Chapter #

ON-CHIP STOCHASTIC COMMUNICATION

Tudor Dumitraş and Radu Mărculescu
Carnegie Mellon University, Pittsburgh, PA 15213, USA

Abstract: As CMOS technology scales down into the deep-submicron (DSM) domain, the Systems-On-Chip (SoCs) are getting more and more complex and the costs of design and verification are rapidly increasing due to the inefficiency of traditional CAD tools. Relaxing the requirement of 100% correctness for devices and interconnects drastically reduces the costs of design but, at the same time, requires that SoCs be designed with some degree of system-level fault-tolerance. In this chapter, we introduce a new communication paradigm for SoCs, namely stochastic communication. The newly proposed scheme not only separates communication from computation, but also provides the required built-in fault-tolerance to DSM failures, is scalable and cheap to implement. For a generic tile-based architecture, we show how a ubiquitous multimedia application (an MP3 encoder) can be implemented using stochastic communication in an efficient and robust manner. More precisely, up to 70% data upsets, 80% packet drops because of buffer overflow, and severe levels of synchronization failures can be tolerated while maintaining a much lower latency than a traditional bus-based implementation of the same application. We believe that our results open up a whole new area of research with deep implications for on-chip network design of future generations of SoCs.

Key words: System-on-Chip, Network-on-Chip, On-Chip Communication, High Performance, Stochastic Communication

1. INTRODUCTION

As modern systems-on-chip (SoCs) are becoming increasingly complex, the Intellectual Property (IP)-based design is widely accepted as the main reuse paradigm. This methodology makes a clear distinction between *computation* (the tasks performed by the IPs) and *communication* (the interconnecting architecture and the communication protocols used).

Currently, the traditional CAD tools and methodologies are very inefficient in dealing with a large number of IPs placed on a single chip.

A recently proposed platform for the on-chip interconnects is the network-on-chip (NoC) approach [1], where the IPs are placed on a rectangular grid of tiles (see Fig. 1) and the communication between modules is implemented by a stack of networking protocols. However, the problem of defining such *communication protocols* for NoCs does not seem to be an easy matter, as the constraints that need to be satisfied are very tight and the important resources used in traditional data networks are not available at chip level.

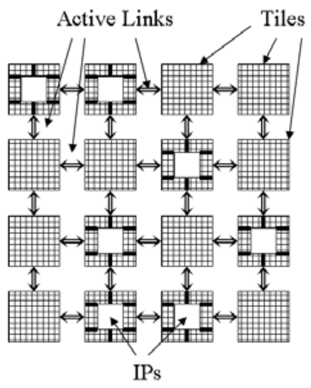

Figure #-1. Network on Chip

Furthermore, as the complexity of designs increases and the technology scales down into the deep-submicron (DSM) domain, devices and interconnect are subject to new types of malfunctions and failures that are harder to predict and avoid with the current SoC design methodologies. These new types of failures are impossible to characterize using *deterministic* measurements so, in the near future, *probabilistic* metrics, such as average values and variance, will be needed to quantify the critical design objectives, such as performance and power [10, 11].

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* [12]. Distributed computing has dealt with fault-tolerance for a long time; unfortunately most of those algorithms are unlikely to be useful, because they need significant resources which are not easily available on-chip. Furthermore, classic networking protocols, as the Internet Protocol or the ATM Layer [13], need *retransmissions* in order to deal with incorrect data, thus significantly increasing the latency. Generally, simple deterministic algorithms do not cope very well with random failures [14]. On the other hand, the cost of implementing full-blown *adaptive routing* [15] for NoCs is prohibitive because of the need of very large buffers, lookup tables and complex shortest-path algorithms.

To address these unsolved problems, the present chapter introduces a new communication paradigm called *on-chip stochastic communication*. This is similar, but *not* identical, to the proliferation of an epidemic in a large population [16]. This analogy explains intuitively the high performance and robustness of this protocol in the presence of failures. More precisely, we place ourselves in a NoC context (see Fig. 1), where the IPs communicate using a probabilistic broadcast scheme, known as *stochastic communication* [4]. 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 those that have its own ID as the destination.

This simple algorithm achieves many of the desired features of the future NoCs. As shown below, 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 despite severe levels of DSM failures on the chip;
– *extremely low latency* since this communication scheme does not require retransmissions in order to deal with incorrect data;
– *low production costs* because the fault-tolerant nature of the algorithm eliminates the need for detailed testing/verification;
– *design flexibility* since it provides a mechanism to tune the tradeoff between performance and energy consumption.

This chapter is organized as follows: at first, we review the previous work relevant to this field. In Section 3, we develop a novel failure model for NoCs and, in Section 4, we describe a communication scheme designed to work in such an environment. In Section 5, we show experimental results that clearly indicate the great potential of this approach. We conclude by summarizing our main contribution.

2. PREVIOUS WORK

One of the important mathematical challenges of this century has been developing analytical models for the proliferation of diseases. The earliest research in this direction [16] showed that, under certain conditions, epidemics spread *exponentially fast*. Much later, [3] proposed a probabilistic broadcast algorithm, the *randomized gossip*, for the lazy update of data objects in a database replicated at many sites and proved that this scheme behaves very much like the spreading of an epidemic. Several networking protocols are based on the same principles, as the gossip protocols proved to be very *scalable* and able to maintain *steady throughput* [7, 13]. Recently, these types of algorithms were applied to networks of sensors [8]. Their ability to limit communication to local regions and support lightweight protocols, while still accomplishing their task, is appealing to applications where power, complexity and size constraints are critical.

The problem of designing NoCs for fault-tolerance has been addressed only recently. From a design perspective, in order to deal with node failures, Valtonen et al. [9] proposed an architecture based on autonomous, error-tolerant cells, which can be tested at any time for errors and, if needed, disconnected from the network by the other cells. A more detailed failure model is described in [2], including data upsets and omission failures. However, a communication protocol which tolerates these types of errors has yet to be specified. Furthermore, the problem of synchronization failures (see Section 3), as well as their impact on NoC communication, have not been addressed so far. Our contribution tries to fill this important gap in the research area of on-chip networks.

3. FAILURE MODEL

Several failure models have been identified in the traditional networking literature [19]. *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, some of these models are not very relevant to the on-chip networks. In the DSM realm faults that are most likely to occur are fluctuations in the dopant distributions, high fields causing band-to-band tunneling and mobility degradation, or important leakage currents. For an efficient system-level communication synthesis, we need failure models that are easy to manipulate and that reflect the behavior of the circuits.

Because of crosstalk and electromagnetic interference, the most common type of failures in DSM circuits will be data transmission errors (also called upsets) [12]. Simply stated, if the 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 situation is when a message is lost because of the buffer overflow; this is modeled by the probability $p_{overflow}$. Furthermore, as modern circuits span multiple clock domains (as in GALS architectures [18]) and can function at different voltages and frequencies (as in a "voltage and frequency island"-based architecture advocated by IBM [17]) *synchronization errors* are very hard to avoid. In our experiments, we have adopted a tile-based architecture in which every tile has its own clock domain; synchronization errors are normally distributed with a standard deviation σ_{synchr}.

Summarizing, our fault model depends on the following parameters:

– p_{upset}: probability a packet is scrambled because of a *data upset*;
– $p_{overflow}$: probability a packet is dropped because of *buffer overflow*;
– σ_{synchr}: standard deviation error of the duration of a round (T_R) (see Section 4.3), which indicates the magnitude of *synchronization errors*.

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

4. STOCHASTIC COMMUNICATION

The technique we call *on-chip stochastic communication* implements the NoC communication using a *probabilistic broadcast* algorithm [4]. The behavior of such an algorithm is similar to the spreading of a rumor within a large group of friends. Assume that, initially, only one person in the group knows the rumor. Upon learning the rumor, this person (the initiator) passes it to someone (confidant) chosen at random. At the next round, both the initiator and the confidant, if there is one, select independently of each other someone else to pass the rumor to. The process continues in the same fashion; namely, everyone informed after t rounds passes the rumor at the $(t + 1)^{th}$ round to someone selected at random independently of all other past and present selections. Such a scheme is called *gossip algorithm* and is known to model the spreading of an epidemic in biology [16].

Let $I(t)$ be the number of people who have become aware of the rumor after t rounds ($I(0) = 1$) and let $S_n = min\ \{t : I(t) = n\}$ be the number of rounds until n people are informed. We want to estimate S_n, in order to evaluate how fast the rumor is spread. A fundamental result states that:

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

with probability 1 [6]. Therefore, after $O(\log_2 n)$ rounds (n represents the number of nodes) all the nodes have received the message *with high* *probability (w.h.p.)*[1] [4]. For instance, in Fig. 2, in less than 20 rounds as many as 1000 nodes can be reached. Conversely, after *t* rounds the number of people that have become aware of the rumor is an exponential function of *t*, so this algorithm spreads rumors *exponentially fast*.

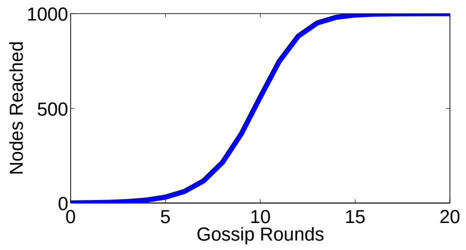

Figure #-2. Message spreading in a 1000 node fully connected network

The analogy we make for the case of NoCs is that tiles are the "gossiping friends" and packets transmitted between them are the "rumors" (see Fig. 3). Since any friend in the original setup is able to gossip with anyone else in the group, the above analysis can be applied directly only to the case of a fully connected network, like the one in Fig. 3a). However, because of its high wiring demands, such an architecture is *not* a reasonable solution for SoCs. Therefore, for NoCs, we consider a grid-based topology (Fig. 3b), since this is much easier and cheaper to implement on silicon. Although the theoretical analysis in this case is an open research question, our experimental results show that the messages can be spread explosively fast among the tiles of the NoC for this topology as well. To the best of our knowledge, this represents the first evidence that gossip protocols can be applied to SoC communication as well.

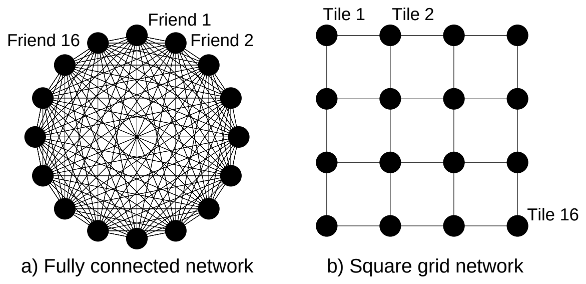

Figure #-3. Different topologies for a 16 node network

[1]The term *with high probability* means with probability at least $1 - O(n^{-\alpha})$ for some $\alpha>0$.

In the following paragraphs, we will describe an algorithm for on-chip stochastic communication and show how it can be used for a real-time multimedia application (an MP3 encoder).

4.1 Example: NoC Producer – Consumer Application

In Fig. 4 we give the example of a simple Producer – Consumer application. On a NoC with 16 tiles the Producer is placed on tile 6 and the Consumer on tile 12. 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. 4a). 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, and 11) become aware of 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:

– the Producer needs *not* know the location of the Consumer, the message will arrive at the destination *w.h.p.;*
– the message reaches the Consumer *before* the full broadcast is completed; that is, by the time when the Consumer gets the message, tiles 14 and 16 have not yet received the message.

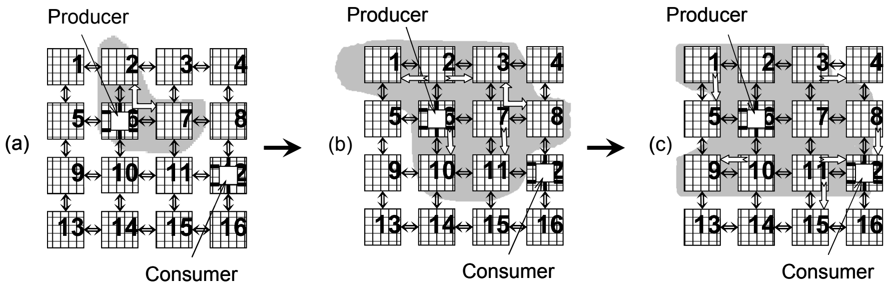

Figure #-4. Producer – Consumer application for a stochastically communicating NoC

The most appealing feature of this algorithm is its excellent performance in the presence of failures. For instance, suppose 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 anyway. The presence of such an upset is detected by implementing on each tile an error detection / multiple transmission scheme. The key observation is that, at chip level, bandwidth is *less* expensive than in traditional macro-networks. This is because of existing high-speed buses and interconnection fabrics which can be used for the implementation of a NoC. Therefore, we can afford more packet transmissions than in previous protocols, in order to simplify the communication scheme and guarantee low latencies.

4.2 The Algorithm

The mechanism presented in the above example assumes that tiles can detect when data transmissions are affected by upsets. This is achieved by protecting packets with a cyclic redundancy code (CRC) [13]. If an error is discovered, then the packet will be simply 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. This is the reason why stochastic communication can sustain low latencies even under severe levels of failures. Furthermore, CRC encoders and decoders are easy to implement in hardware, as they only require one shift register [13]. We also note that, since a message might reach its destination before the broadcast is completed, the spreading could be terminated even earlier in order to reduce the number of messages transmitted in the network. This is important because this number is directly connected to the bandwidth used and the energy dissipated (see Section 4.4). 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 can be garbage-collected. This will lead to important bandwidth and energy savings, as shown in Section 5.

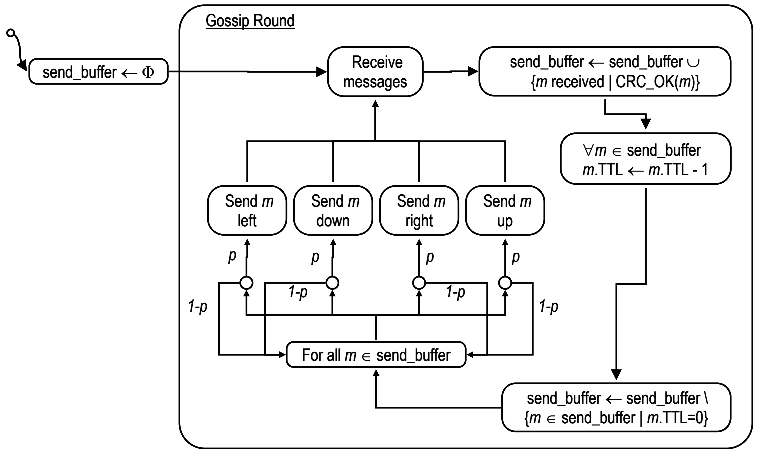
Gossip Round
send_buffer ← Φ
Receive messages
send_buffer ← send_buffer ∪ {m received | CRC_OK(m)}
∀m ∈ send_buffer m.TTL ← m.TTL - 1
Send m left
Send m down
Send m right
Send m up
p
p
p
p
1-p
1-p
1-p
1-p
For all m ∈ send_buffer
send_buffer ← send_buffer \ {m ∈ send_buffer | m.TTL=0}

Figure #-5. Stochastic Communication - The Algorithm

Our algorithm is presented in Fig. 5 (with standard set theory notations). The tasks executed by all the nodes are concurrent. A tile forwards the packet that is available for sending to all four output ports, and then a random decision is made (with probability p) whether or not to transmit the message to the next tile. In Section 5 we will show how this probability can be used to tune the tradeoff between performance and energy consumption.

4.3 Performance Metrics for Stochastic Communication

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. 2, 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.

$$T_R = \frac{N_{packets/round} S}{f} \qquad (2)$$

As we show in Section 5, the fast dissemination of rumors in this algorithm makes it possible to achieve very low latencies. This protocol spreads the traffic onto all the links in the network, reducing the chances that packets are delayed because of congestion. This is especially important in multimedia applications, where a sustainable constant bit rate is highly desirable. Furthermore, the fact that we don't store or compute the shortest paths (as is 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. 3 (see next section).

4.4 Energy Metrics for Stochastic Communication

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. 3, 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:

$$E_{total} = E_{computation} + E_{communication} = E_{computation} + N_{packets} S E_{bit} \quad (3)$$

$N_{packets}$ can be estimated by simulation, S is application-dependent, and E_{bit} is a parameter from the technology library. As shown in Eq. 3, 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 this discussion. This can be added, however, to our estimations from Section 5 to get the combined energy.

5. EXPERIMENTAL RESULTS

Stochastic communication can have a wide applicability, ranging from parallel SAT solvers and multimedia applications to periodic data acquisition from non-critical sensors. A thorough set of experimental results concerning the performance of this communication scheme has been presented in [4]. In the present chapter we demonstrate how our technique works with a complex multimedia application (an MP3 encoder). We simulate this application in a stochastically communicating NoC environment and in the following sections we discuss our results.

We have developed a simulator using the Parallel Virtual Machine (PVM)[2] and implemented our algorithm, as described in Section 4. In our simulations, we assume the failure model introduced in Section 3, where data upsets and buffer overflows uniformly distributed in the network with probabilities p_{upset} and $p_{overflow}$ respectively. The clocks are *not* synchronized and the duration of each round is normally distributed around the optimal T_R (see Eq. 2), with standard deviation σ_{synchr}. 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 we vary is p, the probability that a packet is forwarded over a link (see Section 4). For example, if we set the forwarding probability to 1, then we have a completely deterministic algorithm which floods the network with messages (every tile always sends messages to all its four neighbors). This algorithm is optimal with respect to latency, since the number of

[2] PVM is a programming platform which simulates a message passing multi processor architecture [20].

intermediate hops between source and destination is always equal to the Manhattan distance, but extremely inefficient with respect to the bandwidth used and the energy consumed. Stochastic communication allows us to tune the tradeoff between energy and performance by varying the probability of transmission *p* between 0 and 1.

5.1 Target Application: An MP3 Encoder

MP3 encoders and decoders are ubiquitous in modern embedded systems, from PDAs and cell phones to digital sound stations, because of the excellent compression rates and their streaming capabilities. We have developed a parallel version of the LAME MP3 encoder [5] (shown in Fig. 6) and introduced the code in our stochastically communicating simulator. The following results are based on this experimental setup.

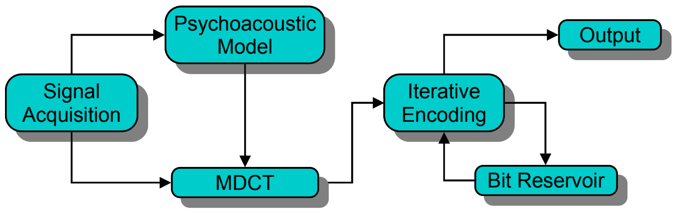

Figure #-6. The modules of the MP3 encoder

5.1.1 Latency

The latency is measured as the number of rounds MP3 needs to finish encoding and it is influenced by several parameters: the value of *p* (probability of forwarding a message over a link), the levels of data upsets in the network, the buffer overflows and the synchronization errors. Fig. 7 shows how latency varies with the probability of retransmission p and with the probability of upsets p_{upset}. As expected, the lowest latency is when $p = 1$ (the messages are always forwarded over a link) and $p_{upset} = 0$ (when there are no upsets in the network), and increases when $p \rightarrow 0$ and $p_{upset} \rightarrow 1$ to the point where the encoding of the MP3 cannot finish because the packets fail to reach their destination too often. Note that p_{upset} is dependent on the technology used, while *p* is a parameter that can be used directly by the designer in order to tune the behavior of the algorithm.

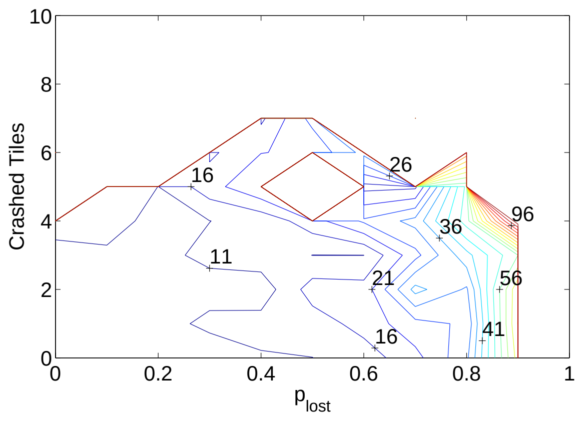

Figure #-7. Latency

5.1.2 Energy Dissipation

We have estimated the energy dissipation of our algorithms, using Eq. 3. We can see from Fig. 8 that the energy dissipation increases almost linearly with the probability of resending *p*. This is because the energy is proportional to the total number of packets transmitted in the network and this number is controlled by *p*. As we can see from Fig. 7, high values of *p* mean low latency, but they also mean high energy dissipation. This shows the importance of this parameter, which can be used to tune the tradeoff between performance and energy dissipation, making the stochastic communication flexible enough to fit the needs of a wide range of applications.

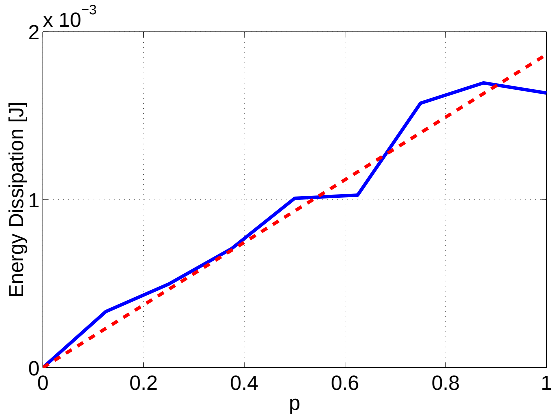

Figure #-8. Energy Dissipation

5.1.3 Fault-Tolerance

Different types of failures have different effects on the performance of stochastic communication. The levels of buffer overflow do not seem to have a big impact on latency (see left part of Fig. 9). However the encoding will not be able to complete if these levels are too high (>80%, as in point A in Fig. 9), because one or several packets are lost and none of the tiles has a copies of them. On the other hand, data upsets seem to have little influence on the chances to finish encoding. However, upsets do have an impact on the latency, especially if $p_{upset} > 0.7$ (see Fig. 7).

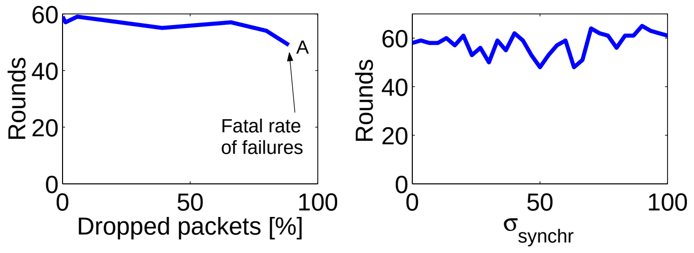

Figure #-9. Rounds needed to finish encoding

The influence of synchronization errors on the output bit rate and latency is shown in Fig. 10 (together with the jitter) and the right part of Fig. 9. Note that even very important synchronization error levels do not have a big impact on the bit rate, the jitter or the latency. This proves that our method tolerates extremely well very high levels of synchronization errors.

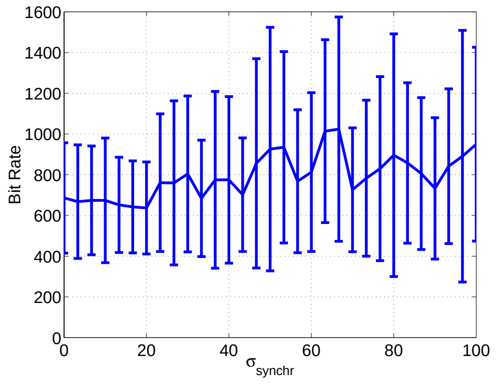

Figure #-10. Impact of synchronization errors on bit rate

These results show that stochastic communication has a very good behavior in time with respect to latency and energy dissipation. In the worst case (very unlikely for typical levels of failures in NoCs), the protocol will not deliver the packet to the destination; therefore this communication paradigm may be better suited for applications which tolerate small levels of information loss, such as our MP3 encoder. In general, real-time streaming multimedia applications have requirements that match very well the behavior of our new communication paradigm, as they can deal with small information losses as long as the bit rate is steady. The results show that our MP3 encoder tolerates very high levels of failures with a graceful quality degradation. If, however, the application requires strong reliability guarantees, these can be implemented by a higher level protocol built on top of the stochastic communication.

6. CONCLUSION

In this chapter, we emphasized the need for on-chip fault-tolerance and proposed a new communication paradigm based on stochastic communication. This approach takes advantage of the large bandwidth that is available on chip to provide the needed system-level tolerance to synchronization errors, data upsets and buffer overflows. At the same time, this method offers a low-latency, 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 interconnect design.

ACKNOWLEDGEMENTS

The authors would like to express their appreciation to Sam Kerner, who helped set up some of the simulators used in the present work. We would also like to thank Mr. Jingcao Hu for his insightful remarks on the results of our experiments.

REFERENCES

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

[2] T. Dumitraş, S. Kerner, R. Mărculescu. Towards on-chip fault-tolerant communication. In Proc. of the ASP-DAC, January 2003

[3] A. Demers, et. al. Epidemic algorithms for replicated database maintenance. In *Proc. of the ACM Symposium on Principles of Distributed Computing*, August 1987.

[4] T. Dumitraş, R. Mărculescu. On-chip stochastic communication. In *Proc. of DATE*, March 2003

[5] The LAME project. http://www.mp3dev.org/mp3/.

[6] B. Pittel. On spreading a rumor. *"SIAM Journal of Appl. Math."*, 1987.

[7] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal Multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, 1999.

[8] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking*. ACM, August 1999.

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

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

[11] L. Benini and G. De Micheli. Networks on chips: A new paradigm for systems on chip design. In *"Proc. of the 39th DAC"*, June 2002.

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

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

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

[15] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, 1993.

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

[17] D. E. Lackey et. al. Managing Power and Performance for System-on-Chip Designs using Voltage Islands. In *Proc. of the ICCAD*, November 2002.

[18] D. M. Chapiro. *Globally Asynchronous Locally Synchronous Systems*. PhD thesis, Stanford University, 1984.

[19] V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR941425, 1994.

[20] PVM: Parallel Virtual Machine. http://www.csm.ornl.gov/pvm/pvm_home.html.