26th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing

Performability Analysis of Mesh-based NoCs Using Markov Reward Model

Jie Hou, Martin Radetzki
Chair of Embedded Systems
University of Stuttgart
Stuttgart, Germany
Email: {jie.hou,martin.radetzki}@informatik.uni-stuttgart.de

***Abstract*—Technology scaling makes it possible to implement systems with hundreds of processing cores, and thousands in the future. The communication in such systems is enabled by Networks-on-Chips (NoCs). A downside of technology scaling is the increased susceptibility to failures in NoC resources. Ensuring reliable operation despite such failures degrades NoC performance and may even invalidate the performance benefits expected from scaling. Thus, it is not enough to analyze performance and reliability in isolation, as usually done. Instead, we suggest treating both aspects together using the concept of performability and its analysis with Markov reward models. Our methodology is exemplified for mesh NoCs and transient faults but can be transferred to other topologies and fault models. We investigate how performability develops with scaling towards larger NoCs and explore the limits of scaling by determining the break-even failure rates under which scaling can achieve net performance increase.**

***Index Terms*—Performability; degradable systems; Markov models; Markov reward models; NoCs; Mesh**

I. Introduction

With the development of integration technology, transistor density has been increased drastically. It helps microprocessors move from single-core to many-core architectures. The NoC has been proposed as an interconnection network for many-core architectures. It is a highly scalable, bandwidth efficient and packet-switched network containing a certain number of routers and links [1]. Technology scaling has increased the susceptibility of its components to failures [2]. E.g. some routers could become faulty sometime after they started operation. However, such NoC can still operate with a performance loss. In this sense, it can be classified as a degradable system or a fault tolerant system. Performance analysis of such systems requires taking into account the impact of faults and the likelihood of their occurrence, i.e. reliability. This is achieved with performability [3], [4], which we apply to NoCs for the first time.

The remainder of this paper is structured as follows. Section II discusses related work relevant to the performability analysis of NoCs. In Section III, Markov reward models are introduced. Section IV explains the Markov model of a mesh-based NoC. The detailed description of our proposed reward metric can be found in Section V. Evaluation results are presented and discussed in Section VI. Section VII concludes our work.

II. Related work

In NoC literature, several works have been done related to performability analysis. In [5], performability $P(L, T)$ was defined as the probability of transmitting L useful bits during the time interval T in the presence of noise. The impacts of three error control schemes were analyzed with respect to the tradeoff between performability and energy. In [6], the definition of performability from [5] was used. They evaluated different forward error correction coding schemes from aspects of performability and energy taking account of voltage swing and noise power. It was concluded that the joint triple error correction was preferred for high performability in the case of high voltage swing and low noise power. In [7], performability was defined as the probability for transmitting a correct flit. They concluded that the Bose-Chaudhuri-Hocquenghem coding algorithm could achieve high performability while saving energy.

The related works deal with the physical network layer only and deviate from the classical performability definition. In our paper, $routers$ on a NoC are assumed to suffer from transient failures. The NoC's evolution in time can be characterized as a stochastic process. If the sojourn time in each state follows an exponential distribution, this process is a homogeneous continuous time Markov chain (CTMC) [8]–[10]. How to model a mesh-based NoC as a CTMC and use Markov reward model for performability analysis are our concern and contribution. To our knowledge, this is the first paper that analyzes NoCs based on the classical definition of performability, which is introduced in Section III.

III. Preliminaries: Markov reward model

Markov reward models are the most commonly used tools for performability analysis. Formally, they contain two parts: a CTMC with state space Ω and a reward function r where r: $\Omega \rightarrow \mathbb{R}$. Each state $i \in \Omega$ is assigned a reward. In general, the reward associated with a state denotes the performance level given by the system while it is in that state [8].

A CTMC is represented by states and transitions between states. Information about states and transition rates are described by a generator matrix $\mathbf{Q}$. Usually, a CTMC would be used for studying dynamics of a system. Many techniques and algorithms have been developed to solve long-term steady state distribution, transient state probabilities and occupancy

0-7695-6380-5/18/$31.00 ©2018 IEEE
DOI 10.1109/PDP2018.2018.00102

times of Markov chains [11]. In our work, the uniformization algorithm is utilized.

As NoC routers are assumed to suffer from transient failures, it is worth knowing how NoC behaves in the long-term perspective. Another question is its performance at a particular time instance, e.g. 100 hours after starting operation. In order to answer such questions, two definitions named long-term steady state performability and transient performability are extracted from [8], [9]. Their mathematical expressions are as follows:

- Long-term steady state performability (SSP) is defined as:

$$SSP = \sum_{i \in \Omega} \pi_i r_i \tag{1}$$

  where π_i means the long-term steady state probability of residing in state i. The equation gives the expected reward rate of the system in the long-term perspective.
- Transient performability (TP) is expressed as:

$$TP = \sum_{i \in \Omega} \pi_i(t) r_i \tag{2}$$

  where $\pi_i(t)$ denotes the probability that the system is in state i at time instance of t. This equation's meaning is the expected reward rate of the system at a certain point in time.

IV. Modeling a Mesh-based NoC as a CTMC

In order to model a NoC as a CTMC, the first step is to determine the state space. An intuitive model is to list all possible combinations of faulty routers. Each combination represents a state. In this model, faults' number and positions are both taken into account. Although this model is precise, the size of its state space grows exponentially with the NoC size. E.g. assuming a NoC of size 196 with 0 to 20 faulty routers, the size of its state space can be computed: $\sum_{x=0}^{20} \binom{196}{x} = 1.1885e+27$. As the state space is so huge, it is impossible to build its generator matrix.

Another way is from the perspective of NoC topology, which can be described mathematically using a graph that contains a vertex set and an edge set. For the NoC, a vertex denotes a router. An edge between two routers contains two physical channels. One is for sending packets and the other one is for receiving packets. Vertex degree means the number of incident edges. Routers are classified into different groups based on their vertex degrees. The actual number of operational routers from the classified groups form a state. In this way, the size of state space can be reduced tremendously. In the following section, we use this approach to construct the Markov model of a mesh-based NoC.

A. Markov model of a mesh-based NoC

As mesh topology has three different vertex degrees, vertexes can be classified into three different groups. The routers from $Group_1$ locate at the four corners. The maximum size of $Group_1$ is 4. Routers that sit in outer edges belong to $Group_2$. The other ones that locate inside a mesh are classified as $Group_3$.

The Markov model of a mesh-based NoC is illustrated in Figure 1. Each state is specified by a triple (i, j, k) indicating the number of functional routers from $Group_1$, $Group_2$ and $Group_3$. The failure and repair rates related to routers from $Group_1$ are λ_1 and μ_1. Similarly, for routers from $Group_2$ and $Group_3$ they are λ_2, μ_2 and λ_3, μ_3 respectively. The $Phase\_x$ contains such states, whose total number of faulty routers is equal to x. When a certain number of routers are faulty, the NoC's performance may become so bad that it is not suitable for fulfilling communication tasks. So, the total allowed number of faulty routers needs to be limited. It is defined as fault limit and denoted by n. A state, in which the total number of faulty routers exceeds n, is called a failure state.

There are two types of repair actions: local repair and global repair. Local repair can be thought of as fault tolerance mechanisms. Our model of local repairs assumes that there is only one repair process for each router group. E.g. state $(3, j, k-1)$ moves back to state $(3, j, k)$ with a repair rate of μ_3. And this state can also go back to state $(4, j, k-1)$ with a transition rate of μ_1. Global repair with a rate of μ can be thought of as a reset and restores the NoCs from any failure state to the initial state.

For a mesh of size 196 (14×14), if the fault limit is 20 routers, the size of its Markov model's state space is 1055, significantly reduced compared to $1.1885e+27$.

B. Monte Carlo method

The number of combinations of faulty routers in some states may be huge. Additionally, each combination affects NoC's performance differently. Therefore, a performance metric needs to be computed for each combination, which results in a huge amount of computation. Such an approach is time consuming. The Monte Carlo method is used for reducing computation time. It works as follows:

1) Randomly sample a new combination of faulty routers and inject it into the evaluated network.
2) Compute the performance metric based on the combination.
3) Average all the computed performance metrics and compare the new average value with previous one. If the difference is smaller than the specified precision and the cumulative number of samples is greater than the specified minimum sample size, the process will be stopped. Otherwise, repeat these three steps until the metric converges.

V. Communication time as a performance metric

To be able to sample a large number of combinations, efficient estimation of NoC's performance is a new challenge.

A. Definition of communication time

In our model, a node is defined as a processing core that is only connected to one router on a NoC. The router and

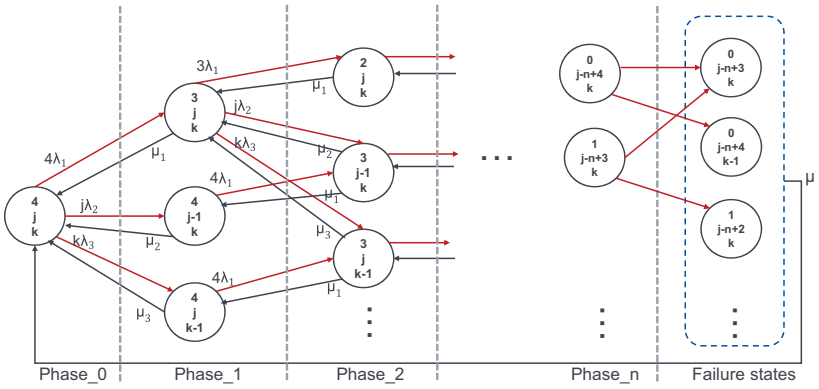

Fig. 1. Markov model of a mesh-based NoC

its connected node share the same index. E.g. node 0 is connected to router 0. All symbols used in the description are summarized in Table I.

TABLE I
LIST OF SYMBOLS

Symbol	Description
Δ	Set of router ids
$f_{i\to j}$	An end-to-end communication flow from node i to node j
Π_i	The i^{th} communication round
$n(\Pi_i)$	The number of successfully delivered packets in the i^{th} communication round
t_{inj}	Injection delay from a source node to a source router
t_{ej}	Ejection delay from a destination router to a destination node
$\sigma(R_i \to R_j)$	Network latency from router i to router j
$\sigma(f_{i\to j})$	Latency of an end-to-end communication flow from node i to node j
$\sigma(\Pi_i)$	Latency of the i^{th} communication round
T	Communication time
N	The specified number of packets for which communication time is determined
$C_{R_i\to R_j}$	Set of channels that make up a path from router i to router j
$\lvert C_{R_i\to R_j}\rvert$	Hop count from router i to router j
b	Physical channel bandwidth
b_S	Shared channel bandwidth
t_R	Routing computation delay of a head flit
t_S	Switching delay of a flit
t_{router}	Time to traverse a router. For a head flit, $t_{router} = t_R + t_S$. For body and tail flits, $t_{router} = t_S$
$t_{channel}$	Time for a flit to traverse a channel
c_B	Bottleneck channel that has minimum shared channel bandwidth

1) End-to-end communication flow: The communication between any two different nodes connected to routers $i, j \in \Delta$ is defined as an end-to-end communication flow $f_{i\to j}$, where i denotes the source and j denotes the sink.

2) Communication round: A communication round $\Pi = \{f_{i\to j} : i, j \in \Delta\}$ is a set of end-to-end communication flows, in which all source nodes are different and every node delivers one packet to its destination that is selected based on the utilized traffic pattern. All packets have the same number of flits. Moreover, it is assumed that all source nodes send packets at the same time. A communication round in which the set of sources is a strict subset of Δ, is called a partial communication round. Similarly, if the set of sources in a communication round is equal to Δ, it is defined as a full communication round.

3) Latency of an end-to-end communication flow: It is defined as the time span from the moment a packet is generated at a source node to the time when it is delivered to a destination node. It is the sum of injection delay, network latency and ejection delay. Formally, it is expressed as follows:

$$\sigma(f_{i\to j}) = t_{inj} + \sigma(R_i \to R_j) + t_{ej} \tag{3}$$

Injection delay is defined as the time, which is taken by a packet to pass through the channel from the source node to the source router. Network latency is the time that a packet takes to traverse from the source router to the destination router. Ejection delay is the time that a packet needs for crossing the channel between the destination router and the destination node. Additionally, the queuing delay in network is zero because packets are injected into an empty network.

4) Latency of a communication round: It is the time that a NoC needs for delivering all packets in this round. It is determined by the maximum latency of all involved flows.

$$\sigma(\Pi_i) = \max_{\forall f \in \Pi_i} \sigma(f) \tag{4}$$

5) Communication time: It is the sum of the latencies of all utilized full communication rounds for successfully delivering the specified number of packets N. If the number of totally needed communication rounds is M, the communication time is defined as follows:

$$T = \sum_{i=1}^{M} \sigma(\Pi_i),\ where \sum_{i=1}^{M-1} n(\Pi_i) < N\ and \sum_{i=1}^{M} n(\Pi_i) \geq N \tag{5}$$

The usage of many full communication rounds enables us to assess a NoC's performance thoroughly, because different source-destination pairs are used in each round.

In order to calculate communication time, the problem of computing the latency of an end-to-end communication flow needs to be solved first. As different flow control methods may produce different network latencies, wormhole-switched flow control method is assumed in our model. In this method, each message is divided into packets, and each packet is further divided into a fixed number of flits. A packet contains a head flit, a tail flit and a number of body flits. To facilitate constructing the model, the following assumptions are made:

1) A port of a router contains one input unit and one output unit. Each input unit consists of only one first-in-first-out (FIFO) buffer, which is assumed to be large enough to store incoming flits. Therefore, no flits are dropped because of the limited input buffer size. All routers in the network need the same time to make routing computation and switching.
2) Physical channels have the same bandwidth b. If the time a flit traverses a channel is $t_{channel}$, t_{inj} and t_{ej} are equal to $t_{channel}$.

B. End-to-end communication flow latency with one flow in NoC

The network latency of a wormhole switched packet contains two parts: the latency of the head flit and the latency of the subsequent flits. The latency of the head flit is determined by router delay t_{router}, channel delay $t_{channel}$ and hop count $|C_{R_i \to R_j}|$ between the source and destination routers. A router uses the head flit for routing computation. Then it transmits the head flit to next router. Therefore router delay is composed of routing computation and switching for the head flit. Other subsequent flits follow the head flit through the NoC in a pipelined manner. Their latencies are determined by their size and maximum delay from switching and channel [12].

In general, the network delay of an m-flit packet is calculated as follows:

$$\begin{aligned}\sigma(R_i \to R_j) = [(|C_{R_i \to R_j}| + 1) \times (t_R + t_S) \\ +(|C_{R_i \to R_j}|) \times t_{channel}] \\ +[\max(t_S, t_{channel}) \times (m-1)]\end{aligned} \tag{6}$$

Equation (3) of latency of an end-to-end communication flow is rewritten as follows:

$$\begin{aligned}\sigma(f_{i \to j}) = 2 \times t_{channel} + \sigma(R_i \to R_j) \\ = [(|C_{R_i \to R_j}| + 1) \times (t_R + t_S) \ + (|C_{R_i \to R_j}| + 2) \times t_{channel}] \\ +[\max(t_S, t_{channel}) \times (m-1)]\end{aligned} \tag{7}$$

C. End-to-end communication flow latency with multiple flows in NoC

If more than one end-to-end communication flow sends packets at the same time, they may overlap at some channels, which leads to network contention.

1) Shared channel bandwidth: In order to consider the effect of network contention, we introduce a new concept named shared channel bandwidth. Figure 2 depicts a communication round consisting of three flows: $f_{3\to1}$, $f_{5\to1}$ and $f_{7\to1}$. We make two assumptions for this network: it uses XY routing and each channel's physical bandwidth is 1 $flit/cycle$. All these flows arrive at Router R_4 at the same time and want to use its south output channel c_{18}. Two different arbitrations can be used to solve their conflicts: winner-takes-all and interleaving. In both cases, the east output channel needs the same total time for transmitting the three flows. The effect is that this channel's bandwidth is shared by them. For each flow, the shared channel bandwidth of c_{18} is $\frac{1}{3}$ $flit/cycle$. Intuitively, if a channel with physical bandwidth b is used by n flows, its shared channel bandwidth is computed as:

$$b_S = \frac{b}{n} \tag{8}$$

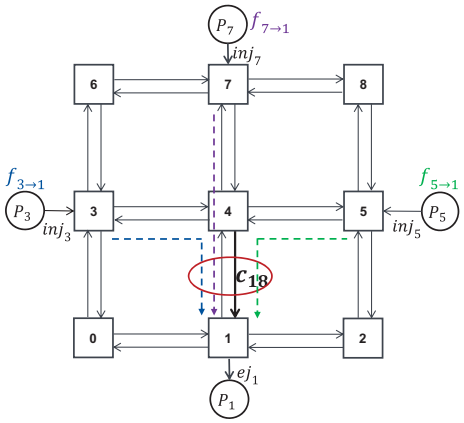

Fig. 2. Flows arrive at Router R_4 at the same time

2) Refined computation of shared channel bandwidth: Results from Equation (8) are pessimistic in some cases. Figure 3 illustrates another communication round containing three flows: $f_{3\to9}$, $f_{4\to13}$, and $f_{7\to9}$. XY routing is used on this network, and every channel has same physical bandwidth 1 $flit/cycle$. All these three flows use channel c_{23}, whose source router is R_5 and target router is R_9. Therefore, the shared channel bandwidth of c_{23} is $\frac{1}{3}$ $flit/cycle$. However, the hop counts from R_3, R_4 and R_7 to R_5 are 3, 1 and 2 respectively. The header flits of these flows arrive at R_5 at different time. In the beginning, $f_{4\to13}$ uses c_{23} without contention. When flits of $f_{7\to9}$ arrive at R_5, $f_{4\to13}$ and $f_{7\to9}$ have to compete for c_{23}. After flits of $f_{3\to9}$ arrive at R_5, these three flows have contention at c_{23} for some time. In this case, $f_{7\to9}$ and $f_{3\to9}$ contribute only parts of their flits to contention on c_{23}. It means the shared channel bandwidth of c_{23} should be larger than $\frac{1}{3}$ $flit/cycle$.

In order to refine calculation of a channel's shared bandwidth, the positions of flows' source routers that use the channel should be taken into account. Algorithmic description of our proposed refinement procedure is given in Algorithm 1, where:

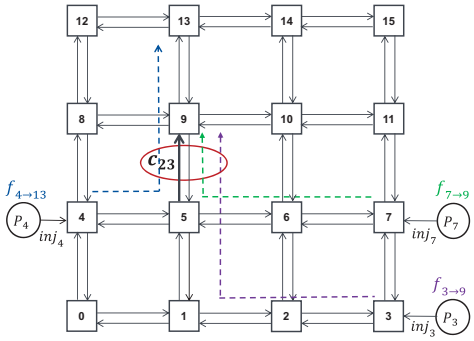

Fig. 3. Flows arrive at Router R_5 at different time

- c – Channel, whose shared channel bandwidth will be refined
- $\Lambda(c)$ – Set of end-to-end communication flows in a communication round uses channel c
- $|\Lambda(c)|$ – Cardinality of $\Lambda(c)$
- sR_c – Source router of channel c
- sR_f – Source router of end-to-end communication flow f
- nF_f – Number of flits that flow f contributes to the contention on channel c
- m – Number of flits a packet contains
- $minHop$ – Minimum hop count
- $dictHop$ – A dictionary storing flows and their corresponding hop counts and flow is the key
- $dictHop[f]$ – Hop count from the source router of flow f to the source router of channel c: $|C_{SR_f \to SR_c}|$
- $sumProp$ – Sum of flows' effective proportions that use channel c

This refinement procedure is performed only for channels that are used by more than one end-to-end communication flow. It mainly does three jobs. First, for each flow f, it computes the hop count from its source router to the source router of the channel c and stores the calculated result into a dictionary structure ($Lines\ 6-9$). After that, it finds the minimum value from these calculated hop counts ($Line\ 10$). Its second job is to compute the sum of these flows' effective proportions ($Lines\ 11-16$). For each flow f, the number of flits it contributes to the contention on channel c is estimated as follows:

$$nF_f = m - |dictHop[f] - minHop| \tag{9}$$

If the computed result is negative, it means this flow's source router locates too far from channel $c's$ source router. This flow does not contribute any flit to the contention on channel c. In another word, $sumProp$ can be thought as the effective number of flows using channel c. Finally, channel $c's$ shared bandwidth is computed at $line\ 17$.

We refine channel $c'_{23}s$ shared bandwidth using the proposed procedure. If m is assumed as 5, then $b_S(c_{23})$ is equal to $\frac{1}{2.4}\ flit/cycle$, which is greater than $\frac{1}{3}\ flit/cycle$.

Algorithm 1 Procedure of refining shared channel bandwidth

```
1:  procedure RefineSCB(c, Λ(c), m)
2:      if |Λ(c)| > 1 then
3:          minHop = 0, sumProp = 0
4:          Let dictHop be a new dictionary
5:          sR_c = GetSrcRouter(c)
6:          for ∀f ∈ Λ(c) do
7:              sR_f = GetSrcRouter(f)
8:              dictHop[f] = CalcHopCount(SR_c, SR_f)
9:          end for
10:         minHop = GetMinHop(dictHop)
11:         for ∀f ∈ Λ(c) do
12:             nF_f = CalcNumFlits(f)
13:             if nF_f ≥ 0 then
14:                 sumProp += nF_f / m
15:             end if
16:         end for
17:         b_S(c) = b / sumProp
18:     else
19:         b_S(c) = b
20:     end if
21: end procedure
```

3) Estimation of end-to-end communication flow latency with network contention: A flow $f_{i\to j}$ uses a set of channels $C_{R_i \to R_j}$ to deliver its packet. These channels have their own shared channel bandwidths under network contention. The time the head flit needs to traverse them is summed up channel by channel: $\sum_{\forall k \in C_{R_i \to R_j}} \frac{1}{b_S(k)}$. The bottleneck channel c_B is used to represent the channel, which has the minimum shared channel bandwidth: $\min_{\forall k \in C_{R_i \to R_j}} b_S(k)$. Overall latency of the flit pipeline is determined by the larger value of switching delay and bottleneck channel delay. As network contention of a communication round exists only in channels between routers, t_{inj} and t_{ej} are still equal to $t_{channel}$. Equation (7) is modified based on these changes to estimate the end-to-end communication flow latency under network contention:

$$\begin{aligned}\sigma(f_{i\to j}) = [(|C_{R_i \to R_j}| + 1) \times (t_R + t_S) + \\ \sum_{\forall k \in C_{R_i \to R_j}} \frac{1}{b_S(k)} + 2 \times t_{channel}] + \\ [max(t_S, \frac{1}{b_S(c_B)}) \times (m-1)]\end{aligned} \tag{10}$$

VI. Evaluation

A. Evaluation of proposed equation and refinement procedure

For the purpose of experiments and evaluation, we implemented a program named *fnoc* in C++ to compute the latency of a communication round for mesh NoCs. In the first step, it computes routes for end-to-end communication flows in the provided communication round. After that, each flow knows which channels it passes through. And each channel knows which flows use it. Then it computes the shared channel

bandwidth for each flow based on the proposed refinement procedure. In the last step, the latency of each flow could be computed using the proposed Equation (10). The maximum value is the latency of this communication round.

In order to validate accuracy of our proposed method, we compare it with a cycle-accurate NoC simulator named *worm_sim* [13]. We extended *worm_sim* with fault injection. If a router suffers from faults, it cannot send and receive flits. Its directly connected neighbors know its status. If a packet cannot be further delivered, it is discarded and not counted as successfully delivered packet. Furthermore, we set input buffer size as 1000 flits in order to make sure it is large enough for storing incoming flits.

We use following parameters in both programs:

- Routing delay (t_R): 2 cycles
- Switching delay (t_S): 1 cycle
- Physical channel bandwidth (b): 1 flit/cycle
- Packet size: 20 flits
- Routing function: XY routing

Totally, four different test cases were carried out for meshes of varying sizes: 36 (6×6), 64 (8×8), 100 (10×10), 144 (12×12) and 196(14×14):

- Partial communication round containing only one flow under fault-free or fault condition
- Full communication round under fault-free or fault condition

All experiments were carried out on the same host machine equipped with Intel Core Duo processor at 2.4 GHz and 12 GB RAM.

In the case of fault-free situation, 1000 communication rounds were first generated for each mesh NoC, to be simulated with *worm_sim* and *fnoc*. Average latencies and simulation times from both programs for different size mesh NoCs are listed in Figure 4 and Table II respectively. As can be seen, the estimated latencies from *fnoc* for partial communication rounds are almost same as results from *worm_sim*. In addition, our program's accuracy for estimating latencies of full communication rounds is in the range from 93.41% to 98.62%. Of particular interest is the average speedup of 69.78 that we achieve in comparison to *worm_sim*.

TABLE II
SIMULATION TIME COMPARISON UNDER FAULT-FREE CONDITION

Mesh	Partial round (ms)		Full round (ms)	
	worm_sim	fnoc	worm_sim	fnoc
36	0.802	0.011	6.934	0.123
64	1.481	0.019	15.021	0.285
100	2.483	0.029	27.846	0.536
144	3.583	0.042	47.106	0.894
196	5.627	0.052	73.594	1.360

In the case of fault situation, first 500 different fault combinations were generated for each mesh. The number of faulty routers is in the range from 1 router to 10 percent of total number of routers. For each fault combination, 100 communication rounds were generated and then used in both

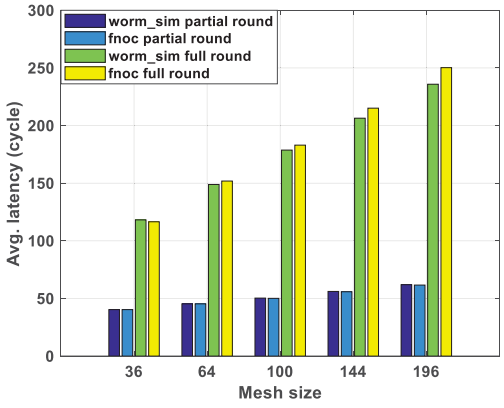

Fig. 4. Average latency comparison under fault-free condition

programs. Thus, totally 50,000 communication rounds were evaluated for each mesh. Average latencies and simulation time from both programs are given in Figure 5 and Table III. Similar to fault-free cases, our program achieves an average speedup of 78.38. Furthermore, it achieves accuracy in the range from 92.08% to 99.55% for full communication rounds. For partial communication rounds, it provides nearly the same results as offered by *worm_sim*.

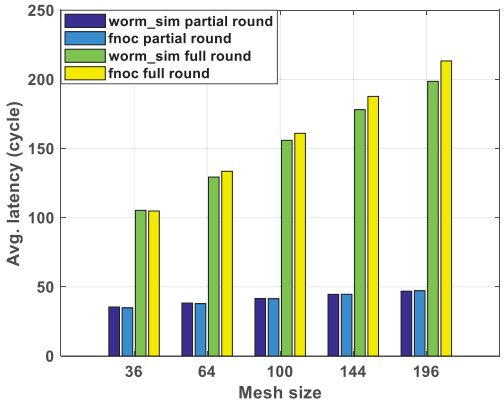

Fig. 5. Average latency comparison under fault condition

Under both fault-free and fault conditions, the latency from *fnoc* for a full communication round is slightly overestimated in comparison to *worm_sim*. In another word, we estimate NoC's performance safely and conservatively.

B. Performability evaluation

1) Evaluation configuration: In order to evaluate a mesh NoC's performability, we implemented a program in Matlab to

TABLE III
SIMULATION TIME COMPARISON UNDER FAULT CONDITION

Mesh	Partial round (ms)		Full round (ms)	
	worm_sim	fnoc	worm_sim	fnoc
36	0.635	0.007	5.749	0.099
64	1.244	0.013	11.996	0.238
100	1.997	0.021	21.283	0.358
144	3.042	0.029	35.676	0.579
196	4.255	0.039	54.436	0.926

generate the NoC's Markov model as defined in Section IV-A and compute each state's long-term and transient probabilities. In this paper, we set the fault limit as 10 percent of total number of routers. Table IV provides the information of states for different size meshes. As the mesh's size increases, its Markov state space grows, but remains tractable.

TABLE IV
FAULT LIMIT AND STATES INFORMATION FOR MESHES WITH DIFFERENT SIZES

Mesh	Fault limit	No. of total states	No. of valid states
36	4	55	35
64	7	145	110
100	10	280	230
144	15	605	530
196	20	1055	955

We extended our program *fnoc* to compute the communication time for each valid state. The procedure of computing communication time works as follows:

1) A full communication round is generated and its traffic pattern is uniform random.
2) Its latency is computed and added up.
3) Its successfully delivered packets are accumulated. If a targeted number of successfully delivered packets is reached, then this procedure is finished. Otherwise, repeat these three steps.

The communication time is equal to the accumulated latencies.

For states that represent faults, the above procedure is repeated for all fault combinations if their number does not exceed $10,000$. Communication time is averaged over these repetitions and represents the state's performance metric. For states that represent more than $10,000$ fault combinations, their performance metrics are computed using the Monte Carlo method described in Section IV-B. For this case, the specified minimum sample size is $10,000$.

We still use parameters listed in Section VI-A for the extended *fnoc*. In addition, two new parameters are added:

- Accuracy of Monte Carlo method: 0.001
- Targeted number of Packets need to be successfully delivered: 5000

As we compare performabilities of different size meshes, for the purpose of fairness, we assume all routers have same failure and repair rates with value $0.001\ h^{-1}$ and $0.02\ h^{-1}$ respectively. The global repair rate is set as $0.03\ h^{-1}$ for all meshes.

2) Results analysis: Table V provides communication times that meshes with different sizes need to successfully deliver 5000 packets under the fault-free condition. As the mesh's size increases, the communication time decreases because a larger mesh needs fewer communication rounds. E.g. a mesh of size 36 needs 139 communication rounds. In contrast, a 196 node mesh needs only 26 communication rounds. The number of routers in the mesh with size of 196 is about 5.44 times larger than that of the mesh with size 36. But its communication time is not 5.44 times as fast as the mesh of size 36. The large size mesh enables more end-to-end communication flows. However, as the network size is scaled up, the average hop counts grow too, which results in increased communication latency.

TABLE V
COMMUNICATION TIME COMPARISON OF DIFFERENT SIZE MESHES FOR SUCCESSFULLY DELIVERING 5000 PACKETS

Mesh	Communication time (cycle)
36	16319.40
64	11754.40
100	9203.85
144	7631.92
196	6218.27

If a mesh runs in the fault-free state, it has minimum communication time. In order to display performances in a normalized way, the communication time of fault-free state is defined as *base time*. The reward rate associated with an operational state (i, j, k) is defined as follows:

$$r_{(i,j,k)} = \frac{Base\ time}{Communication\ time\ of\ state\ (i,j,k)} \quad (11)$$

A zero reward rate is assigned to each non-operational state. The base times for meshes are listed in Table V. Table VI illustrates their long-term performabilities, which were computed using Equation (1). As can be seen, the long-term performability decreases as the size of a mesh increases. In other words, the large size mesh loses more performance than small ones. E.g. performability of a mesh of size 36 is 24.34% higher than that of a mesh of size 196. The reason is that as the size of mesh-based NoCs increases the long-term residing probability in valid states decreases, as displayed in Table VII. A large size mesh has more numbers of nodes in $Group_2$ and $Group_3$ than a small size one. Many states in the Markov model of the large size mesh move to the states in next phase with great transition rates. That is the reason for which the large size mesh has smaller residing probability in valid states. However, from the perspective of long-term communication time, the large size mesh is still preferred.

Figure 6 illustrates the transient performability of various size meshes. As can be seen, a mesh of large size needs more time to reach stable performance. E.g. a mesh with size 196 after 700 hours has a smooth line. However, a mesh of size 36 has a stable performance after 200 hours.

Until now, we assume all routers have the same failure rates. Actually, routers on large size NoCs have higher failure

TABLE VI
LONG-TERM PERFORMABILITIES AND COMMUNICATION TIMES OF MESHES WITH DIFFERENT SIZES

Mesh	Performability	Communication time (cycle)
36	0.7748	21063.88
64	0.6881	17081.40
100	0.6258	14708.51
144	0.5788	13184.74
196	0.5314	11701.89

TABLE VII
LONG-TERM RESIDING PROBABILITIES IN VALID AND FAILURE STATES

Mesh	Long-term residing probability	
	Valid states	Failure states
36	0.9240	0.0760
64	0.8883	0.1117
100	0.8424	0.1576
144	0.8263	0.1737
196	0.8085	0.1915

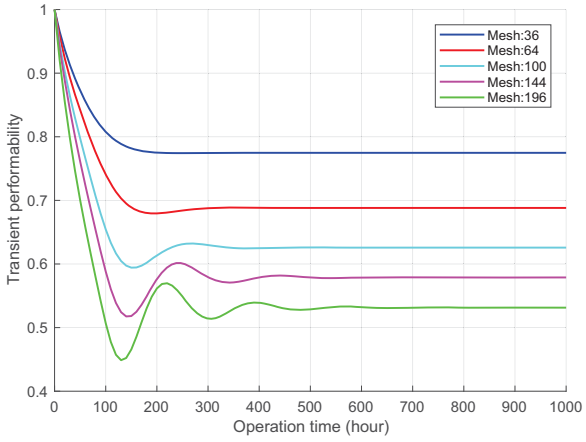

Fig. 6. Transient performabilities of meshes with different sizes

rates due to higher integration density. It is worth determining their break-even failure (BEF) rates under which they can still bring net performance increase. In our work, the long-term communication time of mesh with size 36 under the failure rate $0.001\ h^{-1}$ is used as the reference. In order to find BEF rate, we developed a script, which increases failure rate by an interval of $0.00001\ h^{-1}$ at each step until the corresponding long-term communication time is greater than or equal to the reference value. Table VIII shows the determined BEF rates. As the mesh's size grows, its BEF rate increases as well. E.g. BEF rate of a mesh with size 196 is $0.00268\ h^{-1}$ higher than that for a 64 node mesh.

VII. CONCLUSION

In this paper, we present how to analyze performability of mesh-based NoCs using Markov reward models. First, the NoC is modeled as a CTMC. Then a performance metric named communication time is introduced to represent the performance it can provide in a state. In addition, we present a method to quickly compute the proposed performance metric. From the evaluation results, we make following conclusions:

- We are able to estimate the latency of a communication round on average $78\times$ faster with accuracy of 92% or better, allowing us to investigate performabilities for meshes of bigger sizes.
- Under assumption that failure, repair and global repair rates are $0.001\ h^{-1}$, $0.02\ h^{-1}$ and $0.03\ h^{-1}$, a 196 node mesh has long-term performability of 0.5314, which is 24.34% smaller than that of a 36 node mesh, because the 196 node mesh has smaller long-term residing probability in valid states. In terms of transient performability, a 196 node mesh needs 500 hours longer than a 36 node mesh to reach the stable performance. With increasing size, the mesh suffers from performability loss.
- The scaling towards larger NoCs is possible to bring net performance increase, but failure rate growth must be limited.

TABLE VIII
BEF RATES FOR MESHES WITH DIFFERENT SIZES

Mesh	BEF rate (h^{-1})
64	0.00177
100	0.00273
144	0.00362
196	0.00445

REFERENCES

[1] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Design Automation Conference, 2001. Proceedings*. IEEE, 2001, pp. 684–689.

[2] M. Radetzki, C. Feng, X. Zhao, and A. Jantsch, "Methods for fault tolerance in networks-on-chip," *ACM Computing Surveys (CSUR)*, vol. 46, no. 1, p. 8, 2013.

[3] J. F. Meyer, "Models and techniques for evaluating the effectiveness of aircraft computing systems," 1978.

[4] ——, "On evaluating the performability of degradable computing systems," *IEEE Transactions on computers*, no. 8, pp. 720–731, 1980.

[5] A. Ejlali, B. M. Al-Hashimi, P. Rosinger, S. G. Miremadi, and L. Benini, "Performability/energy tradeoff in error-control schemes for on-chip networks," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 18, no. 1, pp. 1–14, 2010.

[6] D. M. Zamzam, M. A. A. El Ghany, and K. Hofmann, "Performability of error control schemes for noc interconnects," in *NORCHIP, 2012*. IEEE, 2012, pp. 1–5.

[7] A. A. Salem, M. A. A. El Ghany, and K. Hofmann, "Performability measurement of coding algorithms for network on chip," in *Electronics, Circuits, and Systems (ICECS), 2013 IEEE 20th International Conference on*. IEEE, 2013, pp. 691–694.

[8] K. S. Trivedi, E. C. Andrade, and F. Machida, "Combining performance and availability analysis in practice." *Advances in Computers*, vol. 84, pp. 1–38, 2012.

[9] R. Smith, K. S. Trivedi, and A. Ramesh, "Performability analysis: measures, an algorithm, and a case study," *IEEE Transactions on Computers*, vol. 37, no. 4, pp. 406–417, 1988.

[10] B. R. Haverkort, "Markovian models for performance and dependability evaluation," *Lecture notes in computer science*, vol. 2090, pp. 38–83, 2001.

[11] E. d. S. e Silva and H. R. Gail, "Performability analysis of computer systems: from model specification to solution," *Performance Evaluation*, vol. 14, no. 3, pp. 157–196, 1992.

[12] J. Duato, S. Yalamanchili, and L. M. Ni, *Interconnection networks: an engineering approach*. Morgan Kaufmann, 2003.

[13] worm_sim, "Cycle accurate noc simulator," http://www.ece.cmu.edu/~sld/software/index.php, 2005, [Online; accessed 20-July-2017].