Towards On-Chip Fault-Tolerant Communication*

Tudor Dumitraş Sam Kerner Radu Mărculescu

ECE Department
Carnegie Mellon University
Pittsburgh, PA 15213
tdumitra, skerner, radum@ece.cmu.edu

Abstract — As CMOS technology scales down into the deep-submicron (DSM) domain, devices and interconnects are subject to new types of malfunctions and failures that are harder to predict and avoid with the current system-on-chip (SoC) design methodologies. Relaxing the requirement of 100% correctness in operation drastically reduces the costs of design but, at the same time, requires SoCs be designed with some degree of system-level fault-tolerance. In this paper we introduce a high-level model of DSM failure patterns and propose a new communication paradigm for SoCs, namely stochastic communication. Specifically, for a generic tile-based architecture, we propose a randomized algorithm which not only separates computation from communication, but also provides the required fault-tolerance to on-chip failures. This new technique is easy and cheap to implement in SoCs that integrate a large number of communicating IP cores.

I. Introduction

As modern VLSI chips are getting more complex, the designers are facing new challenges. Semi-custom ASICs have evolved into complicated systems-on-chip (SoCs) where dozens, and soon hundreds of pre-designed IPs are assembled together to form large chips with complex functionality. Extensive research on how to integrate and connect these IPs is currently being conducted [12], but many questions remain open, as these issues are very difficult to address within the existing framework of CAD tools.

It is clear that new design methodologies are required to deal with these arising problems. As emphasized in ITRS 2001 [14], it is very important, especially at system-level, to separate the *computation* from *communication*, as they are orthogonal issues which should remain separate whenever possible. Henceforth, SoC design should resemble the creation of large-scale communication networks rather than traditional IC design practice. Furthermore, as the CMOS technology scales down into the nanometer domain, many factors will influence the cost and performance of VLSI designs [11]. Critical leakage currents and high field effects will lead to more transient and permanent failures of signals, logic values, devices, and interconnects [15]. These effects have a negative influence on the interconnect delay and the existing timing models cannot be used anymore. Even with the emergence of new wiring technologies, like the recent copper wiring process developed at IBM [16], the IC designers are still facing the problem that the existing CAD tools and methodologies are no longer efficient in preventing pathologic behavior. This suggests that chips have to be designed with some built-in *fault-tolerance*. Relaxing the requirement of 100% correctness in operation for devices and interconnects when designing chips may also dramatically reduce the costs of manufacturing, verification, and test [14]. Distributed computing has dealt with fault-tolerance for a long time; unfortunately most of those algorithms are unlikely to be useful, because they need a lot of resources which are not available on-chip.

Recently, it has been proposed to connect the IPs using a *network-on-chip* (NoC) architecture [4]. For example, various modules can be placed on a network of 16 tiles which are connected on a 4×4 grid (as in Fig. 1). Consequently, every tile can communicate directly with its four nearest neighbors, except for the tiles on the border/corners, which have only three/two neighbors. As such, the end-to-end communication between non-adjacent tiles will have to be implemented by a set of high-level protocols. These *regular structures* are very attractive because they can offer well-controlled electrical parameters (which enable high performance circuits by reducing latency and increasing bandwidth) and lower probabilities of failure for individual nodes.

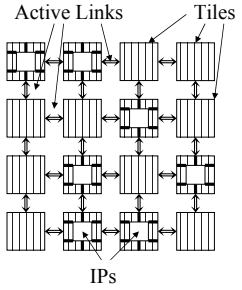

Fig. 1. Tile-based architecture

The problem of defining *communication protocols* for these NoCs has not been addressed yet and does not seem to be an easy matter, as the resources used in traditional networks are not available on-chip. Classic data link layer protocols, as the Internet Protocol or the ATM Layer [1], need *retrans-*

*Research supported by NSF CCR-00-93104, DARPA/Marco Gigascale Research Center (GSRC), and SRC 2001-HJ-898.

missions in order to deal with incorrect data, thus significantly increasing the latency. Generally speaking, simple *deterministic* algorithms do not cope very well with random failures [10]. On the other hand, the cost of implementing full-blown *adaptive routing* for the NoCs is prohibitive because of the need of very large buffers, lookup tables and complex shortest-path algorithms [1].

To address these problems, the present paper introduces a new communication paradigm called *on-chip stochastic communication*. We place ourselves in an NoC context, as in Fig. 1. The IPs communicate using a probabilistic broadcast scheme, similar to the *randomized gossip protocols* [5]. If a tile has a packet to send, it will forward the packet to a randomly chosen subset of the tiles in its neighborhood. This way, the packets are *diffused* from tile to tile to the entire NoC. Every IP then selects from the set of received messages only the ones that have its own ID as the destination.

This simple algorithm achieves many of the desired features of the future NoCs. As shown later in this paper, the algorithm provides:

- *separation between computation and communication*, as the communication scheme is implemented in the network logic and is transparent to the IPs;
- *fault-tolerance* since a message can still reach its destination even under severe levels of DSM failures on the chip;
- *low latency* since it does not require retransmissions (our results indicate great improvements over a bus-based solution);
- *design flexibility* since it provides a mechanism to tune the tradeoff between performance and energy consumption.

In summary, we propose a fault-tolerant solution for the NoC communication which lowers the production costs, simplifies the design, and can be customized for various applications and architectures.

This paper is organized as follows: at first we review previous work relevant to this paper. In Section III, we introduce our approach and present the metrics for performance evaluation. In Section IV, we show experimental results that clearly indicate the great potential of this approach. Finally, we conclude by summarizing our main contribution.

II. Previous Work

Our communication scheme is derived from a class of randomized broadcast primitives called *gossip algorithms* [5], that have been used before in computer networks and distributed databases. The behavior of such a communication scheme is similar to the spreading of an epidemic, which usually is disseminated exponentially fast [2].

Demers et al. [5] proposed to use randomized gossip for the lazy update of data objects in a database replicated at many sites and proved that, with their algorithm, the updates are spread *exponentially* fast among the instances of the database. Several networking protocols have been developed based on the same principles, as the gossip protocols proved to be very *scalable* and to maintain a *steady throughput* [3, 1]. More recently, these types of algorithms have been applied to networks of sensors [6]. Their ability to limit the communication to local regions and to support light-weight protocols, while still accomplishing their task, is appealing to applications where power, complexity and size constraints are critical. We argue that this paradigm can be successfully applied to the SoCs as well, especially for NoC architectures like the one in Fig. 1.

From a design perspective, in order to deal with node failures, Valtonen et al. [17] proposed an architecture based on autonomous, error-tolerant cells. In their approach, all cells can be tested at any time for errors and, if needed, disconnected from the network by the other cells. However, the authors do not specify the communication scheme which would ensure the desired fault-tolerance on such architectures. Furthermore, the problem of data upsets (see Section III.A) and their impact on NoC communication has not been addressed yet, so our paper fills an important gap in this research area of on-chip networks.

III. Description of Proposed Approach

In what follows, we propose a failure model for NoCs and introduce a new class of protocols that we call *on-chip stochastic communication*. The properties and advantages of this communication scheme are discussed in detail in the following sections.

A. Failure Model for NoCs

Several failure models have been identified in the traditional networking literature [7]. *Crash failures* are *permanent* faults which occur when a tile halts prematurely or a link disconnects, after having behaved correctly until the failure. *Transient* faults can be either *omission failures*, when links lose some messages and tiles intermittently omit to send or receive, or *arbitrary failures* (also called Byzantine or malicious), when links and tiles deviate arbitrarily from their specification, corrupting or even generating spurious messages.

However, as some of these models are not very relevant to the on-chip networks, we have developed a fault model that is more suited to the NoC context. Because of crosstalk and electromagnetic interference, the most common type of failures in DSM circuits will be data transmission errors (also called *upsets*) [14]. Simply stated, if noise in the interconnect causes a message to be scrambled, a *data upset* error will occur; these errors are subsequently characterized by a probability p_{upset}. Another common failure appears when a message is lost because of buffer overflow; this is modeled by probabilities p_{send_miss} if it happens in a send buffer and p_{recv_miss} for a receive buffer. Since our algorithm treats scrambled messages as lost messages (see Section III.B), we can combine these three probabilities into one, p_{lost}, which gives the odds that a message transmission fails; that is:

$$p_{lost} = p_{upset} + p_{send\_miss} + p_{recv\_miss} \quad (1)$$

Furthermore, in order to show that our algorithm can work on partially defective chips (thus helping to reduce the verification costs), we allow tiles and links to be manufactured unreliably. Summarizing, the fault model we propose depends on the following parameters:

- p_{lost}, the probability a message is lost (either because of *data upsets*, or because of *buffer overflows*);
- p_{tiles}, the probability a tile is non-functional from manufacturing;
- p_{links}, the probability a link is defective from manufacturing.

If the links are experiencing arbitrary failures, we must also consider how the information transmitted is altered. If a message contains n bits, the error vector is defined as: $\underline{e} = (e_1, e_2, \ldots, e_n)$, where $e_i = 1$ if an error occurs in the i^{th} transmitted bit and $e_i = 0$ otherwise. If all $2^n - 1$ non-null error vectors are equally likely to occur, we have the *random error vector model*. In this model the probability of $\underline{e}$ does *not* depend on the number of bit errors it contains, therefore:

$$p_{upset} = \sum_{\underline{e} \neq 0} P[\underline{e}] = (2^n - 1)\, p_v \approx 2^n p_v \Rightarrow p_v \approx \frac{p_{upset}}{2^n}$$

where p_v is the probability of an error vector $\underline{e}$. In contrast, in the *random bit error model*, $e_1 \ldots e_n$ are independent of each other, so:

$$p_{upset} = 1 - \sum_{\underline{e}=0} P[\underline{e}] = 1 - (1 - p_b)^n \approx n p_b \Rightarrow p_b \approx \frac{p_{upset}}{n}$$

where p_b is the probability of a bit error.

We believe that establishing this stochastic failure model is an decisive step towards solving the fault-tolerant communication problem, as it emphasizes the non-deterministic nature of DSM faults. This suggests that a stochastic approach (described subsequently) is best suited to deal with these realities.

B. Stochastic Communication

Traditionally, data networks have dealt with fault-tolerance by using complex algorithms, like the Internet Protocol or the ATM Layer [1]. However, these algorithms require a lot of resources that are not available on-chip and they are not always capable to guarantee a constant low latency, which is vital for SoCs. For example, in these protocols, the packets are protected by a *cyclic redundancy code (CRC)*, which is able to detect if a packet contains correct or upset data. If an error is detected, the receiver will ask for the retransmission of the scrambled messages. This method is known as the *automatic retransmission request (ARQ)* paradigm, and it has the disadvantage that it increases the communication latency. Another approach is the *forward error correction (FEC)*, where the errors are corrected directly by the receiver by using an error correction scheme, like the Reed-Solomon codes. FEC is appropriate when a return channel is not available, as for instance in deep-space communications or in audio CD recordings. FEC, however, is less reliable than ARQ and incurs significant additional processing complexity [1].

In light of these considerations, we propose a *fast* and *computationally lightweight* paradigm for the on-chip communication, based on an error detection / multiple transmissions scheme. The key observation behind our strategy is that at chip-level bandwidth is less expensive than in traditional networks, because of existing high-speed buses and interconnection fabrics which can be used for the implementation of an NoC. Therefore, we can afford to have more packet transmissions than in the previous protocols in order to simplify the communication scheme and guarantee low-latencies.

We implement the end-to-end communication between the tiles of an NoC using a *probabilistic broadcast* algorithm [3]. A message propagates from tile to tile until the entire network becomes aware of it. The message is spread *exponentially fast*, and after $O(\log_2 n)$ stages it reaches *all* the tiles *with high probability (w.h.p.)*[1] (see Section III.D). However, the message might reach its destination before the broadcast is completed, so the spreading could be terminated even earlier. In order to do this, we assign a *time to live* (TTL) to every message upon creation and decrement it at every hop until it reaches 0; then the message is destroyed. This will lead to important bandwidth and energy savings, as shown in Section IV.

During transmission, packets are protected against data upsets by a CRC; if an error is discovered, then the packet will be discarded. Because a packet is retransmitted many times in the network, the receiver does *not* need to ask for retransmission, as it will receive the packet again anyway. Furthermore CRC encoders and decoders are easy to implement in hardware, as they only require one shift register [1].

An example of a simple Producer – Consumer application is shown in Fig. 2. On an NoC with 16 tiles the Producer is placed on tile 6 and the Consumer on tile 12; tiles 4, 5, 13 and 15 are assumed dead. Suppose the Producer needs to send a message to the Consumer. Initially the Producer sends the message to a randomly chosen subset of its neighbors (ex. tiles 2 and 7 in Fig. 2-a). At the second gossip round, tiles 6, 2, 7 (the Producer and the tiles that have received the message during the first round) forward it in the same manner. After this round, eight tiles (6, 2, 7, 1, 3, 8, 10, 11) know the message and are ready to send it to the rest of the network. At the third gossip round, the Consumer finally receives the packet from tiles 8 and 11. Note that:

- if one of the tiles tries to send the message to a faulty tile (in Fig. 2-(c) tile 3 tries to send to tile 4), nothing will happen because the receiver is not functioning correctly;
- if the packet transmitted by tile 8 is affected by a data upset, the Consumer will discard it, as it receives from tile 11 another copy of the same packet.
- the Producer needs *not* know the location of the Consumer, the message will still arrive at the destination *(w.h.p.)*.
- the message reaches the Consumer before the full broadcast is completed (tiles 14 and 16 have not yet received the message).

[1]The term *with high probability* means with probability at least $1 - O(n^{-\alpha})$ for some positive constant α.

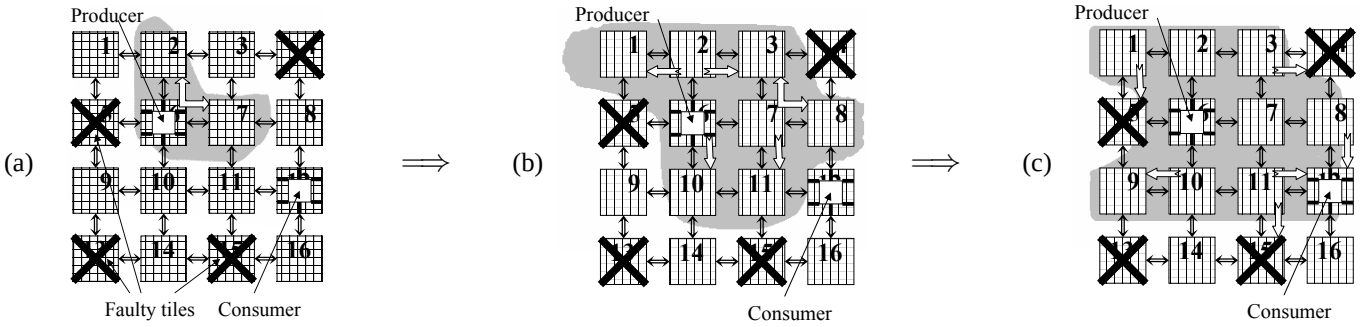

Fig. 2. Producer – Consumer application implemented on a stochastically communicating NoC

The pseudocode for this algorithm is shown in Fig. 3 (with standard set theory notations). The tasks executed by nodes and links are concurrent. A tile forwards the packet that is available to be sent to all four output ports, then every active link makes a random decision (with probability p) whether or not to transmit the message to its outbound tile. In Section IV we will show how this probability can be used to tune the tradeoff between performance and energy consumption. In our implementation, the links are assumed to be *active* (they have some control logic that can choose randomly whether or not an incoming packet is to be transmitted or not). This design choice is *not* vital for the performance of the algorithm; the random decision could be implemented in the tiles instead of the links. However, this further emphasizes the separation between computation and communication in the on-chip networks.

```
Every tile executes:
  INITIALIZATION:
    msg_list ← ∅
  AT EVERY GOSSIP ROUND:
    msg_list ← msg_list ∪ {m received | CRC_OK(m)}
    ∀m ∈ msg_list    m.TTL ← m.TTL - 1
    msg_list ← msg_list \ {m ∈ msg_list | m.TTL =
  0}
  WHEN NEW MESSAGE m IS GENERATED:
    m.TTL ← initial_TTL
    msg_list ← msg_list ∪ {m}
Every link executes:
  INITIALIZATION:
    in ← inbound node
    out ← outbound node
  AT EVERY GOSSIP ROUND:
    with probability p:  out receives in.msg_list
```

Fig. 3. The on-chip gossip algorithm

C. Energy Metrics

In estimating this algorithm's energy consumption we take into consideration the total number of packets sent in the NoC, since these transmissions account for the switching activity at network level. This is expressed by Eq. 2, where $N_{packets}$ is the total number of messages generated in the network, S is the average size of one packet (in bits) and E_{bit} is the energy consumed per bit:

$$\begin{aligned} E_{total} &= E_{computation} + E_{communication} \\ &= E_{computation} + N_{packets}\, S\, E_{bit} \end{aligned} \qquad (2)$$

$N_{packets}$ can be estimated by simulation, S is application-dependent, and E_{bit} is a parameter from the technology library. As shown in Eq. 2, the total energy consumed by the chip will be influenced by the activity in the computational cores as well ($E_{computation}$). Since we are trying to analyze here the performance and properties of the *communication scheme*, estimating the energy required by the computation is not relevant to the present paper. This can be added, however, to our estimations from Section IV to get the combined energy values.

D. Performance Metrics

A *broadcast round* is the time interval in which a tile has to finish sending all its messages to the next hops; this will usually take several clock cycles. The optimal duration of a round T_R can be determined using Eq. 3, where f is the maximum frequency of any link, $N_{packets/round}$ is the average number of packets that a link sends during one round (which is application-dependent), and S is the average packet size.

$$T_R = \frac{N_{packets/round}\, S}{f} \qquad (3)$$

Although there are no references in the literature on how these algorithms should be implemented in a real system, the gossip-based broadcast has been theoretically studied for a fully connected mesh network [13]. It can be proved that, if S_n is the number of rounds until everybody receives the gossip, then:

$$S_n = \log_2 n + \ln n + O(1) \quad \text{as} \quad n \to \infty \qquad (4)$$

with probability 1. Therefore, after $O(\log_2 n)$ rounds (where n represents the number of nodes in the network) all the nodes have received the message *w.h.p.* [9] (see Fig. 4). However, as in the previous example, the message can reach its destination before the broadcast is completed, so the number of rounds actually needed for the end-to-end communication could be even smaller.

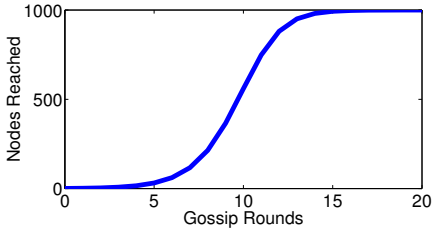

Fig. 4. Message spreading using randomized gossip in a 1000 node network

Randomized gossip performs better than the deterministic algorithms under the presence of faults: F node crash failures

result in only $O(F)$ nodes that do not receive the message [9]. Furthermore, the fact that we don't store or compute the shortest paths (as in the case of dynamic routing) makes this algorithm *computationally lightweight*, simpler and easier to customize for every application and interconnection network.

The following parameters are relevant to our analysis:

- The *number of broadcast rounds needed* is a direct measure of the inter-IP communication *latency*;
- The *total number of packets* sent in the network shows the bandwidth required by the algorithm and can be controlled by varying the message TTL;
- The *fault-tolerance* evaluates the algorithm's resilience to abnormal functioning conditions in the network;
- The *energy consumption*, computed with Eq. 2.

E. Possible Design Methodology for Stochastic Communication

The design flow for an SoC using stochastic communication would need the following steps: 1) identify and describe the concurrent modules that compose the application; 2) design the IPs that will implement these modules, or select them from an existing IP library; 3) map and place the IPs on an off-the-shelf NoC, which will take care of all the communication between them.

The chip's correct functional behavior still needs to be validated, but extensive verification and integration tests can be completely avoided. The stochastic protocol makes the *communication fault-tolerant in nature* and, as shown in Section IV, in some cases the IPs can be duplicated in order to increase the fault-tolerance of the computation as well.

It is obvious that such a methodology will lower drastically the costs of SoC design, especially if a large library of IPs is available. This would enable the mass production of low-cost, disposable devices which need not have the 100% robustness and reliability of traditional VLSI chips.

IV. Practical Considerations and Results

Stochastic communication can have a wide range of applicability, ranging from parallel SAT solvers and multimedia applications to periodic data acquisition from non-critical sensors. In order to demonstrate our technique, we choose a classic parallel application, the Fast Fourier Transform, and simulate it in a stochastically communicating NoC environment. In this section we discuss our results for this setup.

We have implemented our algorithm in Matlab's Stateflow, a tool for describing and simulating concurrent behavior of complex systems, which uses a formalism defined in [8]. In our simulations we assume the failure model described in Section III.A, which has the following parameters: p_{tiles} and p_{links}, the proportions of tiles and links that are defective from construction, and p_{lost}, the combined probability that a packet is scrambled during transmission or dropped because of buffer overflow. As realistic data about failure patterns in regular SoCs are currently unavailable, we exhaustively explore here the parameter space of our failure model. Another important parameter that we vary is p, the probability that a packet is forwarded over a link (see Section III.B). We compare here four versions of the stochastic communication, obtained for different values of the parameter p:

- the *network flooding*, which is a deterministic algorithm where the tiles send the messages to all their neighbors all the time;
- three versions of the *randomized gossip algorithm* described in Fig. 3, which differ in the probability that the links transmit the message to the neighbors (we have used $p = 0.75$, $p = 0.5$ and $p = 0.25$).

The reason why we are comparing our protocol with the flooding algorithm is because the latter is the simplest example of a deterministic broadcast protocol. The flooding algorithm is also optimal with respect to latency; that is, the number of intermediate hops between source and destination is always equal to the Manhattan distance between the two tiles. We will show that our protocol has a latency close to this optimum. The flooding is, however, *extremely* inefficient with respect to the bandwidth used and the energy consumed, while the gossip algorithm allows us to tune the tradeoff between energy and performance by varying the probability of transmission p.

A. Case Study: The Two-Dimensional Fast Fourier Transform

The Fourier Transform has multiple applications in linear systems analysis, antenna studies or signal processing. Its most common implementation is known as the Fast Fourier Transform (FFT), which is currently one of the largest single consumers of floating-point cycles in modern CPUs and is extensively used in multimedia and wireless communication chips. The Discrete Fourier Transform of N samples is defined as:

$$\begin{aligned} \hat{x}(k) &= \frac{1}{N}\sum_{n=0}^{N-1} x(n) e^{-\frac{2i\pi}{N}kn} \qquad (5) \\ &= \begin{cases} \frac{1}{2}\left(\hat{x}_1(k) + e^{-\frac{2i\pi}{N}k}\hat{x}_2(k)\right) & \text{for} \quad k < \frac{N}{2} \\ \frac{1}{2}\left(\hat{x}_1(k) - e^{-\frac{2i\pi}{N}k}\hat{x}_2(k)\right) & \text{for} \quad k > \frac{N}{2} \end{cases} \end{aligned}$$

where $\hat{x}_1(k) = \hat{x}\,(2k) \quad$ and $\quad \hat{x}_2(k) = \hat{x}\,(2k+1)$.

Hence a recursive divide and conquer algorithm will be used to compute the FFT of N samples (Fig. 5). This reduces the number of operations from $O(N^2)$ to $O(N \log_2 N)$. Using this scheme, the FFT algorithm can also be implemented on a parallel architecture. Every node in the tree from Fig. 5 represents a parallel process. The leaves compute the FFT on a small number of samples and send the results back up to the root, which will finally assemble the full FFT.

Because of its wide-spread use in engineering problems and especially in image and multimedia processing, we have decided to analyze the *two-dimensional FFT* algorithm (FFT2), which is a generalization of the above scheme, where every node has 4 children instead of 2. We mapped this application onto a 4×4 NoC, running the gossip algorithm described in

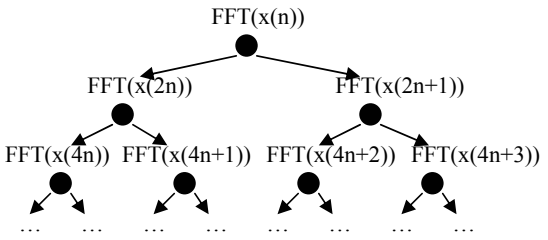

Fig. 5. Parallel scheme for computing the FFT

Section III. As mentioned before, the gossip algorithm provides a natural fault-tolerance to the inter-IP *communication*. In order to increase the *computation*'s fault-tolerance as well, each module can be duplicated, such that if one is located on a dysfunctional tile, the other one will still be able to finish the task.

We test these algorithms in an unreliable NoC environment, assuming that failures occur according to the model in Section III.A. We evaluate the *latency*, *the energy dissipation* and the *fault-tolerance* of our approach. The results presented in this section were obtained after several repeated simulations. Since we are dealing with probabilistic protocols, we present average values for estimated parameters and indicate their standard deviation.

A summary of our results is given in Tables I and II. For example, the second row of Table I shows that, when the probability of an unsuccessful transmission is $p_{lost} = 0.2$, depending on the algorithm used the broadcast of the first message takes 5–30 rounds to reach all the tiles, FFT is computed after 5–16 rounds, there are 444–811 packets transmitted in the network, with an average energy consumption of 7.61–21.29 nJ per message bit[2]. The second row of Table II shows the number of rounds needed to finish: the initial broadcast / the computation of FFT2, for different levels of tile and link failures, when $p_{lost} = 0.2$.

IV.A.1 Latency.

The evolution of the initial broadcast is shown in the upper half of Fig. 6. The spread soon reaches a stage of explosive growth and the whole network becomes aware of the message after a small number of rounds. The shape of these curves is similar for all four algorithms tested and is very close to the one predicted by theory (Fig. 4). The presence of transmission failures slows down the spreading (the dashed line in Fig. 6), but still the message reaches all the tiles.

In the lower part of Fig. 6, we note that stochastic communication with probability of transmission $p = 0.5$ has a latency very close to the optimal one (displayed by the flooding algorithm) when up to 50% of the sent packets get corrupted. On average, we notice that replies come back *before* the full broadcast of the original message is completed, which suggests that the spreading process could be stopped sooner by specifying a finite TTL value. The standard deviation (shown by the error bars in Fig 6) is small, which proves that stochastic

[2]The total energy dissipation can be obtained by multiplying these figures with the packet size S (see Eq. 2).

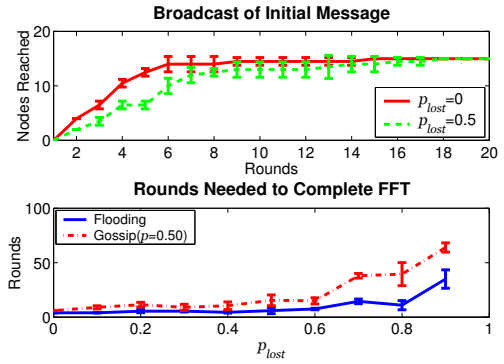

Fig. 6. Latency of the Stochastic Communication

communication has a *stable behavior* and the results are can be reproduced across different runs of the algorithm.

IV.A.2 Energy Dissipation.

We have estimated the energy dissipation of our algorithms, using Eq. 2. Since our goal is to analyze a new communication paradigm, we do not include the energy consumed during the computation. Therefore all the results presented here reflect the performance of the NoC stochastic communication.

In the upper part of Fig. 7 we compare the energy consumption of flooding and gossip ($p = 0.5$) algorithms. As expected, the gossip algorithm (which generates less messages than flooding) displays a significant reduction in the energy dissipation with respect to the flooding algorithm. The energy dissipation drops to 0 when $P_{lost} \approx 100\%$ because the all the packets are corrupted and therefore they are not retransmitted anymore. The energy dissipation of the gossip algorithm also has a smaller variance across several runs of the protocol.

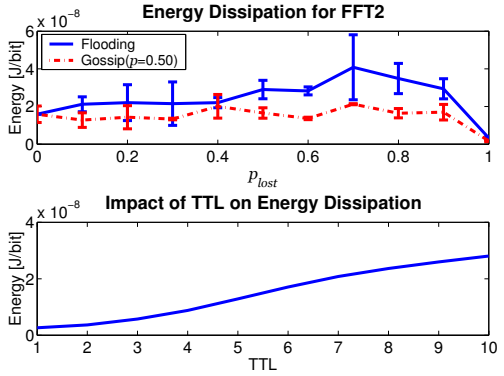

Fig. 7. Energy dissipation of the Stochastic Communication

Further energy savings can be achieved by stopping the spread of the messages after a certain number of rounds. This can be done by assigning a finite time to live (TTL) to the messages. For a small TTL, neither the broadcast of the initial message, nor the task of the application can be completed. However, after a certain threshold, the latency does not improve if the TTL is increased. Fig. 7 shows that the energy consumption increases almost linearly with the TTL. This suggests that the TTL can be set to the smallest value that guarantees the completion of the tasks, in order to keep the energy consumption at the minimum level required.

TABLE I

STOCHASTIC COMMUNICATION IN AN NOC WITH 6.25% DEFECTIVE TILES AND 8.33% DEFECTIVE LINKS

$p_{lost} = p_{upset} + p_{send\_miss} + p_{recv\_miss}$	Initial Broadcast [rounds]				FFT2 Computed [rounds]				Total Number of Packets				Avg. Energy [$\times 10^{-8}$ J/bit]			
	Flood	Gossip (p)			Flood	Gossip (p)			Flood	Gossip (p)			Flood	Gossip (p)		
		0.75	0.50	0.25		0.75	0.50	0.25		0.75	0.50	0.25		0.75	0.50	0.25
0	5	7	–	16	5	6	9	16	471	481	619	558	2.261	1.924	1.651	0.836
0.2	5	9	12	30	5	7	11	16	444	559	811	508	2.129	1.917	1.768	0.761
0.4	9	11	–	–	5	13	16	20	387	1417	1289	486	1.858	2.616	1.934	0.583
0.6	19	11	17	38	10	16	18	39	1226	1997	1172	1700	2.942	2.995	1.562	1.046
0.8	22	18	46	85	23	25	34	84	4158	2734	2080	3562	4.339	2.624	1.468	1.018
1	–	–	–	–	–	–	–	–	1864	1410	1068	466	0.334	0.253	0.191	0.083

TABLE II

IMPACT OF FAILURES ON THE LATENCY OF THE GOSSIP ALGORITHM WITH $p = 0.5$ (ROUNDS TO COMPLETE: INITIAL BROADCAST / FFT2)

$p_{lost} = p_{upset} + p_{send\_miss} + p_{recv\_miss}$	p_{tiles} = 0				p_{tiles} = 0.125				p_{tiles} = 0.25			
	p_{links}				p_{links}				p_{links}			
	0	0.083	0.166	0.25	0	0.083	0.166	0.25	0	0.083	0.166	0.25
0	8/6	9/9	15/8	–/9	8/10	7/7	13/13	–/11	6/8	–/–	–/14	–/–
0.2	11/10	10/8	–/12	–/12	9/12	–/10	23/19	–/19	8/–	–/20	–/17	–/–
0.4	9/11	16/14	–/14	–/25	15/10	15/16	–/16	–/–	–/11	–/37	–/9	–/22
0.6	15/18	21/13	–/22	–/25	43/15	–/36	21/12	–/–	24/16	–/16	–/17	–/–
0.8	33/27	46/33	83/37	–/69	24/53	–/49	35/46	–/–	–/–	42/–	–/39	–/35
1	–/–	–/–	–/–	–/–	–/–	–/–	–/–	–/–	–/–	–/–	–/–	–/–

IV.A.3 Fault-Tolerance.

Different types of failures have different effects on the performance of stochastic communication (see Table II). The levels of defective links and tiles do not seem to have a big impact on latency, however the computation of FFT2 will fail completely if these levels are too high, because too many important modules are not working or entire regions of the chip are isolated. The computation of FFT2 succeeds in more cases than the initial broadcast, for instance even with 12.5% faulty tiles and 16.67% faulty links, FFT2 succeeds in almost all the runs of the protocol. The initial broadcast is especially affected by the number of defective links, which can disconnect a region of the chip from the network and prevent those tiles from sending/receiving messages. However, because of the resource duplication, even in these cases the computation of FFT2 may succeed, if there are enough vital resources left that can communicate with each other.

On the other hand, data upsets seem to have little influence on the chances that FFT2 has to succeed. However upsets do have an impact on the latency and energy dissipation, especially if $p_{lost} > 0.5$ (see Fig. 8). FFT2 cannot finish with more than 8 non-functional tiles; however, for less than 4 stopped tiles, it will eventually succeed with levels of data upsets as high as 90%, even if it requires 100 rounds to do so.

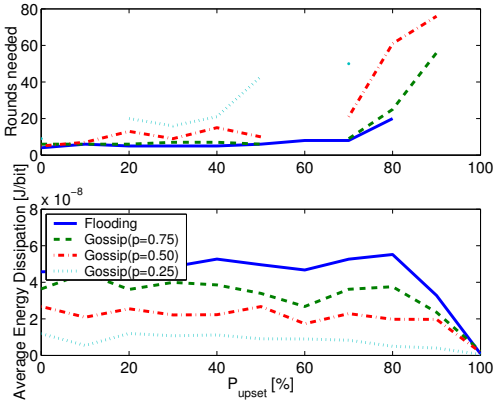

Fig. 8. Impact of data upsets on latency and energy dissipation

V. DISCUSSION AND FUTURE WORK

The communication paradigm introduced in this paper has many interesting properties. Its main advantage over traditional interconnection schemes is the *intrinsic tolerance* to DSM failures, as stochastic communication tolerates high levels of data upsets without needing retransmissions. The protocol also offers a tradeoff between performance and energy consumption which can be tuned by varying the transmission probability. We have started an in-depth study of how *synchronization faults* affect our scheme's performance, and the preliminary results indicate that stochastic communication has a very good tolerance to timing errors as well.

We have also compared our technique with a bus-based solution for the FFT2 application. Both implementations use a 0.25μm technology. The length of the bus is equal to the side of the tile-based grid, and the modules were placed on both sides of the bus. Based on these parameters we calculated that the bus has a maximum working frequency of 43 MHz and it dissipates $21.6 \cdot 10^{-10}$ J for every bit transmitted, while in the case of the tile-based architecture a link has a maximum working frequency of 381 MHz and it dissipates $2.4 \cdot 10^{-10}$ J per bit. The links in this case are faster because they are shorter than the bus in the classical solution. Note that the bus-based

implementation cannot work if the bus or some of the IPs malfunction, while our results have shown that the stochastically communicating environment is still functional when such faults occur. Furthermore, on average the energy consumed by the tile-based chip was comparable with the one of the bus-based solution, while the latency was 11 times lower when using our technique. This is because the links are shorter, and thus faster, and because the messages can arrive on parallel ports and do not have to compete for a shared resource. Furthermore, the energy$\times$delay product looks much better for the gossiping NoC ($7 \cdot 10^{-12}$ J·s per bit) than for the bus ($133 \cdot 10^{-12}$ J·s per bit).

We believe that an interesting direction for future research is to show how our algorithm works with a more complex, heterogeneous application. For example, an MP3 encoder/decoder is a good candidate application, because of the complex traffic patterns, the high bandwidth and constant bit rate requirements, and, not least, because of its interest for the module-based SoC design. Our preliminary results have shown that stochastic communication is very well suited to this application, maintaining an almost constant bit rate, tolerating high levels of data upsets, buffer overflows and even synchronization errors.

In Section IV we have shown that stochastic communication has a very good average case behavior with respect to latency and energy dissipation. In the worst case (which is very unlikely for typical levels of failures), the protocol will not deliver the packet to the destination; therefore this communication paradigm is better suited for applications which tolerate small levels of information loss, like the MP3 encoder/decoder mentioned above. If however the application requires strong reliability guarantees, these can be implemented by a higher level protocol built on top of the stochastic communication.

VI. Conclusion

We introduced a novel failure model for NoCs and proposed a new communication paradigm based on a stochastic communication protocol. This approach takes advantage of the large bandwidth that is available on-chip to provide the needed system-level tolerance to data upsets and crash failures. Our results have shown that this approach is more fault-tolerant and has a much lower latency compared to a traditional bus-based implementation, while keeping the energy consumption at about the same level. At the same time, this method drastically simplifies the design process by offering a low-cost and easy to customize solution for on-chip communication. We believe that our results suggest the big potential of this approach and they strongly indicate that further research in this area would lead to vital improvements of the SoC design.

References

[1] A. Leon-Garcia, I. Widjaja. *Communication Networks*. McGraw-Hill, 2000.

[2] N. Bailey. *The Mathematical Theory of Infectious Diseases*. Charles Griffin and Company, London, 2 edition, 1975.

[3] K. Birman et. al. Bimodal Multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, 1999.

[4] W. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *Proc. of The 38th DAC*, June 2001.

[5] A. Demers et. al. Epidemic algorithms for replicated database maintenance. In *Proc. ACM Symp. on Princ. of Distributed Computing*, 1987.

[6] D. Estrin et. al. Next century challenges: Scalable coordination in sensor networks. In *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking*. ACM, August 1999.

[7] V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR94-1425, Cornell University, Computer Science Department, 1994.

[8] D. Harel. Statecharts: A visual formalism for complex systems. *Sci. of Comp. Progr.*, 8(3):231–274, June 1987.

[9] R. Karp et. al. Randomized rumor spreading. In *Proc. IEEE Symp. on Foundations of Computer Science*, 2000.

[10] F. T. Leighton et. al. On the fault tolerance of some popular bounded-degree networks. In *IEEE Symp. on Foundations of Comp. Sci.*, 1992.

[11] V. Maly. IC design in high-cost nanometer technologies era. In *Proc. of The 38th DAC*, June 2001.

[12] G. Martin. Design methodologies for system level IP. In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 286–289, 1998.

[13] B. Pittel. On spreading a rumor. *SIAM Journal of Appl. Math.*, 47(3):213–223, 1987.

[14] Semiconductor Association. "The International Technology Roadmap for Semiconductors (ITRS)", 2001.

[15] Y. Taur and et. al. CMOS scaling into the nanometer regime. In *Proc. of the IEEE*, volume 85, April 1997.

[16] T. N. Theis. The future of interconnection technology. *IBM Journal on Research and Development*, 44(3), 2000.

[17] T. Valtonen et. al. Interconnection of autonomous error-tolerant cells. In *Proc. of ISCAS*, 2002.