continue, given the amount of time already spent for processing and type of processing done by the PE. Therefore, the inter-arrival times are *not* memoryless, (*i.e. not* exponentially distributed) and so the flit injection process cannot be Poisson. Consequently, we need to modify the classical ON/OFF [13] model to work for NoC traffic sources.

2.2.1. Distribution of t_{ON}

The duration of the ON state is determined by the size of the packets generated by the node and λ_{ON}; specifically, $t_{ON} = \lceil SW/\lambda_{ON} \rceil$. While λ_{ON} is constant, S depends on the particular packet (or packets) generated by the source after completing a certain task. In an NoC, the type of the tasks performed by each PE and the size of the resulting message are typically known at design time. For example, a DSP core implementing DCT/IDCT operations in a multimedia chip, can only produce the cosine or inverse cosine transforms of a fixed size data block. Hence, S can take only certain discrete values, usually known at design time. Note that, this is in stark contrast with a general purpose network, where a node can generate a much wider range of messages. As such, we model the probability mass function F_{ON} as:

$$F_{ON}(t) = p(t_{ON} \le t) = \sum_{i=0}^{t} p(t_{ON} = i) \qquad (2)$$

We can actually compute $F_{ON}(t)$, since the communication volume between the network nodes and λ_{ON} are known at design time.

2.2.2. Distribution of t_{OFF}

The duration of the OFF state is the sum of two random variables. The first is the processing time of the PE, t_{proc}; this can take certain discrete values, based on the number of different tasks implemented by the PE. Therefore, t_{proc} is a discrete random variable with discrete probability mass function:

$$F_{proc}(t) = p(t_{proc} \le t) = \sum_{i=0}^{t} p(t_{proc} = i)$$

The second component of t_{OFF} is the waiting time (t_{wait}) for new data, before the node cannot start processing. Unlike t_{ON} and t_{proc}, the waiting time t_{wait} can take a wide range of values as it depends on the latency in the network. One can express the distribution of t_{OFF} as a function of the waiting time, $p(t_{wait} \le t)$, as follows:

$$F_{OFF}(t) = \sum_{k=1}^{n} p(t_{wait} \le t - t_k | (t_{proc} = t_k)) P(t_{proc} = t_k) \quad (3)$$

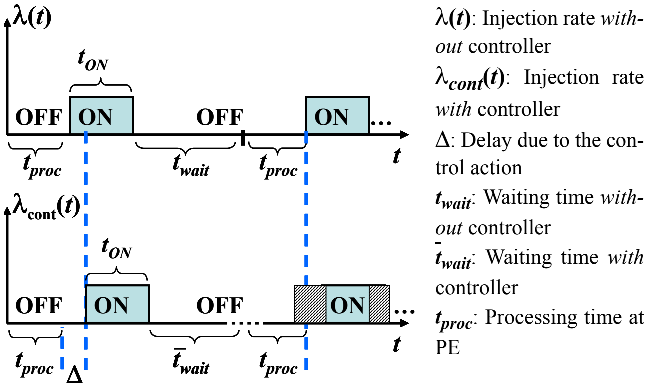
2.3. Predictive control of the traffic sources

Suppose that the ON states of several traffic sources overlap and lead to congestion in the network. Consequently, starting at time t_0, the packets generated by source i cannot be delivered to their destinations. In this scenario, the source i will continue to inject packets to the network until it senses congestion, let say at time $t_0+\Delta$. The number of flits injected during this time is given by:

$$V(t) = min\left(\sum_{t=t_0}^{t_0+\Delta} \lambda(t), \sum B_T\right)$$

where the first element in the tuple represents the total number of flits that can be generated by the source, while B_T is the available buffering space along the path from source i to the congested router. If the interval (t_0, Δ] covers the ON period of the source, it is likely that the source will continue to inject packets until it senses the backpressure effect due to the buffer starvation. This, in turn, can further increase the number of packets in the network and hence make the congestion more severe. Since there are many sources sharing the same network resources, it is extremely important to minimize Δ.

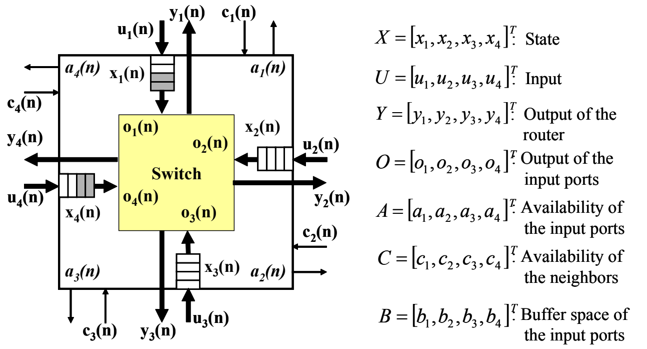
Δ can be reduced by predicting the possible congestion before it becomes severe and propagating this information to all traffic sources. Since the availability in the routers may indicate congestion, the traffic sources can send a packet to the router only if its availability of greater than zero. Otherwise, the traffic source can *delay* the packet injection until the resource availability improves, as illustrated in Figure 1. Delaying the packet injection effectively regulates the total number of packets in the network, hence the average packet latency. While the precise time for packet injection



λ(t)
t_ON
OFF
ON
OFF
ON
t_proc
t_wait
t_proc
t
λ_cont(t)
t_ON
OFF
ON
OFF
ON
t_proc
Δ
t̄_wait
t_proc
t
λ(t): Injection rate without controller
λ_cont(t): Injection rate with controller
Δ: Delay due to the control action
t_wait: Waiting time without controller
t̄_wait: Waiting time with controller
t_proc: Processing time at PE

Figure 1. Illustration of the ON-OFF source model and the control action. By delaying the start of the ON period, the waiting time in the network can be reduced.

is difficult (if not impossible) to find at design time, an online predictor can guide the packet generation at the source in order to utilize the network resources in the best possible way.



u1(n)
y1(n)
c1(n)
a4(n)
x1(n)
a1(n)
c4(n)
o1(n)
o2(n)
x2(n)
y4(n)
u2(n)
Switch
u4(n)
o4(n)
y2(n)
x4(n)
o3(n)
c2(n)
x3(n)
a3(n)
a2(n)
c3(n)
y3(n)
u3(n)
X = [x1, x2, x3, x4]^T: State
U = [u1, u2, u3, u4]^T: Input
Y = [y1, y2, y3, y4]^T: Output of the router
O = [o1, o2, o3, o4]^T: Output of the input ports
A = [a1, a2, a3, a4]^T: Availability of the input ports
C = [c1, c2, c3, c4]^T: Availability of the neighbors
B = [b1, b2, b3, b4]^T: Buffer space of the input ports

Figure 2. The state variables, inputs and outputs of a 4-port router are shown.

3. STATE SPACE MODEL FOR NoC ROUTER

To obtain accurate predictions, we also need a good model for the NoC router. Traditionally, the network research has been focused on directly computing the router delay [3,14]. Unlike previous work, our goal is to predict how many flits the router can accept over the next k time steps. For this reason, the parameter of interest is the *occupancy* of the router *input buffers*[1].

We propose a state space representation of a NoC router driven by stochastic inputs, as shown in Figure 2. The state of the router at time n is given by the number of flits in its input buffers; that is:

$$X(n) = [x_1(n), x_2(n), \ldots, x_P(n)]^T \quad (4)$$

where $x_P(n)$ is the state of the input port P (*i.e. total number of flits in all of the input buffers associated with port P*) and *'T'* denotes the transposition operation. For instance, a router with d neighboring routers and one local PE connection has (d+1) ports. Hence, $X(n)$ is a $(d+1)\times 1$ vector.

The input received at port P, at time n, is denoted by $u_P(n)$. $u_P(n)$ is equal to 1, if a flit is received at time n and is 0 otherwise. Similarly, the output from port P is represented by $y_P(n)$, where $y_P(n) = 1$ implies that a flit is transmitted to the downstream router, at time n. Consequently, the input and output processes of the router are given by the following $P\times 1$ vectors:

$$U(n) = [u_1(n), u_2(n), \ldots, u_P(n)]^T \quad ,$$

$$Y(n) = [y_1(n), y_2(n), \ldots, y_P(n)]^T \quad (5)$$

Next, we model how the flits are read from the input buffers. $o_P(n) = 1$ means that one flit is read from the input buffer at port P, and the vector $O(n) = [o_1(n),\ldots,o_P(n)]^T$ represents the outcome of reading process from the input buffers. Note that this is different from the outputs $Y(n)$ of the router. The output of the input buffers goes through the crossbar switch and then ends up at one of the router output ports (Figure 2).

As a result, the knowledge of either $Y(n)$ or $O(n)$ provides information about the other, given the connections in the crossbar switch. As a result, the router can be described by an integrator where the next state is determined by the current state, current input and current output processes, as follows:

$$X(n+1) = I_{P\times P}X(n) + U(n) - O(n) \quad (6)$$

1. A similar model for the output buffers can be also developed.

Router stability

The router described by Equation 6 becomes unstable (*i.e.* the state grows unbounded), if the average arrival rate to the router is greater than the rate at which the router can serve any given packet. In practice, however, the input buffers are all finite. Hence, in order to avoid packet loss, no more flits are accepted by the link-level flow control, when the buffers are full. As a result, the router model given in Equation 6 can be refined as:

$$X(n+1) = I_{P \times P}X(n) + U(n)H(n) - O(n) \quad (7)$$

where $H(n) = [h(b_1\text{-}x_1(n), h(b_2\text{-}x_2(n), \ldots, h(b_P\text{-}x_P(n)]^T$. $h(x_i)$ is the unit step function (*i.e.* $h(x_i)=0$ if $x_i<0$, and $h(x_i)=1$ otherwise), and b_1 to b_P represent the capacity of each input buffer.

Finally, solving Equation 7 with respect to a known state $X(n_0)$, gives the state at time $n+n_0$ as

$$X(n+n_0) = X(n_0) + \sum_{j=n_0+1}^{n+n_0} (U(j)H(j) - O(j)) \quad (8)$$

Obviously, the router described by Equation 8 has a bounded response. However, since such a control does not limit the source injection directly, if the average arrival rate becomes larger than the service rate of the router, the input buffers will remain full for most of the time. This, in turn, results in blocked links and large delays in the network. While one could regulate the traffic injection by an open loop controller [5], even under a light load, the packets may experience congestion due to the overlaps between the ON periods of the traffic sources. For instance, consider a 4×4 2D mesh network running hotspot traffic[1]. Although the traffic load is kept low such that the input buffers of the most congested router are empty more than 80% of time and the buffers become full only about 1% of time (Figure 3(a)), about 18% of the packets experience delays more than twice as large as the average delay. The delay histogram is shown Figure 3(b). Such packets will not only block the network resources, but also affect the other packets as well. As a result, we cannot merely rely on such an *open-loop* control scheme. In what follows, we show how the router model presented in this section can be used to implement a predictive flow controller which regulates the traffic injection to the network.

4. PREDICTIVE FLOW CONTROLLER

Unpredictable delays in the feedback loop of flow control algorithms prevent the timely transmission of the congestion information and control signals. To mitigate this problem, we propose a prediction-based control which relies on the traffic source and router models developed in sections 2 and 3, respectively. These models enable us to predict the availability of any router at a future time step, as described next.

4.1. Availability predictor

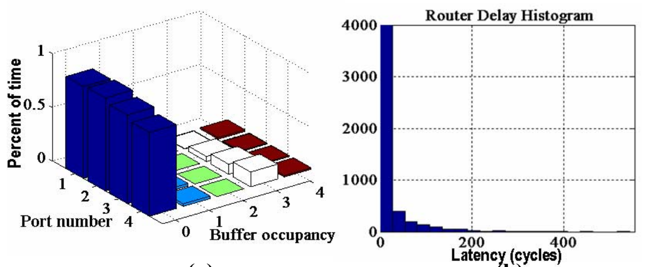
The optimal *k-step* predictor for network state, $\hat{X}(n_0+k|n_0)$, is the conditional expectation at n_0+k, given the state at time n_0 [10]:

$$\hat{X}(n_0+k|n_0) = E[X(n_0+k)|X(n_0), U(n_0)]$$

Using Equation 8, we have:

$$\hat{X}(n_0+k|n_0) = X(n_0) + \sum_{j=n_0+1}^{k+n_0} (E[U(j)H(j)|n_0] - E[O(j)|n_0])(9)$$

1. Under hotspot traffic, all nodes in the network receive packets with uniform probability, while four randomly selected nodes receive some extra traffic.



(a) (b)

Figure 3. (a) Buffer utilization and (b) delay histogram of a router.

where $E[.|n_0]$ stands for $E[.|X(n_0), U(n_0)]$ (for notational simplicity). To compute the *k-step* forward prediction, we need the expected value of input and output processes, given the current state and input. If sufficient processing power is available (*e.g.* when the predictor is implemented in a data macro-network with plenty of resources), then Equation 9 can be directly used to estimate the conditional mean values of the input and output processes to predict the state at n_0+k. However, for NoCs we have to keep the area overhead as small as possible. For this reason, we use Equation 9 to predict how many flits a given input port can accept, over the following k steps, rather than dealing with the absolute value of the state.

We call the number of flits the input port P can accept, over the next k steps, the *availability* of port P and denote it by $a_P(n_0, k)$. $a_P(n_0, k)$ is simply the sum of the number of empty slots in the buffer at time n_0+k, and the number of flits that are expected to be admitted in the following k steps (*i.e.* $b_P - \hat{x}_P(n_0+k) + \sum_{j=n_0+1}^{n_0+k} E[u_P(j)h(b_P - x_P(j))|n_0]$). So the availability vector $A(n_0,k) = [a_1(n_0,k),\ldots,a_P(n_0,k)]$, can be found using Equation 9 as

$$A(n_0, k) = B - X(n_0) + \sum_{j=n_0+1}^{k+n_0} E[O(j)|n_0] \quad (10)$$

where $B = [b_1, b_2, \ldots, b_P]^T$ is the vector containing the depth for each input buffer. Intuitively, $B - X(n_0)$ represents the availability at time n_0, while the last term is the expected number of flits that will be read from the router. Since a new flit can be written to the buffer for each flit being read, the sum of these terms gives the availability for the interval $[n_0, k]$.

The expected value of the read process from the input buffers (the last term in Equation 10) can be approximated using the router output $Y(j)$ as follows:

$$\sum_{j=n_0+1}^{k+n_0} E[O(j)|n_0] = \begin{bmatrix} g_{1,1}(n_0) & \cdots & g_{1,P}(n_0) \\ g_{2,1}(n_0) & \cdots & g_{2,P}(n_0) \\ \cdots & \cdots & \cdots \\ g_{P,1}(n_0) & \cdots & g_{P,P}(n_0) \end{bmatrix} \sum_{j=n_0+1}^{k+n_0} E[Y(j)|n_0] \quad (11)$$

where the coefficients $g_{i,k}(n_0)$ are determined based on the state of the switch matrix and channel allocations in the router. If we let $G(n_0) = \{g_{i,k}(n_0)\}$, then Equation 10 can be written as:

$$A(n_0, k) = B - X(n_0) + G(n_0) \sum_{j=n_0+1}^{k+n_0} E[Y(j)|n_0]$$

Note that $\Sigma_{j=n_0+1}^{n_0+k}(E[Y(j)|n_0])$ is the expected number of flits transmitted by the router in the interval $[n_0, k]$. However, this is nothing but the availability of the immediate neighboring routers. In other words, instead of predicting the number of flits transmitted over the next k steps, we *aggregate* the availability information already predicted by the neighboring routers. As a result, the availability of a router is updated using the following equation:

$$A(n_0, k) = B - X(n_0) + G(n_0)C(n_0 - 1, k) \qquad (12)$$

where the vector $C(n_0\text{-}1, k)$ denotes the availability of the immediate neighbors predicted at time n_0-1, as illustrated in Figure 2.

4.2. Practical implementation of the predictor

The predictor defined by Equation 12 represents an iterative process. The availability predicted at time n_0 is a function of the availability of the neighboring routers predicted at time $n_0 - 1$. The initial availability values are the sum of buffer capacities and the prediction step, *i.e.*

$$A(0, k) = B + k[1, 1, \ldots, 1]_{1 \times P}^{T} \qquad (13)$$

since all of the buffers are empty at time 0.

The computation of the availability depends mainly on the coefficients $g_{i,k}$, as illustrated in Figure 4. The update process consists of distributing the availability of the neighboring routers to the input ports as a function of the current connections in the crossbar switch. For example, when processing the availability data from port 1 (*i.e.* c_1), we first check whether or not one of the input ports is connected to port 1 through the crossbar switch. If this is true (*e.g.* input port 2) and δ flits have been already sent over this connection, the remaining flits coming from the same input port will use this connection. As a result, $min(c_1, S\text{-}\delta)$ flits are allocated to the input port 2, while the remaining slots are distributed among the remaining ports.

The coefficients g_{ik} determine this allocation process. In our implementation, we do this distribution uniformly to minimize the area overhead for the predictor. The allocation requires $(P-1)^2$ *m-bit* full adders and P *m-bit* shifters, where m is the total number of bits needed to express the maximum availability, *i.e.* $log_2(a_1(0, k))$. For instance, in our implementation $m = 4$, and the area overhead of the predictor is in the order of 1000 transistors. We estimate the overhead of the predictor implemented for a 5-port wormhole router with 16x16 bit input buffers to be about 10%. A more precise analysis of the implementation complexity for the predictor is left as future work.

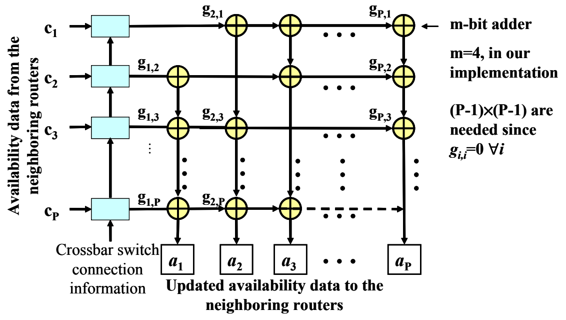

Availability data from the neighboring routers
c1
c2
c3
cP
g2,1
gP,1
g1,2
gP,2
g1,3
g2,3
gP,3
g1,P
g2,P
m-bit adder
m=4, in our implementation
(P-1)×(P-1) are needed since gi,i=0 ∀i
Crossbar switch connection information
a1
a2
a3
aP
Updated availability data to the neighboring routers

Figure 4. Practical implementation of the predictor.

4.3. Using prediction for network control

Each router in the network updates periodically its availability by aggregating the data received from the immediate neighbors using Equation 12. As a result, the availability of a local input port connected to a traffic source reflects the backpressure from all of the downstream routers. As such, when a traffic source sees that the input port connected to it has a zero availability, it delays the generation of new packets until the availability of the port becomes greater than zero. Since the information exchange between the neighboring routers is achieved by a small number (*i.e.* $log_2(a_P(0, k))$) of dedicated control wires, the congestion prediction does not experience long queuing delays. In order to guarantee the timely transmission of the prediction to the traffic sources, k is selected to match the diameter of the network.

5. EXPERIMENTAL RESULTS

Next, we demonstrate the effectiveness of the proposed flow control technique using an audio/video system complying with the H263.1 standard, as well as synthetic benchmarks which are all mapped to a 4×4 2D mesh network. The simulations are performed using a custom cycle-accurate NoC simulator which implements a basic ON/OFF switch-to-switch flow control, the ON/OFF traffic sources and the proposed flow control scheme. The simulations are repeated for a range of buffer sizes in routers and local PEs. The results reported next are obtained for 4-flit input buffers in the routers and 100-flit local memory in the host PE[1].

5.1. Audio/Video traffic

The audio/video system from [8] is first simulated using only the switch-to-switch flow control. When the offered load is about half of the maximum achievable throughput, the average and maximum packet latencies in the network are found to be 149 and 897 *cycles*, respectively. After that, the simulations are repeated with the proposed flow controller. As shown in Table 1, the average packet latency becomes 47 *cycles*, while the maximum packet latency drops to 466 *cycles*. This huge reduction in packet latencies is mainly due to the reduced number of packets in the network.

As mentioned before, unlike the switch-switch flow controller, the proposed controller regulates the number of packets in the network directly. As such, the number of packets in the network drops from 129 to 52 *packets* which is about a 2.5× reduction.

	Switch-to-switch control only	***The proposed flow control***	***Reduction (×)***
Ave. latency	149 *cycles*	47 *cycles*	3.2
Max. latency	897 *cycles*	466 *cycles*	1.9
Ave. # of packets	94 *packets*	29 *packets*	3.2
Max. # of packets	129 *packets*	52 *packets*	2.5

Table 1: The reduction in the latency and number of packets in the network due to the proposed flow control algorithm.

To better understand the effects of the controller, in Figure 5, we further analyze the packet latencies. For the network without the proposed flow controller, about 49% percent of the packets experience longer delays than the average delay (*i.e* 149 *cycles)*. The packets located at the tail of the distribution in Figure 5(a) are the

1. Note that the local memory in the host PE is *not* part of the router. The 100-flit local buffer is used to emphasize that (*i*) its size is finite and (*ii*) PEs sense the backpressure from the network for the switch-to-switch flow control.

main cause for this poor performance. The proposed technique prevents the packets that are likely to experience such long delays from entering the network. Indeed, as depicted in Figure 5(b), the latency histogram is pushed significantly towards left so about 91% of packets experience less than 100 *cycles* latency.

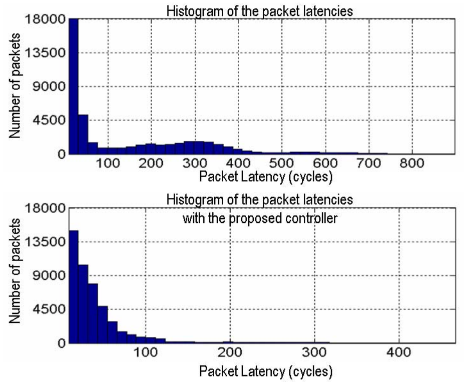


Figure 5. Histogram of the packet latencies without (a) and with (b) the proposed flow controller.

5.2. Synthetic traffic

Similar to Section 5.1, new experiments are performed for uniform and hotspot traffic patterns. First, we compare the performance of a 4×4 2D mesh network under hotspot traffic *with* and *without* the proposed controller. The average packet latency in the network is plotted as a function of the packet injection rate in Figure 6(a). We observe that without the flow controller, the network become congested as the packet injection rate increases. The reason for this behavior is uncovered in Figure 6(b). Indeed, in absence of a traffic controller, as the traffic injection rate increases, the number of packets in the network grows at an increasing pace. The proposed flow controller, on the other hand, effectively limits the number of packets injected to the network, as depicted in Figure 6(b). This, in turn, results in significant improvements in the average packet latency. Finally, Figure 6 (a) and (b) demonstrate that the average packet latency is proportional to the average number of packets in the network and justify once more controlling the packet injection as an effective means for improving the NoC performance.

Local PE buffer size	50-flit	100-flit	200-flit	500-flit
Ave. latency without proposed controller	80 *cycles*	106 *cycles*	158 *cycles*	217 *cycles*
Ave. latency with proposed controller	43 *cycles*	44 *cycles*	44 *cycles*	44 *cycles*

Table 2: Average packet latency for the hotspot traffic (at 0.2 *packets/cycle* traffic rate) w/o and with the proposed controller for different local memory sizes are summarized.

The performance of the proposed predictive controller is further analyzed for various buffer sizes for the local PE (see Table 2[1]). The impact of the controller becomes more pronounced as the size of the buffer in the local PE increases. The switch-to-switch control is less effective for large local buffers, since the PEs cannot deal with the backpressure, hence congestion, in time. On the other hand, reducing the local buffer size does not solve the problem, since in this case the PE has to stall to avoid packet loss. Nevertheless, the proposed controller works efficiently for various buffer sizes, as shown in Table 2.

1. For confidence reasons, each experiment is repeated 50 times with different seeds and average values are reported.

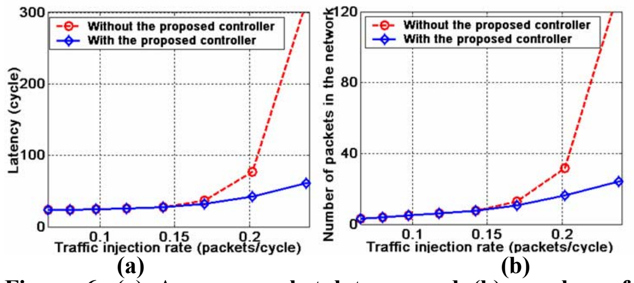


Figure 6. (a) Average packet latency and (b) number of packets in the network are plotted as a function of the injection rate.

6. CONCLUSION

While effective flow control mechanisms are necessary for efficient utilization of network resources, neither switch-to-switch, nor end-to-end control schemes proposed for macro-networks can satisfy the requirements of NoCs. For this reason, we proposed a predictive flow controller based on novel traffic source and router models specifically targeted to NoCs. The proposed scheme controls the packet injection rate in order to regulate the number of packets in the network. Experimental results show that the proposed controller effectively regulates the number of packet in the network and delivers superior performance compared to traditional switch-to-switch flow control algorithms.

7. REFERENCES

[1] L. Benini and G. De Micheli. Networks on chips: A new SoC paradigm, *IEEE Computer*, 35(1), 2002.
[2] D. Bertsekas and R. Gallager, *Data Networks*. Prentice Hall, 1992.
[3] A. A. Chien. A cost and speed model for k-ary n-cube wormhole routers, *IEEE Trans.Parallel and Distributed Systems*, 9(2) 1998.
[4] W. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. *In Proc. DAC*, June 2001.
[5] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.
[6] M. Gerla and L. Kleinrock, Flow control: A comparative survey, *IEEE Trans. on Communications* COM-28, no 4, 1980.
[7] A. Jalabert, *et al.* XpipesCompiler: A tool for instantiating application specific networks on chip. *In Proc. DATE*, March 2004.
[8] J. Hu and R. Marculescu, Energy- and performance-aware mapping for regular NoC architectures, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol.24, No.4, April 2005.
[9] A. Jantsch and H. Tenhunen (Eds.). *Networks on Chip*. Kluwer, 2003.
[10] J. M. Mendel. *Lessons in Estimation Theory for Signal Processing, Communications, and Control*. Prentice Hall, 1995.
[11] E. Nilsson *et. al.* Load distribution with the proximity congestion awareness in a network on chip. *In Proc. DATE*, March 2003.
[12] F. Paganini, J. Doyle, S. Low. Scalable Laws for Stable Network Congestion Control, *In Proc. IEEE Conf. on Decision and Control*, Dec., 2001.
[13] K. Park and W. Willinger (Editors). *Self-Similar Network Traffic and Performance Evaluation*. Wiley Interscience, 2000.
[14] Li-Shiuan Peh and William J. Dally. A delay model for router microarchitectures. *IEEE Micro*, Jan/Feb 2001.
[15] A. Pullini *et. al.* Fault tolerance overhead in network-on-chip flow control schemes. *In Proc. Symp. on IC and System Design,* September 2005.
[16] A. Radulescu *et. al.* An efficient on-chip ni offering guaranteed services, shared-memory abstraction, and flexible network configuration, *IEEE Trans. on CAD of ICs and Systems*, 24(1) 2005.
[17] D. Qiu and N. B. Shro. A predictive flow control mechanism to provide QoS and efficient network utilization, *IEEE Trans. on Networking* 12(1), 2004.
[18] C. A. Zeferino *et. al.* Paris: a parameterizable interconnect switch for networks-on-chip, *In Proc. Symp. on IC and Systems Design*, 2004.