Fault Tolerant Source Routing for Network-on-chip

Young Bok Kim, Yong-Bin Ki
Dept. of Electrical and Computer Engineering ,
Northeastern University, Boston, MA ,USA
ybk@ece.neu.edu

Abstract

This paper presents a new routing protocol of network-on-chip(Noc) called 'Source Routing for Noc'(SRN) for fault tolerant communication of Systems-on-chip(Soc). The proposed SRN algorithm is composed of two mechanisms of Route Discovery and Route maintenance to allow nodes to discover and maintain source routes to arbitrary destinations in Noc, and all aspects of the protocol operate entirely on-demand allowing the routing packet overhead to scale automatically based on its need. The SNR algorithm in this paper demonstrates up to 50 % more fault tolerance comparing with the conventional algorithms. This new method can be easily adapted and implemented with lower cost due to less hardware overhead in SoC that integrate a large number of communicating IP cores.

1. Introduction

As CMOS technology scales down into the deep submicron(DSM) or nano-technology domain, the fault tolerance is becoming a key issue. The new types of malfunction and failure are difficult to predict and they are very difficult to diagnose with the current Systems-on-chip(Soc) design methodologies. To reduce the cost of design, Soc and Noc must be designed with a high degree of system level fault tolerance. There are many researches on how to integrate and connect the IPs, however many issues remain open since these are very hard to address within the existing framework of CAD tools. As technology scales, it is common to see the increase of errors and faults. Crosstalk interferes with signal transmission while soft errors result in random bit-flips throughout the design [1]. Designing systems that operate in the presence of transient or permanent failures has been studied for years, but research has focused mostly on large scale systems and their interconnects. The Network-On-Chip (NoC) design paradigm has been proposed as the future of SoC design [2]. Using the NoC design methodology different IP blocks are connected by a packet-based on-chip-network. This is totally different from traditional large-scale interconnects due to the limited area resources available on the chip as well as the increased dependency on low-latency communication. Implementing traditional fault tolerant algorithms in the NoC domain is infeasible due to these area restrictions. Therefore, it is essential to develop a new routing communication techniques for fault tolerant SoCs design..

2. Related Works

The probabilistic flooding algorithm is a variant of a simple flooding protocol. In this algorithm, whenever a message is generated it will be passed to all of its neighbors with probability *p*, and will be dropped with probability *1-p*. *p* must be carefully chosen to obtain near flooding performance without sending many redundant copies of the message [5]. Given enough time every node in the mesh will have received multiple copies of the message with a high probability. This mode of communication is most susceptible to failure during its initial phase of transmission when only a limited amount of nodes needs not pass on the message in order for the message propagation to halt. To make sure that the message does not die off during this period, an initial flooding phase of two cycles is performed to ensure that all functional nodes within two hops receive a copy of the message. Implementing this algorithm in hardware requires a pseudo-random number generator to determine whether or not to replicate a given message. The algorithm and further implementation details are discussed later in the paper.

The directed flooding algorithm makes use of a NoC' s highly regular structure to direct a flood towards the destination. In this algorithm, the probability *p* of passing a message to each outgoing link is not fixed but varies based on the destination of the packet instead. The algorithm' s onset packets have a high probability of being routed towards the destination node, allowing for packets to get in the general vicinity of the destination node quickly. As the packets approach the destination node, it is desirable for them to take varied paths to mitigate the risk of all packets encountering the same defective nodes on their way to the destination. Recently the research for fault tolerant algorithms, specifically for NoC, has begun. In this paper, a new routing algorithm called Source Routing for Network-on-chip(SRN) is proposed and compared with the traditional algorithms such as probabilistic flooding algorithm[3] and directed flooding algorithm[4].

3. Source Routing for NoC

The proposed Source Routing for NoC(SRN) is a simple and efficient routing protocol designed specifically for use in Network-on-chip. The SRN allows the network to be completely self-organizing, self-configuring, and fault tolerant without the need for periodic routing management or administration. The SRN is composed of two mechanisms of *Route Discovery* and *Route Maintenance*, which work together to allow nodes to discover and maintain *source routes* to arbitrary destinations in NoC. The use of source routing allows packet routing to be trivially loop-free, avoids the need for up-to-date routing information in the intermediate nodes through which packets are forwarded, and allows nodes forwarding packets to cache the routing information in them for their own future use. All aspects of the protocol operate entirely *on-demand*, allowing the routing packet overhead to scale *automatically* to only that needed in order to react to changes in the routes currently in use

3.1 SRN Route Discovery

When a node **S** originates a new packet destined to another node **D**, it places in the header of the packet a *source route* giving the sequence of the hops that the packet should follow on its way to **D**. Normally **S** will obtain a suitable source route by searching its *Route Cache* of the routes previously learned, but if no route is found in its cache, it will initiate the Route Discovery protocol to dynamically find a new route to **D**. In this case, we call **S** the *initiator* and **D** the *target* of the Route Discovery.
Figure.1. illustrates an example of Route Discovery, in which a node **1** is attempting to discover a route to node **9**. To initiate the Route Discovery, **1** transmits a ROUTE REQUEST message as a single local broadcast packet, which is received by all neighbors. Each ROUTE REQUEST message identifies the initiator and target of the Route Discovery, and also contains a unique *request id* determined by the initiator of the REQUEST. Each ROUTE REQUEST also contains a record listing the address of each intermediate node through which this particular copy of the ROUTE REQUEST message has been forwarded.

This route record is initialized to an empty list by the initiator of the Route Discovery. When another node receives a ROUTE REQUEST, if it is the target of the Route Discovery, it returns a ROUTE REPLY message to the initiator of the Route

Discovery, giving a copy of the accumulated route record from the ROUTE REQUEST; when the initiator receives this ROUTE REPLY, it caches this route in its Route Cache for use in sending subsequent packets to this destination.

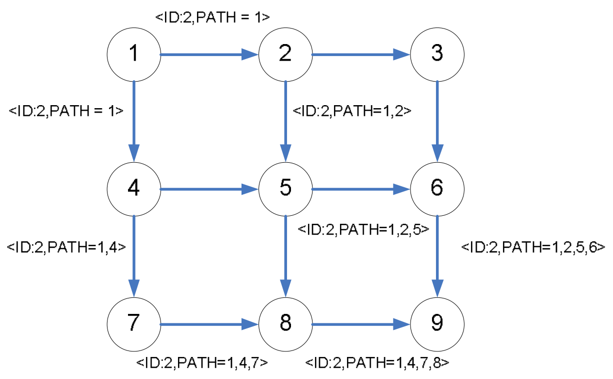


Figure 1. Route Discovery example : Node 1 is the initiator, and 9 is the target

Otherwise, if the node receiving the ROUTE REQUEST has recently seen another ROUTE REQUEST message from this initiator bearing this same request id, or if it finds that its own address is already listed in the route record in the ROUTE REQUEST message, it discards the REQUEST. Otherwise, this node appends its own address to the route record in the ROUTE REQUEST message and propagates it by transmitting it as a local broadcast packet (with the same request id). In returning the ROUTE REPLY to the initiator of the Route Discovery such as node **9** replying back to **1** in Figure 1, node **9** will typically examine its own Route Cache for a route back to **1**, and if it is found, it will be used for the source route for delivery of the packet containing the ROUTE REPLY. Otherwise, **9** may perform its own Route Discovery for target node **1**, but to avoid possible infinite recursion of Route Discoveries, it must piggyback this ROUTE REPLY on its own ROUTE REQUEST message for **1**. Node **9** could also simply reverse the sequence of hops in the route record that it tries to send in the ROUTE REPLY, and uses this as the source route on the packet carrying the ROUTE REPLY itself. This route reversal is preferred as it avoids the overhead of a possible second Route Discovery. If the node **1** has the ROUTE REPLY message, node **1** saves it in the cache. Because node **1** broadcasts the ROUTE REQUEST, multiple routes will be discovered. Node **1** sets up the route whichever comes first. Due to limited memory, node **1** only keeps maximum three ROUTE REPLY and uses the first one as a main route. If the first route is broken, node **1** chooses the second one. The node **1** can have multiple routes for alternative routes. As a result it can achieve the fault tolerant acting like spare machine.

The faulty nodes or links are filtered by the proposed SRN algorithm. On the process of route discovery, if the ROUTE REQUEST message reaches faulty node, the faulty node makes a broken ROUTE REQUESET message and broadcasts it again, which is detectable in other nodes with CRC(Cycle Redundancy Check). If the CRC finds that broken packet is received, the node will discard the packet. Thus, the ROUTE REQUEST message will not be reached to the target node and the route which contains faulty node will not be selected. Through this mechanism, the faulty routing unit can be avoided and it is possible to have the alternative routes.

3.2 SRN Route Maintenance

When a data is originated or forwarded using a source route, each node transmitting the data is responsible for confirming that the packet has been received by the next hop along the source route; the data is retransmitted (up to a maximum number of attempts) until this confirmation of receipt is received.

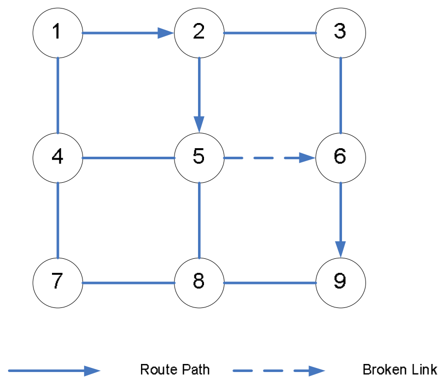


Figure 2. Route Maintenance example: Node 5 is unable to forward a packet from 1 to 9

For example, in the case of Figure 2, node **1** has originated a packet for **9** using a source route through intermediate nodes **2**, **5**, and **6**. In this case, node **1** should make sure the packet is received at **2**, node **2** should make sure the packet is received at **5**, node **5** has to make sure the packet is received at **6**, and node **6** is responsible for receipt of the packet finally at the destination **9**. If the data is retransmitted by some hops for the maximum number of times and no receipt confirmation is received, this node returns a ROUTE ERROR message to the original sender of the packet, identifying the link over which the packet could not be forwarded. For example, in Figure 2, if **5** is unable to deliver the packet to the next hop **6**, then **5** returns a ROUTE ERROR to **1**, stating that the link from **5** to **6** is currently " broken." Node **1** then removes this broken link from its cache and uses the second alternative route. In order to retransmit the packet to the destination or to send other packets to this same destination **9**, if **1** has another route to **9** in its Route Cache (for example, from additional ROUTE REPLYs from its earlier Route Discovery), it can send the packet using the new route immediately. Otherwise, it may perform a new Route Discovery for this target.

3.3 Replying to ROUTE REQUESTs using Cached Routes

A node receiving a ROUTE REQUEST searches for a route to the target of the REQUEST its own Route Cache. If a route is found, the node generally returns a ROUTE REPLY to the initiator itself rather than forwarding the ROUTE REQUEST. In the ROUTE REPLY, it sets the route record to list the sequence of hops over which this copy of the ROUTE REQUEST was forwarded and it is concatenated with its own idea of the route from itself to the target from its Route Cache. However, before transmitting a ROUTE REPLY packet that was generated using the information from its Route Cache in this way, a node must verify that the resulting route being returned in the ROUTE REPLY, after this concatenation, contains no duplicate nodes listed in the route record.

For example, Figure 3 illustrates a case where a ROUTE REQUEST for target **9** has been received by node **5**, and node **5** already has a route from itself to **9** in its Route

Cache. The concatenation of the accumulated route from the ROUTEREQUEST and the cached route from **5**' s Route Cache would include a duplicate node in passing from **5** to **9** and back to **2**. Using the caching mechanism, the route discovery time is reduced. The NoC's IP block usually communicates between nodes frequently. The probability that the route for frequently communicated block is in cache becomes high.

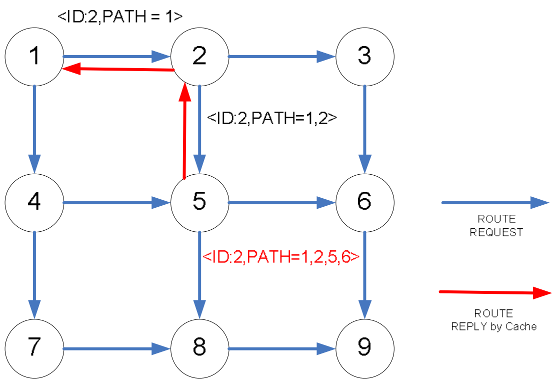


Figure 3 The ROUTE REPLY performed by intermediate node's cache.

4. Simulation

The proposed algorithm is implemented in Matlab's Stateflow, a tool for describing and simulating concurrent behavior of complex systems, and it uses a formalism defined in [6]. The network for simulation is composed of 256 routing nodes arranged as a 16 by 16 mesh. Each packet is composed of a 32-bit header and a 256-bit payload. These packets are transmitted across the network in two 144-bit flits in a store-and-forward fashion.

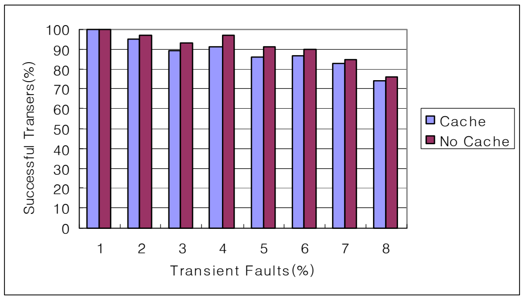
Each routing node has both input and output buffers on each port for a total of 12 packets of storage per routing node. The crossbar is a fully connected 5 x 5 design with a bit width of 144 bits. To evaluate the performance of the algorithms, three separate metrics are used: fault tolerance, latency, and implementation costs. In each simulation all non-faulty tiles communicate uniformly with all other non-faulty tiles. All message recipients were non-faulty tiles so that the algorithms are evaluated not based on how fortunate they were to pick a working destination but based on their fault-tolerance. For all simulations, an injection rate (i.e. the proportion of nodes that generates messages per iteration) of 0.05% was used. The injection rate was chosen to allow meaningful results for the flooding algorithms. Higher injection rates quickly saturate the network when the high levels of redundancy and flooding algorithms are used.

A TTL(time-to-live) of 50 hops is used for all simulations unless otherwise noted; The maximum number of hops is limited to 50 before the packet(data) is discarded when a route is not found. The choice of this value is explained in detail later.

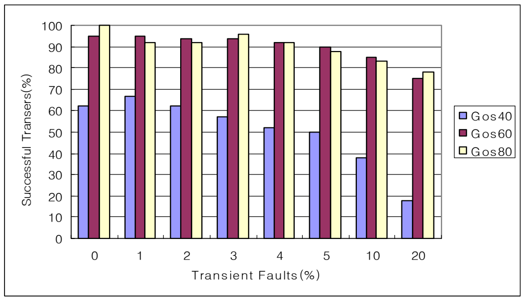
4.1 Fault Tolerance

Fault tolerance evaluates how reliably each algorithm can route messages in spite of transient and permanent faults. Transient faults are considered to cause a message disruption during each hop while permanent faults indicate the nodes that are unable to send any packets.

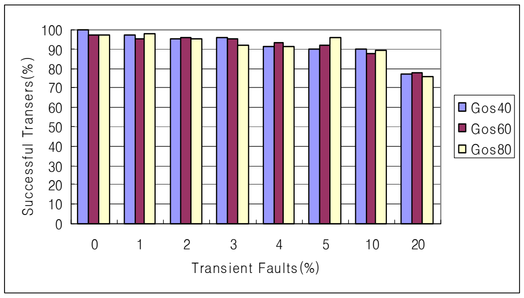
We first set the variable, the transient fault rate, which is a generic representation of transient errors that happen in the system [3]. This is implemented in the simulations by randomly discarding a fixed percentage of all packets in the network for every iteration of the algorithms.



(a) Transient fault tolerance of SRN algorithm



(b) Transient Fault tolerance of gossip flooding algorithm

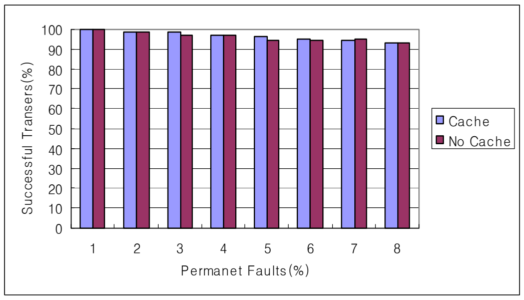


(c) Transient Fault tolerance of directed flooding algorithm

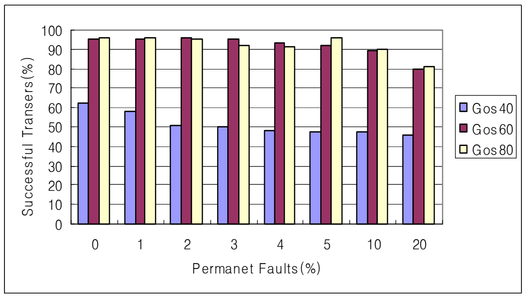
Figure 4. Fault tolerance for transient faults

Figure 4 shows the fault tolerance of the three different algorithms for transient faults such as crosstalk and soft errors. In Figure 4, (a) illustrates transient fault tolerance for the cases where cache system is used and it is not used, and (b) and (c) illustrates the transient fault tolerances for three different cases that have different passing probabilities(gossip rates, 30%, 60%, and 80%). We see that the fault tolerance of the proposed SRN(stochastic reward nets) and the directed flood are at the similar levels, while the fault tolerance of gossip flooding algorithm is slightly less than the other two algorithms. Increasing the number of redundantly sent packets significantly affects the transient fault tolerance of the SRN. With a 5% transient fault rate, a packet has about 90% chance of being intact after 14 hops. We then evaluate the effect of changing the

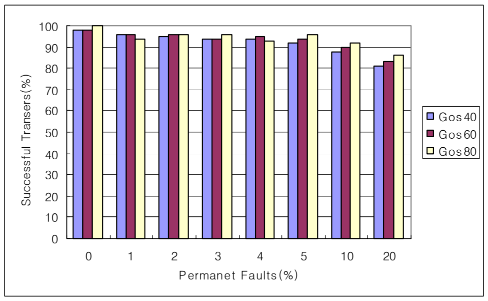
permanent fault rate. Permanent fault rate corresponds to the percent of all tiles in the NoC that are not functioning properly due to manufacturing faults.
Figure 5 shows the effect of permanently faulty nodes on successful transmission rate. In Figure 5, (a) illustrates permanent fault tolerance for the cases where cache system is used and it is not used, and (b) and (c) illustrates the permanent fault tolerances for three different cases that have different passing probabilities(gossip rates, p = 30%, 60%, and 80%). In the permanent faults, our proposed algorithm SRN shows the best successful transfer percentage. The reason for this result is that the proposed SRN algorithm has the mechanism for the permanently faulty nodes which is done during the route discovery process.



(a) Permanent fault tolerance of SRN algorithm



(b) Permanent fault tolerance of gossip flooding algorithm



(c) Permanent fault tolerance of directed flooding algorithm

Figure 5 Fault tolerance for transient faults

4.2 Latency

Latency measures how quickly a message arrives at its destination. This is measured as the number of hops that are required for a message to reach its destination. Since these algorithms utilize redundant packets, only the first time a message arrives at its destination is included in this metric.

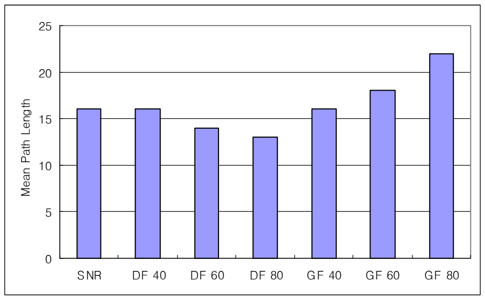


Figure 6 Comparison of latency (2% permanent and 1% transient fault rate)

We have compared the latencies of the three algorithms as shown in Figure 6. This figure indicates that the proposed SRN and directed flooding algorithms tends to have the shortest paths while the gossip flooding has a longer latency. Although SRN has setup time latency, SRN latency is reasonable due to cache mechanism. It is notable that for the parameter settings chosen for directed flooding, increasing redundancy has little effect on latency. This is due to directed flooding's tendency to push the flood towards the destination early and let the flood effect find the destination once in the general area.

4.2 Total packets

Total packets measures how many packets are produced for simulation. The Gossip flooding and directed flooding algorithms use broadcasting mechanism which produces large redundant packets.

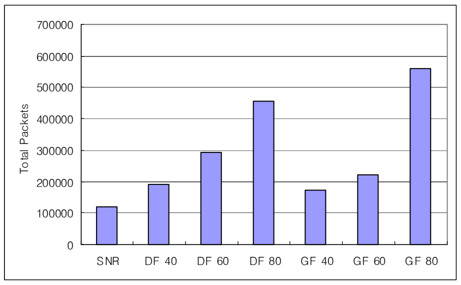


Figure 7 Total packets produced for simulation

But our proposed SRN method uses broadcast for route discovery while data transfer mechanism is unicast. As shown in Figure 7, in case of the SRN, the total packets produced for simulation is very low because SRN uses unicast mechanism. Large amount of packets usually means it consumes much power. Therefore, the proposed SRN algorithm is expected to contribute to low power design

5. Conclusions

Implementing traditional fault tolerant communication algorithms in NoC's is infeasible due to area constraints, so new approaches to fault tolerant communication must be found.

In this paper, we proposed a new fault tolerant communication algorithms, Source Routing for NoC(SRN). We find that the flooding algorithms cause significant communication overhead, which limits the feasibility of those algorithms to a very low message injection rate. We recognized this fact and tried to make a new routing mechanism that satisfies the fault tolerance and low routing overhead.

The SRN algorithm generally requires less communication overhead than the flooding algorithms due to limited number of broadcast. We have also observed that the our proposed SRN algorithm tends to be fault resistant like flooding algorithms. Further, improvements could be made for the proposed SRN algorithm by developing dynamical routing with cache of ROUTE REQUEST messages to cope with current network conditions.

The flooding algorithms, especially the directed flood, tend to be quite error resistant for higher error rates. Directed flooding also appears to operate quite well with a lower gossip rate than probabilistic flooding. Also directed flooding algorithm shows a large amount of packets.

Our simulation results indicate that flooding protocols, while simple and lightweight, rely on too many message replications to obtain their fault tolerance. On the contrary, the proposed SNR achieves the goal to minimize the routing overhead and gain fault tolerance.

The proposed SNR algorithm in this paper shows the better or at least the same fault tolerance and better effectiveness than the conventional flooding algorithms in terms of fault tolerance, performance, hardware overhead, and power saving.

6. References

[1] W. Dally and J. Poulton, Digital Systems Engineering, Cambridge University Press, 1998.

[2] L. Benini and G. D. Micheli, "Networks on chip: a new paradigm for systems on chip design", In Proceedings Design Automation and Test in Europe Conference and Exhibition, 2002, pp. 418 –419.

[3] T. Dumitras, S. Kerner, and R. Marculescu, "Towards onchip fault-tolerant communication", In Proceedings Asia and South Pacific Design Automation Conference, 2003.

[4] Pirretti, M., Link, G.M., Brooks, R.R, Vijaykrishnan, N, Kandemir, M, Irwin, M.J., "Fault tolerant algorithms for network-on-chip interconnect", In Proceedings IEEE Computer society Annual Symposium on 19-20 Feb. 2004, pp. 46 - 51

[5] L. Li, J. Halpern, and Z. Haas, "Gossip-based ad hoc routing", In Networking , IEEE/ACM Transactions on June 2006, Volume 14, Issue 3, pp. 479-491

[6] D. Harel. Statecharts, "A visual formalism for complex systems" Science of Computer Programming **8,** June 1987, pp. 231–274.