Analysis and Optimization of Prediction-Based Flow Control in Networks-on-Chip

UMIT Y. OGRAS and RADU MARCULESCU
Carnegie Mellon University

Networks-on-Chip (NoC) communication architectures have emerged recently as a scalable solution to on-chip communication problems. While the NoC architectures may offer higher bandwidth compared to traditional bus-based communication, their performance can degrade significantly in the absence of effective flow control algorithms. Unfortunately, flow control algorithms developed 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 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. We also illustrate the proposed traffic source model and the applicability of the proposed flow controller to actual designs using real NoC implementations. Finally, simulations and experimental study using our FPGA prototype show that the proposed controller delivers a better performance compared to the traditional switch-to-switch flow control algorithms under various real and synthetic traffic patterns.

Categories and Subject Descriptors: B.4 [**Input/Output and Data Communications**]

General Terms: Algorithms, Performance, Design

Additional Key Words and Phrases: Multi-processor systems, networks-on-chip, flow control, congestion control

ACM Reference Format:

Ogras, U. Y. and Marculescu, R. 2008. Analysis and optimization of prediction-based flow control in networks-on-chip. ACM Trans. Des. Autom. Elect. Syst. 13, 1, Article 11 (January 2008), 28 pages. DOI = 10.1145/1297666.1297677 http://doi.acm.org/10.1145/1297666.1297677

The authors acknowledge the support of the Gigascale Systems Research Focus Center, one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program.

Authors' address: Department of Electrical and Computer Engineering, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213; email: {radum,uogras}@ece.cmu.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2008 ACM 1084-4309/2008/01-ART11 $5.00 DOI 10.1145/1297666.1297677 http://doi.acm.org/10.1145/1297666.1297677

1. INTRODUCTION

Systems-on-Chip (SoCs) designed at nanoscale will soon contain billions of transistors [Semiconductor Industry Association 2006]. This makes it possible to integrate hundreds of IP cores running multiple concurrent processes on a single chip. As a result, novel on-chip communication solutions that enable the design of complex SoCs are needed. Due to their limited bandwidth and large capacitive load, the legacy bus-based architectures become a performance bottleneck for the design of multicore systems. On the other hand, the point-to-point communication architectures fail to provide enough scalability in terms of area and design effort [Bolotin et al. 2004; Lee et al. 2007].

In contrast to these traditional methods, the Network-on-Chip (NoC) communication architectures have been proposed to address the communication problems generated by the increasing complexity of single chip systems [Benini and De Micheli 2002; Dally and Towles 2001; Guerrier and Greiner 2000; Hemani et al. 2000; Jantsch and Tenhunen 2003]. While the NoC architectures offer substantial bandwidth increase and concurrent communication capability, their performance can significantly degrade in absence of an effective flow control mechanism. Such a control algorithm avoids resource starvation and congestion in the network by regulating the flow of the packets competing for shared resources, such as links and buffers [Bertsekas and Gallager 1992; Dally and Towles 2004].

In the NoC domain, the term flow control was used almost exclusively in the context of switch-to-switch [Dally and Towles 2001; Jalabert et al. 2004; Hu and Marculescu 2005; Nilsson et al. 2003; Zeferino et al. 2004] or end-to-end [Radulescu et al. 2005] 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 [Bertsekas and Gallager 1992].[1] This is precisely the main objective of this article. To the best of our knowledge, this is the first study that addresses the congestion control problem in the NoC domain.

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 that 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. Moreover, wormhole routing is prone to head of line (HOL) blocking which is a significant performance limiting factor. HOL blocking happens when the packet header cannot propagate to the next router due to lack of buffering

[1]This function is also referred as congestion control. However, following the convention in Bertsekas and Gallager [1992] and Gerla and Kleinrock [1980], we do not make such a distinction.

space. When the HOL blocking occurs, all subsequent packets remain blocked and thus the router output ports can starve. Therefore, congestion becomes even more severe for networks that employ wormhole routing [Dally 1992; Smai and Thorelli 1998].

End-to-end flow control algorithms, on the other hand, try to 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 [Bertsekas and Gallager 1992]. Besides this, the unpredictable delay in the feedback loop can cause unstable behavior as the link capacities increase [Paganini et al. 2001].

1.1 Overall Approach and Article Contribution

In this article, 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 an ON/OFF traffic source model. During the ON state, the traffic sources generate packets in a bursty manner until the entire message is transmitted. During the OFF state, on the other hand, the sources are silent, that is, they either process data or wait for new inputs. The knowledge of the target application enables us to characterize the distribution of the ON state. Next, we develop a novel router model based on state space representation, where the state of a router is given by the number of flits already stored in the input buffers. Using the traffic source and router models, 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 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, 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. In summary, the major contributions of this work are as follows:

—First, we present an ON/OFF traffic source model. The knowledge of the target application enables us to precisely characterize the distribution of the ON period. We illustrate this model using an NoC-based implementation of the MPEG-2 encoder.

—Second, we develop a state space model of NoC routers. The proposed model is a powerful tool for the analysis of router behavior. In this work, the proposed

model is used to predict the availability of a given router. However, it can be also utilized for power and performance analysis purposes [Ogras and Marculescu 2007]. The predictions on buffer availability made by the routers are, in turn, used by the traffic sources to decide whether or not to inject new packets in the network. When congestion is likely, the source flow controllers delay the injection of packets until the congestion is resolved.

—Finally, we present extensive experimental evaluation including the hardware implementation of the proposed controller and simulation study involving real and synthetic benchmarks. Through hardware implementation, we show that the proposed controller has a small area (0.011 mm^2 for 0.18 um technology). Furthermore, the simulations show much better performance compared to basic link-level flow control.

1.2 Article Organization

In Section 2, we review the related work. System and traffic source models are presented in Section 3. In Section 4, we develop the router model based on a novel state space representation. The proposed flow controller and its practical implementation are discussed in Section 5. Finally, the experimental results appear in Section 6, and Section 7 concludes the article.

2. RELATED WORK

2.1 Networks-on-Chip

Due to the increasing complexity of SoCs, the design of scalable communication architectures has recently received significant attention [Guerrier and Greiner 2000; Hemani et al. 2000; Sgroi et al. 2001]. The NoC architectures are motivated as a scalable solution for on-chip communication in [Benini and De Micheli 2002; Bolotin et al. 2004; Dally and Towles 2001; Guerrier and Greiner 2000]. Design methodologies for application mapping to NoC architectures and NoC architecture synthesis appear in [Ascia et al. 2004; Hu and Marculescu 2005; Murali et al. 2005] and [Ogras and Marculescu 2005, 2006; Pinto and Sangiovanni-Vincentelli 2003; Srinivasan et al. 2004], respectively. Several concrete NoC implementations are presented in [Adriahantenaina and Greiner 2003; Bjerregaard and Sparso 2005; Dielissen et al. 2003; Lee et al. 2007; Liang et al. 2004; Millberg et al. 2004].

From a flow control perspective [Dally and Towles 2004; Gerla and Kleinrock 1980], most of the work presented in the NoC domain relies on the switch-to-switch flow control; this is primarily due to the large overhead incurred by the end-to-end flow control algorithms. A comparison of the fault-tolerance overhead of various flow control algorithms employed in NoCs can be found in [Pullini et al. 2005]. In that article, the authors consider buffer and channel bandwidth allocation in presence of pipelined switch-to-switch links and analyze varying degrees of fault tolerance support, resulting in different area and power trade-offs.

We note that, in real applications, the *best-effort* (or non–real time) and *guaranteed service* (or real-time) traffic may coexist. The Aethereal network

architecture presented in [Radulescu et al. 2005] employs the end-to-end flow control for guaranteed service in addition to the basic link-level control. More precisely, the authors implement guarantees using contention-free routing based on a time-division multiplexed approach and differentiate best-effort traffic from the guaranteed traffic. The end-to-end flow control uses credits that reflect the available buffering space in the destination. Similarly, the SPIN architecture in [Adriahantenaina and Greiner 2003] also uses credit-based flow control where buffer overflows at the target end of a path are checked at the source. The receiver notifies the sender of every datum consumed, with a dedicated feedback wire. [Bjerregaard and Sparso 2005] propose an asynchronous NoC router architecture to support connection-oriented guaranteed service, as well as best-effort routing. The guaranteed service operation is provided through use of multiple virtual channels, a nonblocking switch and a link access protocol, while the best-effort traffic uses credit-based flow control. Bolotin et al. [2004] present the QNoC network architecture based on a two-dimensional mesh topology and deterministic shortest path routing. In this architecture, the traffic is divided into four classes with different levels of priority. Then, a preemptive priority-based scheduling is proposed to provide QoS based on these traffic types. [Harmanci et al. 2004] also provide guaranteed services on top of best-effort traffic using prioritization of flows. A quantitative comparison between this connectionless scheme and a connection-oriented scheme [Radulescu et al. 2005] such as is presented in Harmanci et al. [2005]. The authors conclude that the connectionless scheme offers more stable end-to-end delay and it is able to provide guaranteed latency for individual flows. Nostrum NoC architecture [Millberg et al. 2004] has two dimensional mesh topology and employs an adaptive, deflection routing. In deflection routing, the incoming packet is routed to one of the free output channels belonging to a minimal path. If all the channels belonging to minimal paths are occupied, then the packet is misrouted. This increases message latency even in the absence of congestion and bandwidth consumption [Baydal et al. 2005; Duato et al. 2002; Hyatt and Agrawal 1997]. Moreover, when there are no available output channels, the entire packet needs to be stored; this requires buffers large enough to store the packets. Nostrum deals with this by fixing the packet size as 1-flit. However, this requires putting the header information such as destination address to each packet. Hence, this results in a large overhead and poor bandwidth utilization. Unlike the Nostrum architecture, we support packets with *arbitrary* length. We employ wormhole routing and deterministic shortest path routing algorithms. Finally, Nostrum handles both best-effort and guaranteed latency traffic. The guaranteed service is provided through virtual circuits implemented using looped containers and temporally disjoint network concepts which require a synchronous design (i.e., a common notion of time across the network). As opposed to this, our proposed technique targets best-effort traffic.

Flow control techniques mentioned in the previous paragraph target QoS guarantees, whereas our technique targets best-effort traffic and prevents congestion by regulating the best-effort traffic. Hence, our technique cannot be used to provide guaranteed services per se. Instead, it makes the best use of network bandwidth without sacrificing the bandwidth allocated to the guaranteed

service traffic. Therefore, when a mechanism for the guaranteed service traffic is in place, the proposed technique can be used in conjunction with this service to fully exploit the bandwidth not utilized by the guaranteed service traffic.

2.2 Interconnection Networks

When the network becomes congested, performance degradation in terms of both average message latency and accepted traffic is experienced. Congestion control is well studied for macronetworks [Bertsekas and Gallager 1992; Gerla and Kleinrock 1980; Paganini et al. 2001; Qiu and Shro 2004]. Paganini et al. [2001], 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 Qiu and Shro [2004], where 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.

Injection limitation techniques are studied in the context of parallel computer networks to avoid network saturation and cope with deadlock. Lopez et al. [1998] use the number of busy output channels in a node to measure the level of congestion. If the number of busy output channel exceeds a properly selected threshold value, the router prevents injection of new messages. Since this threshold is a function of the traffic pattern and packet sizes, the authors adjust it dynamically as a function of network load. Baydal et al. [2005] survey a family of mechanisms for congestion control in wormhole networks. In the first technique, congestion is measured as the ratio between the number of free virtual channels and total number of useful virtual channel that could be used by a certain message. If this ratio is larger than a threshold which should be tuned manually, then the packet is injected to the network. In second technique, packet injection is permitted if all physical channels have at least one virtual channel free or at least one physical channel has all its virtual channels free. Finally, the third method computes the number of flits sent through each virtual channel in a certain time interval to detect network congestion. If a channel is busy and the number of flits sent is less than a threshold, then the channel is considered congested. In case congestion is detected, packet injection restrictions are applied at the local node. The time interval and threshold need to be tuned, as in the first mechanism. These techniques rely on local feedback, hence they lack knowledge about global information. On the other hand, Smai and Thorelli [1998] present a global congestion control scheme based on time-outs. In this scheme, each node monitors the time the header flit stays in the source queue. If the waiting time is larger than a threshold, the node sends a congestion signal to its neighbors. All the nodes receiving the congestion message limit packet injection and share this information with their own neighbors. The technique presented in Thottethodi et al. [2001] aims at detecting congestion in early stages by taking the global conditions into account. The fraction of full virtual channel buffers of all routers is used as the congestion metric. The congestion data collected at each node is then disseminated to all other nodes through an exclusive side-band reserved for this purpose. The authors

develop an all-to-all communication mechanism for dissemination of congestion information with guaranteed delay bounds. However, this mechanism is specific to the particular network topology and its generalization to other topologies is not straightforward [Hedetniemi et al. 1998].

Our approach is different from the previously mentioned work in a number of ways. First, our technique is computationally light since it relies on *local* data transfers, similar to the basic switch-to-switch flow control. At the same time, our mathematical formulation enables us to predict the available buffering space in the network without assuming any particular traffic pattern or network topology. Due to the aggregation process performed at the routers, 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. In what follows, we present the details of our approach.

3. SYSTEM AND TRAFFIC SOURCE MODELING

3.1 System Model and Basic Assumptions

We assume the network nodes consist of processing and storage elements (referred to as PEs); the nodes communicate by exchanging packets across the network. We consider wormhole routing, so 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). No assumption is made about the underlying network topology.

In order to avoid packet loss, a basic link-level ON-OFF flow control mechanism is implemented at the routers [Dally and Towles 2004]. The proposed predictive control technique works together with this link-level mechanism to control directly the behavior of the traffic sources.

3.2 Traffic Source Model

Traffic injection rate into the network is the main knob for source control. Therefore, an accurate model of the input traffic is necessary for the development of flow controller. Such a model will not only show how the input traffic can be handled, 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 2.

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.

3.2.1 *Experimental Justification for the ON/OFF Traffic Model.* To support this observation with measured data, we collected traces from the sources in the MPEG-2 encoder design presented in Lee et al. [2007]. The MPEG-2

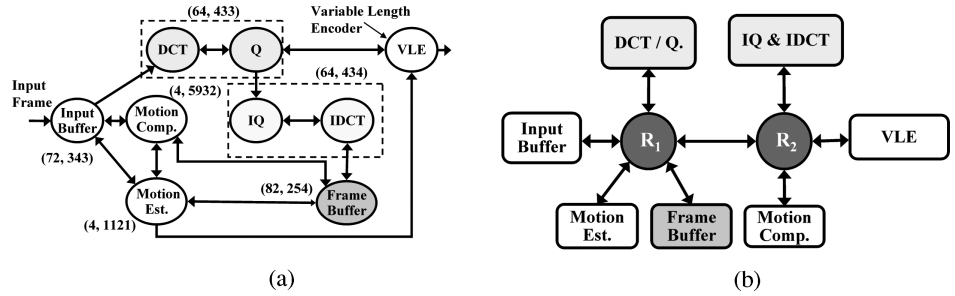


Fig. 1. (a) The data flow graph of the MPEG-2 encoder in Lee et al. [2007] and its (b) NoC-based implementation are shown. The average values of ON and OFF periods (F_{ON_ave}, F_{OFF_ave}), as number of cycles, are also shown in Figure 1(a).

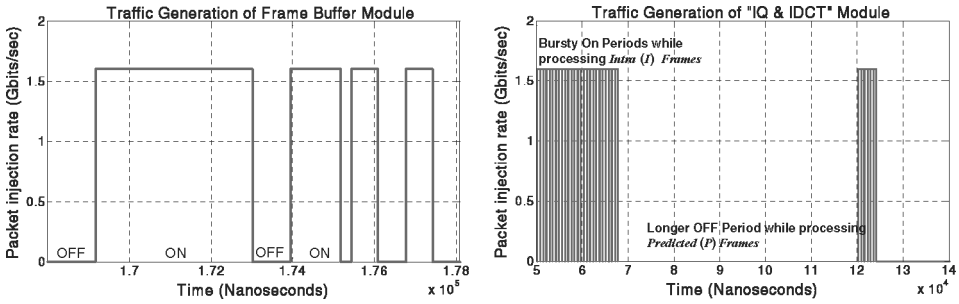


Fig. 2. (a) Traffic injection by the *Frame Buffer* module is shown. It can be observed that the module switches between ON and OFF periods. (b) Traffic injection by the *Inverse Discrete Cosine Transform*/*Inverse Quantization module* is plotted for a longer time scale. Bursty ON periods are followed by long OFF period due to long data waiting time.

encoder data flow graph and its NoC-based implementation are depicted in Figure 1. Figure 2(a) shows the traffic generated by the *Frame Buffer* module which stores the frames reconstructed using the previously encoded frames. The *Frame Buffer* module alternates between ON and OFF states. During the OFF state, it waits for a read request from the *Motion Compensation* module, as shown in Figure 1(a). Hence, no packet is generated during this period. Once a read request is received, the *Frame Buffer* module prepares a packet containing the next macroblock to be processed and injects the packet to the network; hence the module switches to the ON state. It stays in the ON state until all the requested data is transmitted to the *Motion Compensation* module.

In Figure 2(b), we investigate the traffic generated by the *Inverse Discrete Cosine Transform* (*IDCT*)/*Inverse Quantization* (*IQ*) *module* at a larger time scale. This module receives macroblocks from *Discrete Cosine Transform* (*DCT*)/*Quantization* (*Q*) module and performs inverse quantization and inverse discrete cosine transform to produce the reconstructed frame.

Since encoding the *Intra* (*I*) frames does not require motion estimation and compensation, the *IDCT*/*IQ* module receives and processes packets at a very high pace. Consequently, we observe bursty ON periods with small OFF periods

in between (time scale $5 \times 10^4 - 6.8 \times 10^4$ *nsec* in Figure 2(b)). On the other hand, encoding *Predicted* (P) frames is much slower in our design. For this reason, the *IDCT*/*IQ* module waits longer for new blocks to be processed and experience longer OFF periods.

We also note that the bursty nature of the on-chip traffic has been observed by other researchers. For instance, the traffic model considered in Murali et al. [2005] consists of bursts (i.e., ON periods) and silent (i.e., OFF) periods. Similarly, the long-range dependent (LRD) behavior of on-chip multimedia traffic is demonstrated and studied in Varatkar and Marculescu [2004]. It is a known fact that ON-OFF traffic sources with heavy tailed distribution of ON (or OFF) times gives rise to LRD traffic [Park and Willinger 2000].

3.2.2 *Characterization for the Distribution of ON/OFF Periods.* 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:

$$V(t) = V(t-1) + \lambda(t), \quad V(0) = 0, \quad t \in Z^+. \tag{1}$$

In the ON state, the flit injection rate $\lambda(t)$ is constant and equal to channel width; that is, $\lambda_{\mathrm{ON}} = W\ bits/sec$. 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 interarrival times are *not* memoryless, (i.e., *not* exponentially distributed) and so the flit injection process cannot be modeled as a Poisson process. Consequently, we employ the following classical ON/OFF [Park and Willinger 2000] model to work with NoC traffic sources.

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, $t_{ON} = \lceil SW/\lambda_{ON} \rceil$ where, again, S is the length of a packet (number of flits) and W is the physical channel width (W bits are transmitted per flit). 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 produce only 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:

$$F_{ON}(t) = p(t_{ON} \leq t) = \sum_{i=0}^{t} p\big(t_{ON} = i\big). \tag{2}$$

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

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:

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

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. When t_{proc} can take n different values, the distribution of t_{OFF} can be expressed as a function of the waiting time, $p(t_{wait} \leq t)$, as follows:

$$F_{OFF}(t) = \sum_{k=1}^{n} p(t_{wait} \leq t - t_k | (t_{proc} = t_k)) p(t_{proc} = t_k). \tag{3}$$

In general, it is difficult to compute the distribution of t_{wait}, since it depends on the latency experienced in the network. However, the predictive flow controller presented in this paper depends only on the distribution of the t_{ON}, as explained in Section 5.

3.3 Predictive Control of Traffic Sources

Suppose that the ON states of several traffic sources overlap and lead to temporary 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, 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 $(t_0, t_0 + \delta]$ 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

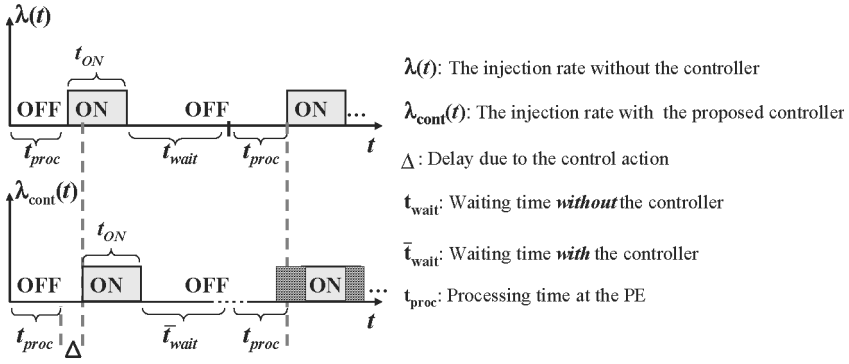


Fig. 3. Illustration of the ON-OFF source model and the control action. By delaying the start of the ON period, the waiting time in the network can be reduced.

improves, as illustrated in Figure 3. 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 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.

4. STATE SPACE MODELING OF NoC ROUTERS

To obtain accurate predictions for the available buffering space in the routers, we also need a good model for the NoC router. Traditionally, the network research has been focused on directly computing the router delay [Chien 1998; Peh and Dally 2001]. 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*.[2]

We propose a state space representation of a NoC router driven by stochastic inputs, as shown in Figure 4. The state of the router at time n is given by the number of flits in its input buffers; that is:

$$X(n) = [x_1(n), x_2(n), \ldots, x_p(n)]^T, \tag{4}$$

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:

$$\begin{aligned} U(n) &= [u_1(n), u_2(n), \ldots, u_p(n)]^T, \\ Y(n) &= [y_1(n), y_2(n), \ldots, y_p(n)]^T. \end{aligned} \tag{5}$$

[2]A similar model for the output buffers can be also developed.

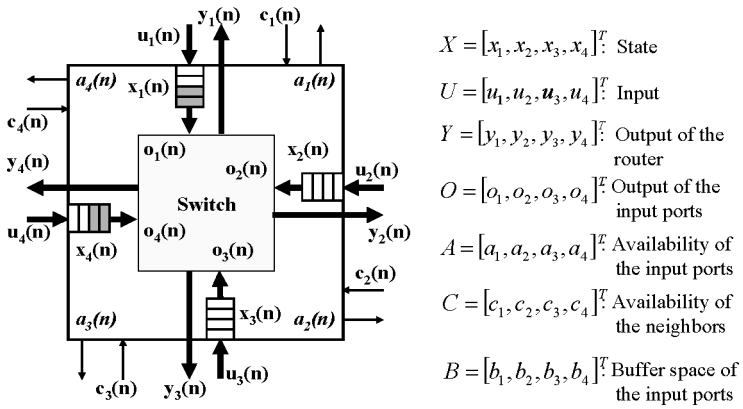


Fig. 4. The state variables, inputs and outputs of a 4-port router are shown.

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 4).

As a result, the knowledge of either $Y(n)$ or $O(n)$ provides information about the other, given the connections in the crossbar switch. So, 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:

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

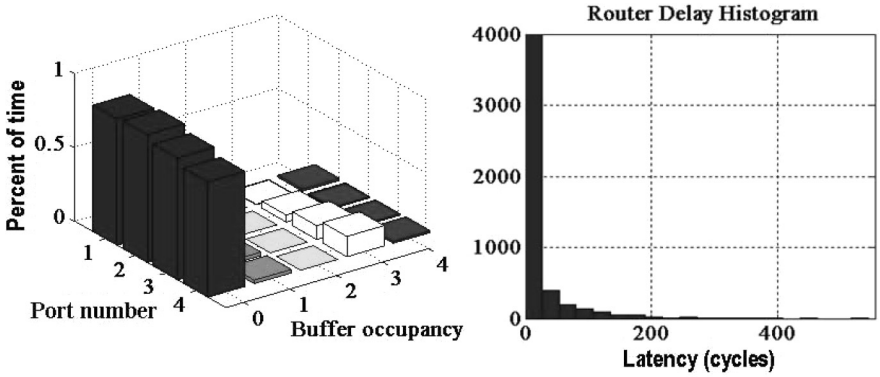
Router Stability. The router described by Equation (6) can become unstable (i.e., the state grows unbounded), if the average arrival rate to the router is greater than the rate at which the router can serve any given packet. In practice, however, the input buffers are all finite. Hence, in order to avoid 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:

$$X(n+1) = I_{P\times P}X(n) + [U(n)H(n)] - O(n), \tag{7}$$

where $H(n) = [h(b_1 - x_1(n)), h(b_2 - x_2(n)), \ldots, h(b_P - x_P(n))]^T$.$h(x_i)$ is the unit step function (*i.e.*, $h(x_i) = 0$ if $x_i \leq 0$, and $h(x_i) = 1$ otherwise), and b_1 to b_P represent the capacity of each input buffer. We also emphasize that $[U(n)H(n)]$ represents the *element-wise* product in this equation; it is used hereafter for notational simplicity.

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

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



(a) (b)

Fig. 5. (a) Buffer utilization and (b) delay histogram of a router.

Obviously, the router described by Equation (8) has a bounded response. However, since such a control does not limit the source injection directly, the input buffers will remain full for most of the time, if the average arrival rate becomes larger than the service rate of the router. This, in turn, results in blocked links and large delays in the network. One can regulate the traffic injection by an open loop controller [Dally and Towles 2004]. However, this solution does not solve the congestion problem completely, since the packets may experience congestion due to the overlaps between the ON periods of the traffic sources even under a light load. For instance, consider a 4×4 2D mesh network running hotspot traffic.[3] 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 (see Figure 5(a)), about 18% of the packets experience delays more than twice as large as the average delay, as shown by the delay histogram in Figure 5(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 so, in what follows, we show how exactly the router model presented in this section can be used to implement a predictive flow controller which regulates the traffic injection to the network.

5. PREDICTION-BASED FLOW CONTROLLER

Collecting congestion data at the routers and delivering this data to the traffic sources for flow control may cause a large communication overhead; so it is not a scalable approach. Moreover, 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 3 and 4, respectively. These models enable us to predict the availability of any router at a future time step, as described next.

[3]Under the hotspot traffic, the nodes in the network receive packets with uniform probability, except a few (in our experiments 4) randomly selected nodes that receive some extra traffic.

5.1 Availability Predictor

We use the conditional expectation of the state at n_0+k, given the state at time n_0, *i.e.*, $\hat{X}(n_0+k|n_0)$, as the *k-step* predictor for network state [Mendel 1995].

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

Using Equation (8), we have:

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

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 macronetwork 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 as the *availability* of port P and denote it by $a_P(n_0, k)$. $a_P(n_0, k)$ simply consists of the (I) sum of the number of empty slots in the buffer at time n_0+k, and (II) the number of flits that are expected to be admitted in the following k steps, that is,

$$\underset{(I)}{a_P(n_0,k) = b_P - \hat{x}_P(n_0+k|n_0)} + \underset{(II)}{\sum_{j=n_0}^{n_0+k-1} E[u_P(j)h(b_P - x_P(j))|n_0]}.$$

If we define the availability vector as $A(n_0, k) = [a_1(n_0, k), \ldots, a_P(n_0, k)]$ and $B = [b_1, b_2, \ldots, b_P,]^T$ is the vector containing the depth for each input buffer, then we can find $A(n_0, k)$ as:

$$A(n_0,k) = B - \hat{X}(n_0+k|n_0) + \sum_{j=n_0}^{n_0+k-1} E[[U(j)H(j)]|n_0]. \tag{10}$$

Next, we can substitute $\hat{X}(n_0+k|n_0)$ in Equation (9) to Equation (10) and obtain $A(n_0, k)$ as:

$$A(n_0,k) = B - X(n_0) + \sum_{j=n_0}^{n_0+k-1} E[O(j)|n_0]. \tag{11}$$

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 in the interval $[(n_0, n_0+k)$. 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 $[(n_0, n_0+k)$.

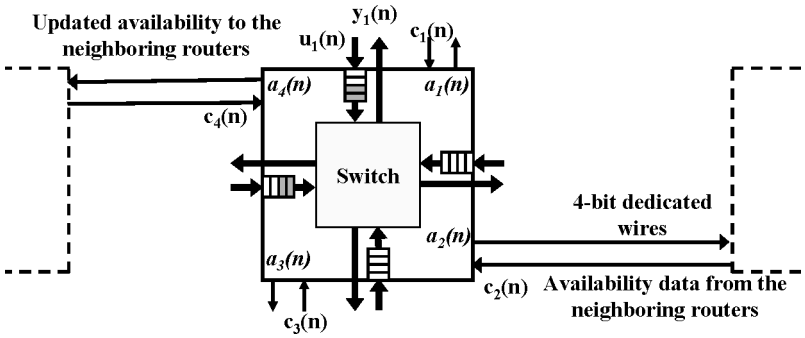


Fig. 6. Exchange of the availability information between the neighboring routers.

The expected value of the read process from the input buffers (that is, the last term in Equation (11)) can be approximated using the router output $Y(j)$ as follows:

$$\sum_{j=n_0}^{n_0+k-1} E[O(j)|n_0] = \begin{bmatrix} g_{1,1}(n_0) & \dots & g_{1,P}(n_0) \\ g_{2,1}(n_0) & \dots & g_{2,P}(n_0) \\ \dots & \dots & \dots \\ g_{p,1}(n_0) & \dots & g_{P,P}(n_0) \end{bmatrix} \sum_{j=n_0}^{n_0+k-1} E[Y(j)|n_0] \qquad (12)$$

where the coefficients $g_{i,k}(n_0)$ reflect the state of the crossbar switch and channel allocation in the router. Computation of these coefficients are illustrated in Section 5.2 using a concrete example. If we let $G(n_0) = \{g_{i,k}(n_0)\}$, then Equation (11) can be written as:

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

Note that $\Sigma_{j=n_0}^{n_0+k-1}(E[Y(j)|n_0]$ is the expected number of flits transmitted by the router in the interval $[n_0, n_0 + k)$. However, this represents 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:

$$A(n_0, k) = B - X(n_0) + G(n_0)C(n_0 - 1, k), \qquad (13)$$

where the vector $C(n_0-1, k)$ denotes the availability of the immediate neighbors predicted at time $n_0 - 1$, as illustrated in Figure 6 (see also Figure 4).

In summary, the availability of the routers for the interval $[(n_0, n_0 + k)$ are predicted using the empty buffer slots at time n_0, the state of the crossbar switch and the availability of the neighboring routers using Equation (13). Hence, the computations depend on only local information. At the same time, the availability information computed at time n is obtained by aggregating the availability of the immediate neighbors at time $n-1$. This information, in turn, reflects the state of the routers situated two hops away, at time $n-2$, and so on

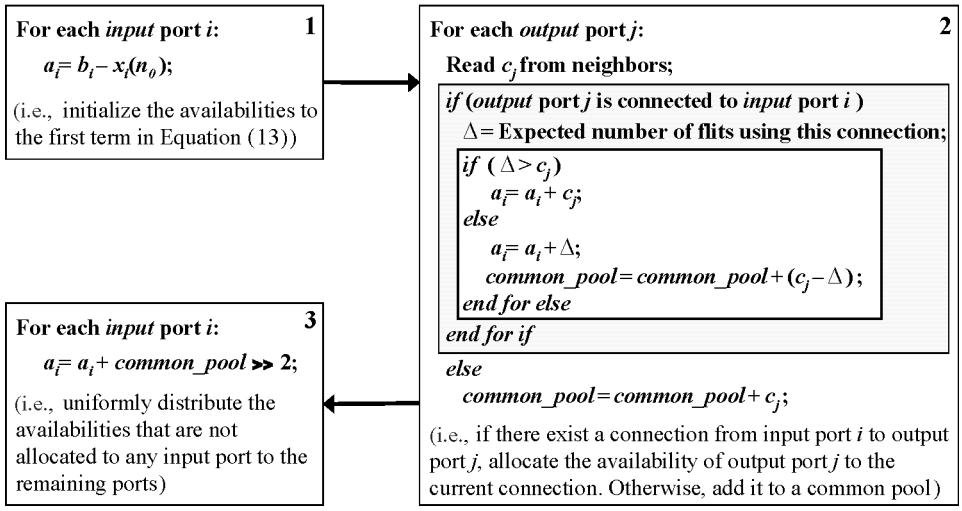


Fig. 7. Practical implementation of the predictor.

so forth. Therefore, due to the aggregation process the *local* predictions actually reflect the global view of the network.

5.2 Practical Implementation of the Predictor

Since all the input buffers in the network are initially empty, the availability values are initialized to the sum of buffer capacities and the prediction step, that is,

$$A(0,k) = B + k[1,1,\ldots,1]^T_{1\times P}. \tag{14}$$

According to Equation (13), a router needs the number of free slots in its input buffers ($B - X(n_0)$), the state of the crossbar switch ($G(n_0)$), and the availabilities from the neighboring routers ($C(n_0 - 1, k)$) to update its availability. The routers keep track of the number of free slots in the input buffers and the state of the crossbar switch internally. On the other hand, they receive the availabilities of the neighboring routers through dedicated control wires, as depicted in Figure 6. The number of these dedicated wires determine the maximum availability value that can be transferred between the neighboring values. For instance, in our implementation we use 4 parallel wires, as shown in Figure 6; this means that we can transfer 4 bits of information over these wires. Hence, the maximum availability value that can be transferred is $2^4 - 1 = 15$.

Once a router receives the availabilities from the immediate neighbors through dedicated connections, it needs to determine how to distribute these availability values to its input ports. This distribution is achieved according to Equation (13), where the coefficients $g_{i,k}$ reflect the state of the crossbar switch. The details of this distribution process are provided in Figure 7.

The first step towards computing the availabilities is to initialize the availability of each input port with the number of free slots in the corresponding input buffer (i.e., the first term in Equation (13)), as shown by the box labeled

with "1" in Figure 7. We note that this term gives the zero-order approximation of the availabilities, when no input is received from the neighbors. After this initialization, the availability of the neighboring routers are processed as described in the box labelled with "2" in Figure 7. For each output port j, the predictor checks whether there exist a connection through the crossbar switch between that port and any input port i. If there is a connection, then the number of flits that are expected to use this connection (Δ in box 2, in Figure 7) is determined using the packet length available in the header flit and the number of flits that are already transmitted. If Δ is greater than the availability of output port j(i.e., c_j), then c_j flits are allocated to input port i. Otherwise, Δ flits are allocated to input port i, while the remaining (i.e., $c_j - \Delta$) is distributed uniformly to all input ports except port j. Port j is excluded, since a packet cannot leave the router using the same port it arrived, that is, 180 degree turns are not possible due to shortest path routing. In case the output port j is not connected to any input port, then the whole availability c_j is distributed uniformly to all input ports except port j, as described by the outer *if* statement in box 2, in Figure 7.

Example 1. Assume that at time n_0, the depth of the input buffers, their occupancies and the availability values received from the neighboring routers are given as follows:

$$B = \begin{matrix} \textit{Local} \\ \textit{North} \\ \textit{West} \\ \textit{South} \\ \textit{East} \end{matrix} \begin{bmatrix} 4 \\ 4 \\ 4 \\ 4 \\ 4 \end{bmatrix}, \quad X(n_0) = \begin{bmatrix} 0 \\ 4 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad C(n_0) = \begin{bmatrix} 0 \\ 8 \\ 3 \\ 8 \\ 8 \end{bmatrix}.$$

We further assume that the crossbar switch connects the *North* input port to the *West* output port, and 4 flits in the *North* input port are waiting to traverse the crossbar switch (i.e., $\Delta = 4$).

Computation of Availabilities. We explain the computation of availabilities in plain english according to the pseudo code in Figure 7. This description also shows how the algorithm is implemented.

We start by initializing the availabilities to $B - X(n_0)$; that is, the number of empty slots at time n_0 : $A = [4, 0, 4, 4, 4]^T$.

Next, we distribute the availabilities of the neighboring routers to the input buffers. Since the *West* output port is connected to the *North* input port and $\Delta > 3$, the availability of the *West* output port is added to the *North* input port. Hence, the availability vector becomes: $A = [4, 3, 4, 4, 4]^T$.

No input port is connected to the *North*, *South*, and *East* output ports. Therefore, their availabilities are distributed uniformly to all input ports. For example, after the 8-flit availability from the *North* output port is distributed to *Local*, *West, South*, and *East* input ports, the availability vector becomes: $A = [6, 3, 6, 6, 6]^T$.

The computation of the availabilities can be written in matrix notation (similar to Equation (13)), as follows:

$$A = \begin{bmatrix} 4 \\ 4 \\ 4 \\ 4 \\ 4 \end{bmatrix} - \begin{bmatrix} 0 \\ 4 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 8/4 & 0 & 8/4 & 8/4 & 8/4 \\ 0 & 3 & 0 & 0 & 0 \\ 8/4 & 8/4 & 8/4 & 0 & 8/4 \\ 8/4 & 8/4 & 8/4 & 8/4 & 0 \end{bmatrix}. \tag{15}$$

The first two component correspond to the initialization step; that is, $B - X(n_0)$. In the last term, each row describes how the availability from each output port is distributed. For example, the first row is all zeros, since the first row of $C(n_0)$ is zero. On the other hand, the 8-flit availability from the *North* port is distributed uniformly to *Local, West, South*, and *East* input ports, as described by the second row. Likewise, the fourth and fifth rows (which denote the availabilities from *South* and *East* ports) are uniformly distributed. However, all availabilities from the *West* output port go to the *North* input port, as described by the third row. We note that the entries in each row sum up to the availabilities obtained from the neighbors. Finally, Equation (15) can be rewritten by factoring $C(n_0)$ out such that it takes the form of Equation (13):

$$\begin{array}{ccccc} A = & \begin{bmatrix} 4 \\ 0 \\ 4 \\ 4 \\ 4 \end{bmatrix} & + & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1/4 & 0 & 1/4 & 1/4 & 1/4 \\ 0 & 1 & 0 & 0 & 0 \\ 1/4 & 1/4 & 1/4 & 0 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 & 0 \end{bmatrix}^T & \begin{bmatrix} 0 \\ 8 \\ 3 \\ 8 \\ 8 \end{bmatrix} \\ B - X(n_0) & & + & G(n_0) & C(n_0 - 1, k). \end{array}$$

Hence, the components of the matrix $G(g_{ij})$ in Equation (13) reflect the distribution process as demonstrated in this example.

5.3 Hardware Implementation of the Proposed Flow Controller

In order to accurately evaluate the area overhead, we implemented the proposed flow control in Verilog HDL and synthesized the design using Synopsys Design Compiler. The equivalent gate count of the proposed flow controller is found as 1093 gates.

We also integrated the proposed flow controller into an existing router and developed an FPGA prototype based on a Xilinx XC2V3000 platform. The basic NoC router without the proposed flow controller is based on the prototype presented in [Ogras and Marculescu 2006]. The router implements the basic link-level ON/OFF flow control. The input buffers of the router have 16 flit depth and 16 bit width. The router implements wormhole flow control with deterministic table-based routing. It takes 4 cycles for the router to process the header flit (that is, to receive, make a routing decision, traverse the crossbar switch and place it to the desired outgoing link). After that, the remaining flits simply follow the header in a pipelined fashion. The time it takes to route

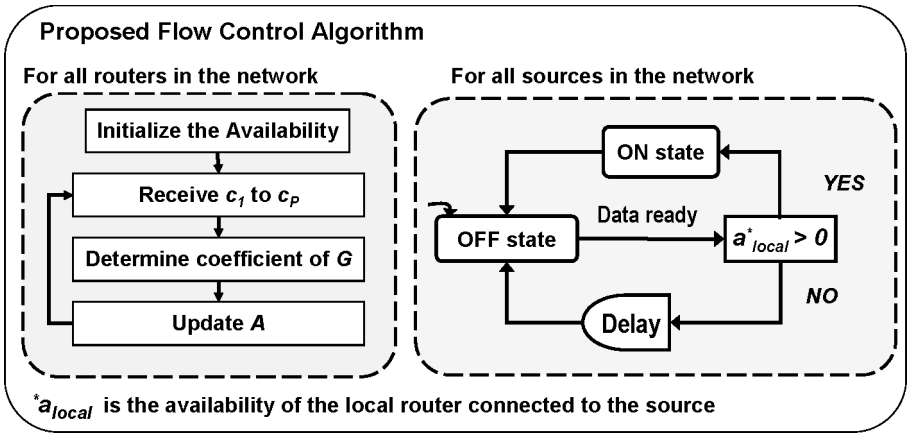


Fig. 8. Operation of the proposed flow control algorithm is illustrated.

packets is not affected by the proposed controller, since the computation of the availabilities are performed concurrently with routing.

The proposed controller takes up 80 slices in the target FPGA; this corresponds to about 18% increase in the number of resources used by the router. It is also important to evaluate the overhead of the router in a real design. For instance, the overhead of the proposed controller is about 0.8% for the MPEG-2 encoder presented in Lee et al. [2007]. In general, the overhead of our controller is estimated be about 1% of the total chip area.

5.4 Using Prediction for Network Control

The overall operation of the proposed flow controller in summarized in Figure 8. Each router in the network updates its availability periodically by aggregating the data received from the immediate neighbors. As a result, the availability of a local input port connected to a traffic source reflects the backpressure from all of the downstream routers. In the absence of the proposed flow controller, a traffic source switches freely from OFF state to ON state, whenever it needs to inject a packet to the network. As opposed to this, the traffic sources check the availability of its host router, before entering the ON state. In analogy with wireless networks, the sources *listen before transmit*, that is, they sense the congestion in the network through the local router. As such, when a traffic source sees that the input port connected to it has zero availability, it delays the generation of new packets until the availability of the port becomes greater than zero, as shown in Figure 8.

Since the congestion information propagates in the network through aggregation, prediction step k is selected as the diameter of the network. In this way, the timely transmission of the prediction to the traffic sources is guaranteed, 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 and the availability signals do not experience queuing delays.

Table I. The Reduction in the Average Packet Latency and Number of Packets in the Network Due to the Proposed Flow Control Algorithm are Indicated. The Latency Values are the Sum of the Latencies Experienced at the Source Queue and in the Network. The Average Latencies Experienced at the Source Queues are Also Given (in Parentheses)

	Switch-to-Switch Control Only	The Proposed Flow Control	Reduction (×)
Ave. latency	149 (112) *cycles*	47 (12) *cycles*	3.2
Max. latency	897 (774) *cycles*	466 (404) *cycles*	1.9
Standard deviation of latency	173.8 (156) *cycles*	55.7 (46) *cycles*	3.1
Ave. # of packets	94 *packets*	29 *packets*	3.2
Max. # of packets	129 *packets*	52 *packets*	2.5
Standard deviation of # of packets	24.7 *packets*	7.2 *packets*	3.4

6. EXPERIMENTAL RESULTS

In this section, 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. Wormhole routing and deterministic XY routing algorithm is used throughout the simulations. 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[4]. The average packet latency reported in this paper includes the latency experienced in the local memory (i.e., the source queuing delay), and the network latency. The network latency denotes the time the packet travels in the network before being ejected. Finally, we also present experimental results obtained using the FPGA prototype in addition to the simulations.

6.1 Audio/Video System

We first used the audio/video system described in Hu and Marculescu [2005] to evaluate the potential of the proposed algorithm for real applications. The target system includes an H263 video encoder, an H263 video decoder, an MP3 audio encoder, and an MP3 audio decoder. It is partitioned into 40 concurrent tasks and then these tasks are mapped to the 4×4 2D mesh-network. Finally, the traffic traces obtained from real video and audio clips are used to determine the communication patterns among the processing cores in the network.

The audio/video system 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 in place. As shown in Table I, the average packet latency becomes 47 *cycles*, while the maximum packet latency

[4]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.

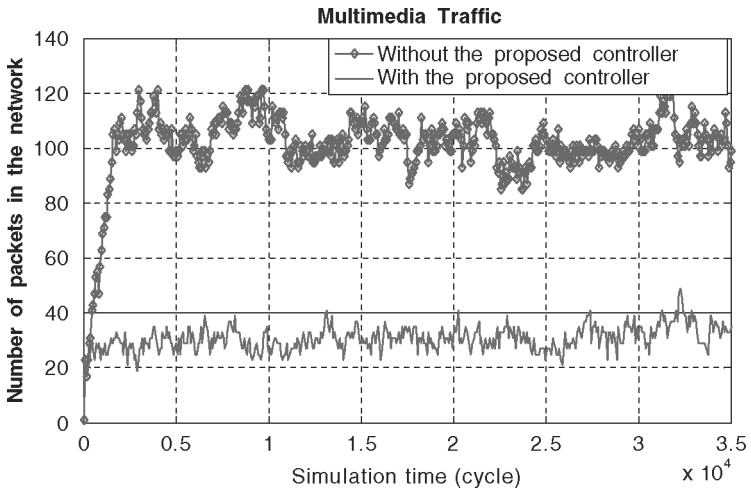


Fig. 9. Variation of the number of packets in the network over time for multimedia traffic.

drops to 466 *cycles*. Table I summarizes also the queueing delay experienced at the traffic sources (shown separately inside parentheses). We observe that the average source queuing delay reduces from 112 *cycles* to 12 *cycles* with the use of the proposed controller. Similarly, the maximum value of the source queuing delay and the standard deviation drops significantly as a result of the proposed controller.

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 average number of packets in the network drops from 94 to 29 *packets*, which is about a 3.2× reduction, as summarized in Table I. Likewise, the maximum number of packets in the network and the standard deviation of the number of packets in the network drop by 2.5× and 3.4×, respectively.

We further investigate the number of packets travelling through the network as a function of time in Figure 9. Without the proposed flow controller, the number of packets quickly rises to about 100 packets and oscillates around the average value (94 *packets*) with a standard deviation of 24.7 *packets*. On the other hand, the proposed controller provides about 3.2× reduction in the average number of packets and 3.4× reduction in standard deviation, as summarized in Table I and plotted in Figure 9. The variation in the packet latency and number of packets observed in the absence of the proposed controller show that the network can oscillate between congested and free traffic due to the overlap in the ON periods of traffic sources, as discussed in Section 3.3.

To better understand the effects of the controller, in Figure 10, we further analyze the histogram of the packet latencies. We notice that, for the network without the proposed flow controller, about 50% 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 10(a) are the main cause for this poor

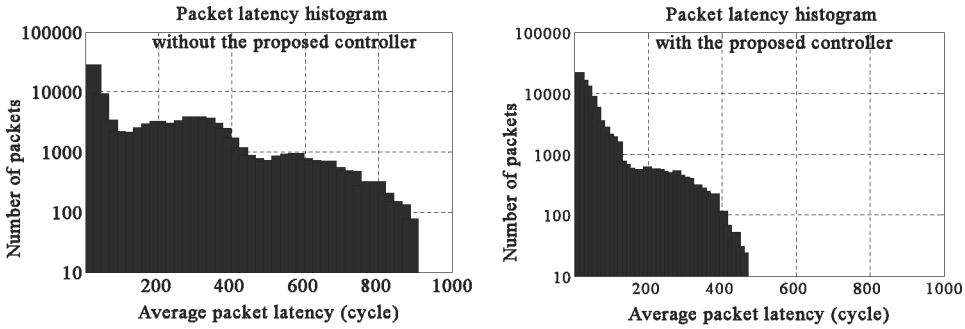


Fig. 10. Histogram of the packet latencies for the Audio/Video traffic without (a) and with (b) the proposed flow controller. Note that the y-axes of the plots are in log-scale. A significant improvement due to the flow-control mechanism can be observed.

performance. The technique we propose prevents the packets that are likely to experience such long delays from entering the network. Indeed, as depicted in Figure 10(b), the latency histogram is pushed significantly towards left; so about 91% of packets experience less than 100 *cycles* latency. We observe that there are no packets with latency more than 466 cycles, if the proposed controller is used. Moreover, the number of packets with latency more than 100 cycles drops quickly due to the proposed controller. On the other hand, half of the packets experience latencies longer than 100 cycles in the absence of the proposed controller. Besides this, there are packets with as high as 900 cycles latency.

Since the audio/video application we consider includes strong access locality, the average hop count for this application on the 4×4 mesh network is only 1.98, while the average hop count for uniform traffic would be 2.67. For applications with less access locality, the number of packet contentions; therefore, the improvement due to the proposed controller is expected to be larger.

6.2 Synthetic Traffic

Additional experiments are presented for uniform and hotspot traffic patterns to further assess the effectiveness of the proposed controller. 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 11(a). We observe that without the flow controller, the network becomes congested as the packet injection rate increases. The reason for this behavior is uncovered in Figure 11(b). Indeed, in absence of a traffic controller, the number of packets in the network grows at an increasing pace as the traffic injection rate increases. The proposed flow controller, on the other hand, effectively limits the number of packets injected to the network, as depicted in Figure 11(b). This, in turn, results in significant improvements in the average packet latency. Finally, Figures 11(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.

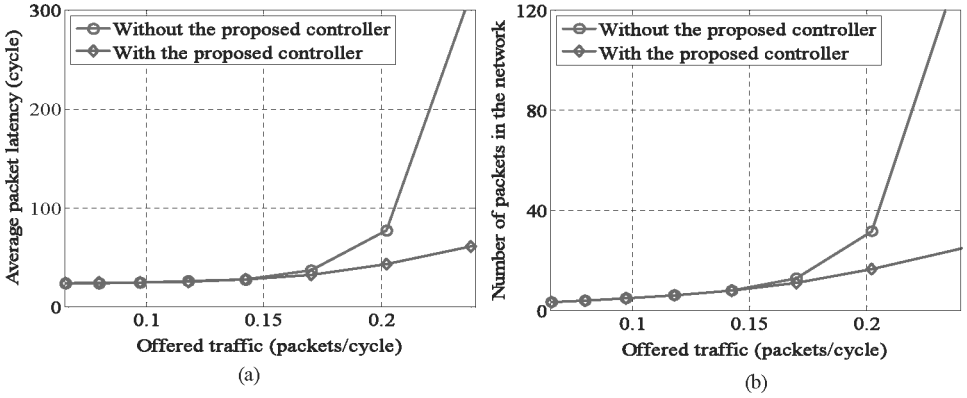


Fig. 11. (a) Average packet latency and (b) average number of packets in the network are plotted as a function of the offered traffic.

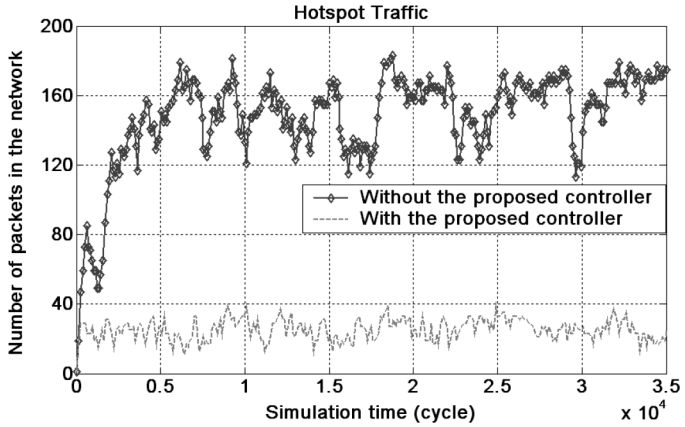


Fig. 12. Variation of the number of packets in the network over time for the hotspot traffic. The proposed controller reduce the average number of packet in the network and their variation over time significantly.

Similar to the multimedia traffic, we monitored the number of packets in the network when the packet injection rate was about half of the maximum throughput. The mean and variance of the number of packets in the network is significantly reduced with the proposed controller, as depicted in Figure 12. More specifically, the average number of packets in the network drops from 151 to 25 packets resulting in 6× reduction, while the maximum number of packets in transit decreases from 189 to 45 packets. Furthermore the standard deviation of the number of packets is reduced from 24.6 to 6.2, as summarized in Table II.

6.2.1 *Impact of the Local Buffer Size on Performance.* The PEs write the packets that will be transmitted over the network to a local memory in the network interface. Then the local router reads the packets from this memory. In general the buffering space available in the local PEs are much larger than

Table II. The Reduction in the Latency and Number of Packets in the Network Due to the Proposed Flow Control Algorithm

	Switch-to-Switch Control Only	The Proposed Flow Control	Reduction (×)
Ave. # of packets	151 *packets*	25 *packets*	6.0
Max. # of packets	189 *packets*	45 *packets*	4.2
Standard deviation of # of packets	24.6 *packets*	6.2 *packets*	4.0

Table III. Average Packet Latency for the *hotspot* Traffic (at 0.2 *packets*/*cycle* traffic rate) Without and with the Proposed Controller for Different Local Memory Sizes are Summarized

Local PE Buffer Size	50-flit		100-flit		200-flit		500-flit	
	Ave. Latency	Ave. PE Pausing	Ave. Latency	Ave. PE Pausing	Ave. Latency	Ave. PE Pausing	Ave. Latency	Ave. PE Pausing
Without the proposed controller	80 *cycles*	0.75 *cycles*	106 *cycles*	0.43 *cycles*	158 *cycles*	0.31 *cycles*	217 *cycles*	0.14 *cycles*
With the proposed controller	43 cycles	1.93 *cycles*	44 *cycles*	1.94 *cycles*	44 *cycles*	1.92 *cycles*	44 *cycles*	1.96 *cycles*

those on the routers. Nevertheless, local buffering space has also a finite value and may fill up as a result of the backpressure from the network. When the local memory at the PE becomes full, the PE cannot write to this memory. Since no packets are dropped, the PE pauses in this situation.

In this section, we compare the performance of the proposed predictive controller and switch-to-switch control for various buffer sizes in the local PE. For the switch-to-switch control, the average packet latency at a given traffic injection rate increases considerably with increasing local buffer size, as summarized in the first row of Table III. Switch-to-switch control is less effective for large local buffers, since a large number of packets can be generated before the PE feels the backpressure. In Table III, we also give the average number of cycles the PEs pause. We observe that the PEs pause on average 0.75 *cycles*, when the local memory size is 50 flits. The PEs pause very rarely, since the network is not heavily congested. We also observe that this number is reduced as the local memory size is increasing. This is also expected, since the PEs are affected by the backpressure less for large local memory.

In contrast to the switch-to-switch control, the proposed flow controller uses directly the availability information predicted by routers as a measure of congestion level. Therefore, the traffic sources do not rely on backpressure from the network. Consequently, the proposed flow controller performs well over a wide range of local PE sizes, as shown in the last row of Table III. Table III also shows that the proposed controller causes PEs pause more often than the switch-to-switch control. This is also expected, since the controller delays the packet generation. However, we observe that the average time the PEs pause is about 1.9 *cycles*, as shown in Table III. Hence, the sum of the latency and pausing time is still much smaller when using the proposed controller.

6.2.2 *Scalability of the Proposed Approach.* The computation of the availabilities at the routers depend only on the local information received from the immediate neighbors. Therefore, the computational complexity

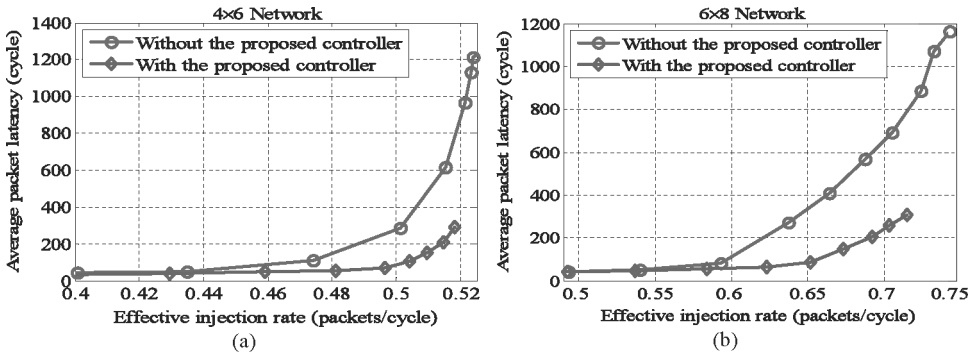


Fig. 13. (a) Average packet latency as a function of effective traffic injection rate is plotted for (a) 4×6 and (b) 6×8 networks under *hotspot* traffic pattern.

depends only on the number of ports in the router, *not* on the size of the network.

In order to demonstrate the performance of the proposed technique for larger network sizes and provide a more quantitative evaluation, we performed new experiments involving 4×6 and 6×8 networks. Figure 13(a) and Figure 13(b) show the average packet latency as a function of the effective traffic injection rate for the 4×6 and 6×8 networks, respectively. We note that, the x-axes of these figures show the effective traffic injection rates, which take the PE pausing (discussed in Section 6.2.1) into account. Since the proposed flow controller limits the number of packets in the network to prevent congestion, the total traffic injection rate is limited, as shown in Figure 13. In particular, the maximum effective injection rate is 0.51 packets/cycle for the 4×6 NoC and 0.72 packets/cycle for the 6×8 NoC. On the other hand, without the proposed controller, the effective traffic injection rate continues to increase even in the presence of congestion. Hence the average packet latency grows significantly. As it can be clearly seen from Figure 13, due to the proposed controller, the average latency improves especially at larger injection rates.

6.3 Experimental Evaluation with the FPGA Prototype

We also present results directly measured using the FPGA prototype to support the simulation results. For this purpose, a 4×4 Mesh network is implemented with and without the proposed flow controller. Similar to the simulations, a basic link level ON/OFF flow control is implemented in both cases, and deterministic XY routing is employed.

The reduction in the mean and standard deviation of the packet latencies in the network are similar to those obtained using simulation. Table IV summarizes measured values for the mean and standard deviation for a wide range of traffic injection rates. For instance, at 0.016 *packets*/*cycle* traffic injection rate,the average packet latency without the proposed controller is found as 32.6 *cycles*. The proposed controller reduces the average latency to only 18.2 *cycles* including the waiting time in the processing element, as shown in Table IV. Even more importantly, *without* the flow-controller, the standard deviation of packet latencies over multiple simulations is as large as 17.6 *cycles;* this is

Table IV. Mean and Standard Deviation of Packet Latencies Obtained Using the FPGA Prototype are Shown with and without the Proposed Controller

Applied Traffic(packets/cycle)		0.016	0.031	0.062	0.0126
Average latency(*cycles*)	*w/o the proposed controller*	32.6	46.1	94.8	169.4
	with the proposed controller	18.2	21.2	28.7	45.9
	Reduction (×)	1.8	2.2	3.3	3.7
Standard deviation of latency (*cycles*)	*w/o the proposed controller*	17.6	18.2	23.6	37.1
	with the proposed controller	0.15	8.1	13.2	14.7
	Reduction (×)	119	2.3	1.8	2.5

due to the lack of capability of the link level flow control to regulate number of packets in the network. On the other hand, the standard deviation with the proposed flow controller is less than one cycle. Even though this value increases for larger applied traffic, we see about 2× improvement over the basic link level controller over a wide range of input traffic.

To summarize, the measurements obtained using the FPGA prototype are in perfect agreement with the simulation results presented in Sections 6.1 and 6.2.

7. CONCLUSIONS

Effective flow control mechanisms are necessary for efficient utilization of network resources in the context of NoCs. However, neither switch-to-switch, nor end-to-end control schemes proposed so far for macro-networks can satisfy the requirements of NoCs. For this reason, we have 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, similar to the end-to-end flow controller. At the same time, our approach relies only on local information transfer between the neighboring routers only; therefore, it has a low communication overhead, similar to the switch-to-switch flow control.

Our experimental results show that the proposed controller effectively regulates the number of packet in the network and delivers much better performance compared to traditional switch-to-switch flow control algorithms. The improvement in the average latency, latency distribution, as well as the maximum value and standard deviation are demonstrated by extensive simulations involving both real and synthetic benchmarks. Furthermore, direct measurements obtained using the FPGA prototype we implemented fully support the simulation results. Finally, the FPGA prototype shows that the area overhead of the proposed controller is less than 1% when incorporated into an existing MPEG-2 encoder design.

REFERENCES

ADRIAHANTENAINA, A. AND GREINER, A. 2003. Micro-network for SoC: Implementation of a 32-Port SPIN network. In *Proceedings of the Design Automation and Test in Europe Conference*.

ASCIA, G., CATANIA, V., AND PALESI, M. 2004. Multi-objective mapping for mesh-based NoC architectures. In *Proceedings of International Conference on Hardware/Software Codesign and System Synthesis*.

BAYDAL, E., LOPEZ, P., AND DUATO, J. 2005. A family of mechanisms for congestion control in wormhole networks, *IEEE Trans. Paral. Distrib. Syst. 16*, 9.

BENINI, L. AND DE MICHELI, G. 2002. Networks on chips: A new SoC paradigm. *IEEE Comput. 35*, 1.

BERTSEKAS, D. AND GALLAGER, R. 1992. *Data Networks*. Prentice Hall, Englewood Cliffs, NJ.

BJERREGAARD, T. AND SPARSO, J. 2005. A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In *Proceedings of the Design Automation and Test in Europe Conference*.

BOLOTIN, E., CIDON, I., GINOSAR, R., AND KOLODNY, A. 2004. Cost considerations in network on chip. *Integr. VLSI J. 38*, 1.

BOLOTIN, E., CIDON, I., GINOSAR, R., AND KOLODNY. 2004. QNoC: QoS architecture and design process for network on chip. *J. Syst. Archite.: EUROMICRO J. 50*, 2–3.

CHIEN, A. A. 1998. A cost and speed model for k-ary n-cube wormhole routers. *IEEE Trans. Paral. Distrib. Syst. 9*, 2.

DALLY, W. J. 1992, Virtual-channel flow control. *IEEE Trans. Paral. Distrib. Syst. 3*, 2.

DALLY, W. J. AND TOWLES, B. 2001. Route packets, not wires: On-chip interconnection networks. In *Proceedings of Design Automation Conference*.

DALLY, W. J. AND TOWLES, B. 2004. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, San Francisco, CA.

DIELISSEN, J., RADULESCU, A., GOOSSENS, K., AND RIJPKEMA, E. 2003. Concepts and implementation of the Philips network-on-chip. In *Proceedings of the IP-Based SoC Design*.

DUATO, J., YALAMANCHILI, S., AND NI, L. M. 2002. *Interconnection Networks: An Engineering Approach*. *Morgan Kaufmann*, San Francisco, CA.

LOPEZ, P., MARTINEZ, J. M., AND DUATO, J. 1998. DRIL: Dynamically reduced message injection limitation mechanism for wormhole networks. *In Proceedings of the International Conference on Parallel Processing*.

GERLA, M. AND KLEINROCK, L. 1980. Flow control: A comparative survey. *IEEE Trans. Commun. 28*, 4.

GUERRIER, P. AND GREINER, A. 2000. A generic architecture for on-chip packet switched interconnections. In *Proceedings of the Design Automation and Test in Europe Conference*.

JALABERT, M. S., BENINI, L., AND DE MICHELI, G. 2004. XpipesCompiler: A tool for instantiating application specific networks on chip. In *Proceedings of the Design Automation and Test in Europe Conference*.

HARMANCI, M., ESCUDERO, N., LEBLEBICI, Y., AND IENNE, P. 2004. Providing QoS to connection-less packet-switched NoC by implementing DiffServ functionalities. In *Proceedings of the International Symposium on System-on-Chip*.

HARMANCI, M., ESCUDERO, N., LEBLEBICI, Y., AND IENNE, P. 2005. Quantitative modeling and comparison of communication schemes to guarantee quality-of-service in networks-on-chip. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*.

HEDETNIEMI, S. M., HEDETNIEMI, S. T., AND LIESTMAN, A. L. 1988. A survey of gossiping and broadcasting in communication networks. *Networks 18*, 4.

HEMANI, A., JANTSCH, A., KUMAR, S., POSTULA, A., OBERG, J., MILLBERG, M., AND LINDVIST, D. 2000. Network on a chip: An architecture for billion transistor era. In *Proceedings of the IEEE NorChip Conference*.

HU, J. AND MARCULESCU, R. 2005. Energy- and performance-aware mapping for regular NoC architectures. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 24*, 4.

HYATT, C. AND AGRAWAL, D. P. 1997. Congestion control in the wormhole-routed torus with clustering and delayed deflection. *In Proceedings of Parallel Computing, Routing, and Communications Workshop*.

JANTSCH, A. AND TENHUNEN, H. EDS. 2003. *Networks on Chip*. Kluwer.

LEE, H. G., CHANG. N, OGRAS, U. Y., AND MARCULESCU, R. 2007. On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches. *ACM Trans. Des. Automat. Elect. Syst. 12*, 3.

LIANG, L., LAFFELY, A., SRINIVASAN, S., AND TESSIER, R. 2004. An architecture and compiler for scalable on-chip communication. *IEEE Trans. VLSI Syst. 12*, 1.

MENDEL, J. M. 1995. *Lessons in Estimation Theory for Signal Processing, Communications, and Control*. Prentice Hall, Englewood Cliffs, NJ.

MILLBERG, M., NILSSON, E., THID, R., AND JANTSCH, A. 2004. Guaranteed bandwidth using looped

containers in temporally disjoint networks within the nostrum network on chip. In *Proceedings of the Design Automation and Test in Europe Conference*.

MURALI, S., BENINI, L., AND DE MICHELI, G. 2005. Mapping and physical planning of networks on chip architectures with quality-of-service guarantees. In *Proceedings of the Asia and South Pacific Design Automation Conference*.

NILSSON, E., MILLBERG, M., OBERG, J., AND JANTSCH, A. 2003. Load distribution with the proximity congestion awareness in a network on chip. In *Proceedings of the Design Automation and Test in Europe Conference*.

OGRAS, U. Y. AND MARCULESCU, R. 2005. Energy- and performance-driven NoC communication architecture synthesis using a decomposition approach. In *Proceedings of the Design Automation and Test in Europe Conference*.

OGRAS, U. Y. AND MARCULESCU, R. 2006. "It's a small world after all": NoC performance optimization via long-range link insertion. *IEEE Trans. VLSIS (Special Section on Hardware/Software Codesign and System Synthesis) 14*, 7.

OGRAS, U. Y. AND MARCULESCU, R. 2007. Analytical router modeling for networks-on-chip performance analysis. In *Proceedings of the Design Automation and Test in Europe Conference*.

PAGANINI, F., DOYLE, J., AND LOW, S. 2001. Scalable laws for stable network congestion control, In *Proceedings of IEEE Conference on Decision and Control*.

PARK, K. AND WILLINGER, W. EDS. 2000. *Self-Similar Network Traffic and Performance Evaluation*. Wiley Interscience.

PEH, L. S. AND DALLY, W. J. 2001. A delay model for router micro-architectures. *IEEE Micro*.

PINTO, A. AND SANGIOVANNI-VINCENTELLI, A. 2003. Efficient synthesis of networks on chip. In *Proceedings of the 21st International Conference on Computer Design*.

PULLINI, A., ANGIOLINI, F., BERTOZZI, D., AND BENINI, L. 2005. Fault tolerance overhead in network-on-chip flow control schemes. In *Proceedings of Symposium on Integrated Circuits and System Design*.

RADULESCU, ET AL. 2005. An efficient on-chip ni offering guaranteed services, shared-memory abstraction, and flexible network configuration. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 24*, 1.

THE SEMICONDUCTOR INDUSTRY ASSOCIATION. 2006. The International Technology Roadmap for Semiconductors (ITRS). San Jose, CA.

QIU, D. AND SHRO, N. B. 2004. A predictive flow control mechanism to provide QoS and efficient network utilization, *IEEE Trans. Network. 12*, 1.

SGROI, M., SHEETS, M., MIHAL, A., KEUTZER, K., MALIK, S., RABAEY, J., AND SANGIOVANNI-VINCENTELLI, A. 2001. Addressing the system-on-a-chip interconnect woes through communication-based design. In *Proceedings of Design Automation Conference*.

SMAI, A. AND THORELLI, L. 1998. Global Reactive Congestion Control in Multicomputer Networks. In *Proceedings of the 5th International Conference on High Performance Computing*.

SRINIVASAN, K., CHATHA, K. S., AND KONJEVOD, G. 2004. Linear programming based techniques for synthesis of network-on-chip architectures. In *Proceedings of the IEEE International Conference on Computer Design*.

THOTTETHODI, M., LEBECK, A. R., AND MUKHERJEE, S. S. 2001. Self-tuned congestion control for multiprocessor networks. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture.*

VARATKAR, G. AND MARCULESCU, R. 2004. On-Chip traffic modeling and synthesis for MPEG-2 video applications. *IEEE Trans. VLSI 12*, 1, 108–119.

ZEFERINO, C. A., SANTO, F. M. E., AND SUSIN, A. A. 2004. Paris: A parameterizable interconnect switch for networks-on-chip, In *Proceedings of Symposium on Integrated Circuits and Systems Design*.

Received June 2006; revised March 2007, July 2007; accepted August 2007