Chapter 2
A Heuristic Framework for Designing and Exploring Deterministic Routing Algorithm for NoCs

Abbas Eslami Kiasari, Axel Jantsch, and Zhonghai Lu

Abstract In this chapter, we present a system-level framework for designing minimal deterministic routing algorithms for Networks-on-Chip (NoCs) that are customized for a set of applications. To this end, we first formulate an optimization problem of minimizing average packet latency in the network and then use the simulated annealing heuristic to solve this problem. To estimate the average packet latency we use a queueing-based analytical model which can capture the burstiness of the traffic. The proposed framework does not require virtual channels to guarantee deadlock freedom since routes are extracted from an acyclic channel dependency graph. Experiments with both synthetic and realistic workloads show the effectiveness of the approach. Results show that maximum sustainable throughput of the network is improved for different applications and architectures.

2.1 Introduction

Thanks to high performance and low power budget of ASICs (application specific integrated circuits), they have been common components in the design of embedded systems-on-chip. Advances of semiconductor technology facilitate the integration of reconfigurable logic with ASIC modules in embedded systems-on-chip. Reconfigurable architectures are used as new alternatives for implementing a wide range of computationally intensive applications, such as DSP, multimedia and computer vision applications [1]. In the beginning of the current millennium, network-on-chip (NoC) emerged as a standard solution in the on-chip architectures [10, 11].

A.E. Kiasari (✉) • A. Jantsch • Z. Lu
KTH Royal Institute of Technology, Stockholm, Sweden
e-mail: kiasari@kth.se

In network-based systems, the performance of the communication infrastructure is critical, as it can represent the overall system performance bottleneck. The performance of networks depends heavily on the routing algorithm effectiveness, since it impacts all network metrics such as latency, throughput, and power dissipation.

Routing algorithms are generally categorized into deterministic and adaptive. A deterministic routing algorithm is oblivious of the dynamic network conditions and always provides the same path between a given source and destination pair. In contrast, in adaptive routing algorithms, besides source and destination addresses, network traffic variation plays an important role for selecting channels to forward packets. However, adaptive routing may cause packets to arrive out-of-order since they may be routed along different paths. The re-order buffers needed at the destination for ordering the packets impose large area and power on system [18]. Deterministic routers not only are more compact and faster than adaptive routers [5], but also guarantee in-order packet delivery. Therefore, it is not surprising that designers would like to use deterministic routing algorithms in the NoCs which suffer from limited silicon resources. However, in deterministic routing a packet cannot use alternative paths to avoid congested channels along its route; this leads to degraded performance of the communication architecture at high levels of network throughput.

A well-designed routing algorithm utilizes the network resources uniformly as much as possible and avoids the congested channels, even in the presence of non-uniform traffic patterns, which are usual in the embedded systems. In this chapter, we propose a system-level Latency-Aware Routing (LAR) framework for designing minimal deterministic routing algorithms for network-based platforms. Especially, LAR is appropriate for reconfigurable embedded systems-on-chip which host several applications with high computational requirements and static workloads. To the best of our knowledge, the proposed framework is the first one to deal with traffic burstiness. Before the execution of a new application, the routing tables are configured with pre-computed routes, as well as other components in the system. After selecting the route and adding it to the packet, no further time is needed on routing at the intermediate nodes along the path. Due to advantages of table-based routing, it is one of the most widely used routing methods for implementing deterministic routing algorithm, e.g., IBM SP1 and SP2 [5].

LAR uses a recently proposed analytical model in [14] to calculate the average packet latency in the network. The results obtained from simulation experiments confirm that the proposed routing framework can find efficient routes for various networks and workloads.

The rest of the chapter is organized as follows. We start by reviewing previous studies in Sect. 2.2. The proposed heuristic framework is proposed in Sect. 2.3. Experimental results in Sect. 2.4 show that our proposed approach can improve the system performance. Finally, concluding remarks are given in Sect. 2.5.

2.2 Related Work

Turn model for designing partially adaptive routing algorithms for mesh and hypercube networks was proposed in [9]. Prohibiting minimum number of turns breaks all of the cycles and produces a deadlock-free routing algorithm. Turn model was used to develop the Odd-Even adaptive routing algorithm for meshes [4]. This model restricts the locations where some turns can be taken so that deadlock is avoided. In comparison with turn model, the degree of routing adaptivity provided by the Odd-Even routing is more even for different source-destination pairs.

DyAD routing scheme, which combines deterministic and adaptive routing, is proposed in [12] for NoCs, where the router works in deterministic mode when the network is not congested, and switches to adaptive mode when the network becomes congested. In [23] the authors extend routers of a network to measure their load and to send appropriate load information to their direct neighbors. The load information is used to decide in which direction a packet should be routed to avoid hot-spots. Recently, the authors in [19] present APSRA, a methodology to develop adaptive routing algorithms for NoCs that are specialized for an application or a set of concurrent applications. APSRA exploits the application-specific information regarding pairs of cores that communicate and other pairs that never communicate in the NoC platform to maximize communication adaptivity and performance.

Since all of these approaches are based on adaptive routing, they suffer from out-of-order packet delivery. Our proposed routing framework overcomes this problem while it minimizes the average packet latency across the network.

An application-aware oblivious routing is proposed in [14] that statically determines deadlock-free routes. The authors presented a mixed integer-linear programming approach and a heuristic approach for producing routes that minimize maximum channel load. However, in case of realistic workload, they did not study the effect of task mapping on their approach. Also, we have addressed the congestion-aware routing problem in [15]. With the analysis technique, we first estimated the congestion level in the network, and then embedded this analysis technique into the loop of optimizing routing paths so as to find deterministic routing paths for all traffic flows while minimizing the congestion level in the network. Since this framework cannot capture the traffic burstiness, in this work we utilize an analytical model [14] to deal with traffic burstiness.

2.3 LAR Framework

The LAR framework consists of five steps as its flowchart is shown in Fig. 2.1. At first, we represent the architecture and application using *topology graph* (TG) and *communication graph* (CG), respectively. Then we construct the *channel dependency graph* (CDG) based on TG and CG. In the third step, an acyclic CDG is extracted by deleting some edges from CDG to guarantee the deadlock freedom.

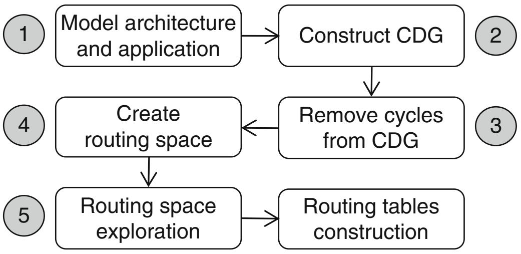
1
Model architecture and application
Construct CDG
2
3
Remove cycles from CDG
4
Create routing space
5
Routing space exploration
Routing tables construction

Fig. 2.1 The flowchart of LAR framework

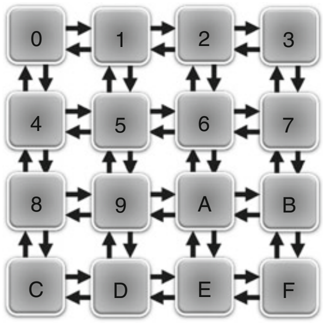

Fig. 2.2 TG of a 4×4 mesh network

After that, we find all possible shortest paths for each flow to create the routing space. Finally, we formulate an optimization problem over the routing space and solve it. In the following subsections, each step is described in detail.

2.3.1 Model Architecture and Application

In order to characterize network performance, a network model is essential. As shown in Fig. 2.2, a directed graph, which is called *Topology Graph* (*TG*), can represent the topology of an NoC architecture. Vertices and edges of TG show nodes and links of the NoC, respectively. Every node in TG contains a core and a wormhole-switched router. Such cores are local computing or storage regions, which may contain a processor, a DSP core, a configurable hardware, a high-bandwidth I/O controller, or a memory block. Each core is equipped with a Resource Network Interface (RNI). The RNI translates data between cores and routers by packing/unpacking data packets and also manages the packet injection process. Packets are injected into the network on injection channel and leave the network from ejection channel. We consider the general reference architecture for routers [7], where a routing logic determines the output channel over which the packet advances. Routing is only performed with the head flit of a packet. After routing phase, a crossbar switch handles the connections among input and output channel.

An application can be modeled by a graph called *Communication Graph* (CG). CG is a directed graph, where each vertex represents one selected task, and each

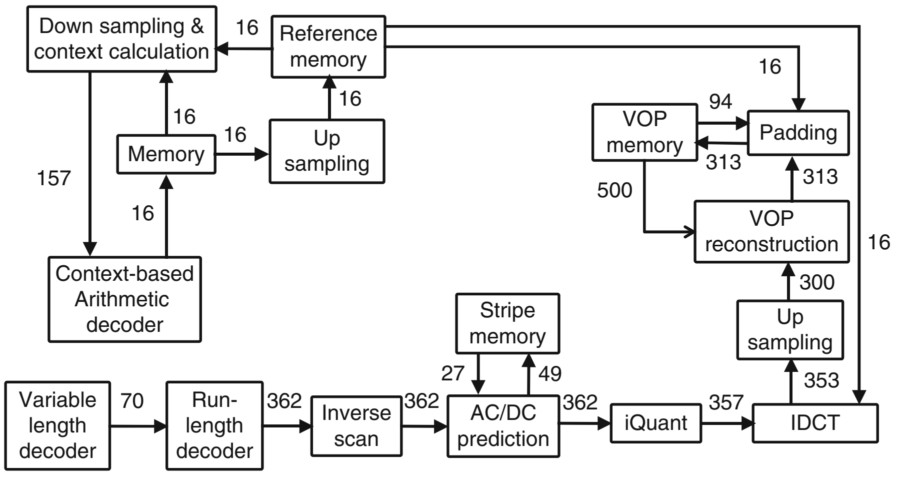

Fig. 2.3 CG of a video object plane decoder (VOPD) application [24]

directed arc represents the communication volume from source task to destination task. As an example, the CG of a video object plane decoder (VOPD) is shown in Fig. 2.3 [24]. Each node in the CG corresponds to a task and the numbers near the edges represent the bandwidth (in MBytes/s) of the data transfer, for a 30 frames/s MPEG-4 movie with 1,920 $\times$ 1,088 resolution [24].

2.3.2 *Construct Channel Dependency Graph*

Dally and Seitz simplified designing deadlock-free routing algorithms with a proof that an acyclic channel dependency graph (CDG) guarantees deadlock freedom [6]. Each vertex of the CDG is a channel in TG. For instance, vertex 01 in Fig. 2.4 corresponds to the channel from node 0 to node 1 in Fig. 2.2. There is a directed edge from one vertex in CDG to another if a packet is permitted to use the second channel in TG immediately after the first one. To find the edges of a CDG, we use the *Dijkstra's algorithm* to find all shortest paths between source and destination of any flows in corresponding TG. CDG of a 4 $\times$ 4 mesh network (Fig. 2.2) under minimal fully adaptive routing is shown in Fig. 2.4a, when any two nodes have the need to communicate such as in the uniform traffic pattern.

2.3.3 *Remove Cycles from CDG*

Traditional routing algorithms, such as *dimension-order routing* (DOR) and turn model, extract an acyclic CDG by systematically removing some edges from the CDG regardless of the traffic pattern. This may result in poor performance of

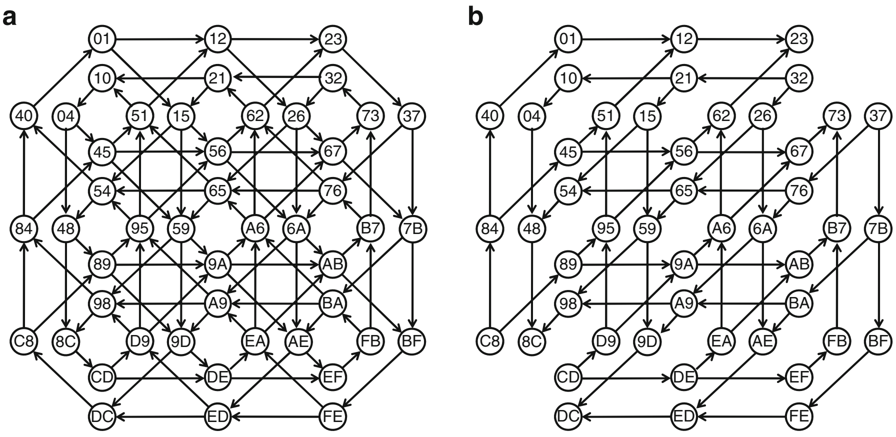

Fig. 2.4 The CDG of 4×4 mesh network for minimal fully adaptive routing under (**a**) uniform and (**b**) transpose traffic patterns

Table 2.1 Number of cycles in CDG of mesh networks

TG	Number of cycles in corresponding CDG
Mesh (2×2)	2
Mesh (2×3)	8
Mesh (3×3)	292
Mesh (3×4)	14,232
Mesh (4×4)	6,982,870
Mesh (4×5)	3,656,892,444

routing algorithm due to prohibition of unnecessary turns. For instance, as shown in Fig. 2.4b, there is no cycle in CDG of 4×4 mesh network under transpose traffic pattern, which the node in row i and column j sends packets to the node in row j and column i. However, traditional routing algorithms conservatively remove some edges from the CDG.

We modify the *depth-first-search* (*dfs*) algorithm to find cycles in a given CDG. Since we want to remove minimum number of edges, we delete an edge from the CDG which is shared among more cycles. Note that, this edge is removed if the reachability of all flows is guaranteed. For example, in a CDG of 4×4 mesh network, shown in Fig. 2.4a, there are 6,982,870 cycles and the edge from vertex 40 to vertex 01 is shared among 5,041,173 cycles. Thus by removing this edge from the CDG, the number of cycles is considerably reduced to 1,941,697. These steps are repeated again while there is no cycle in the CDG. Table 2.1 shows the numbers of cycles found by LAR in the CDG of different mesh networks. As it can be vividly seen, number of cycles is exponentially grown with the size of the TG and it takes a long time to find all cycles in the CDG. Hence, we find cycles in the CDG till certain number of cycles, and then remove an edge from the CDG which is shared among more cycles.

2.3.4 Create Routing Space (RS)

In this step, we apply Dijkstra's algorithm to the acyclic CDG to find all shortest paths between source and destination of flows in corresponding TG and create a set of f flows $RS = \{F_1, F_2, \ldots, F_f\}$ where f is the number of all flows in the system. $F_i = (\lambda_i, C_{A_i}, n_i, P_i)$, where λ_i is the average packet generation rate and C_{A_i} is the coefficient of variation (CV) of packet interarrival time for flow i. We remind that the relationship between CV of random variable X and its moments is represented by $C_X^2 = \overline{x^2}/\bar{x}^2 - 1$. In [14], we show that CV of a random variable reflects the burstiness intensity very well. n_i is the number of available shortest paths for flow i and P_i is itself a set and includes all n_i routes for flow i.

Usually more than one shortest path is available between two nodes ($n_i > 1$) in the routing space RS, so it is reasonable to choose a path such that the average packet latency is minimized. In the next subsection, we formulate an optimization problem over RS to find a suitable route for each flow and then use the simulated annealing heuristic to solve this problem.

2.3.5 Routing Space Exploration

2.3.5.1 Define Optimization Problem

In this subsection, we define an optimization problem to explore the routing space of RS. It is essential to define *decision variables* and *objective functions* in formulating an optimization problem. Our goal is to select a path for flow i ($1 \leq i \leq f$) among n_i available paths to minimize the average packet latency. Therefore, we define $X = \{x_1, x_2, \ldots, x_f\}$ as decision variables in the space of RS where x_i refers to a path number for flow i ($1 \leq x_i \leq n_i$) and the average packet latency as objective function.

The use of simulation experiments makes the task of searching for efficient designs computationally intensive and does not scale well with the size of networks since the search space of such a problem increases dramatically with the system size. Therefore, it is simply impossible to use the simulation in optimization loops. In the following subsection, we use an efficient analytical model to find nearly optimal solutions in reasonable time.

2.3.5.2 Analytical Latency Model

If the performance of a routing algorithm is measured in terms of average packet latency, then maximizing the performance means, in fact, minimizing the end-to-end packet latency. In this section, we briefly review a recently proposed analytical performance model which estimates the average packet latency in on-chip networks [14].

In a wormhole switched network, the end-to-end delay of a packet consists of two parts: the latency of the head flit and the latency of the body flits which follow the header flit in a pipelined fashion. The average latency of the head flit can be computed as the sum of delays at each hop, clearly, the link delays the head flit experienced and the residence times of the head flit in each of the routers along the path. Therefore, generally the only unknown parameter for computing the average packet latency is the mean waiting time for a packet from input channel i to output channel j in router N ($W^N_{i\to j}$). Using a G/G/1 priority queueing model, we estimated this value by [14]

$$W^N_{i\to j} = \begin{cases} \dfrac{\rho^N_j \left(C^2_A + C^2_{S^N_j}\right)}{2\left(\mu^N_j - \lambda^N_{1\to j}\right)}, & i = 1, \\[2ex] \dfrac{\lambda^N_j \left(C^2_A + C^2_{S^N_j}\right)}{2\left(\mu^N_j - \sum_{k=1}^{i-1} \lambda^N_{k\to j}\right)^2}, & 2 \le i \le p \end{cases} \tag{2.1}$$

where the variables are listed in Table 2.2 along with other parameters used in this chapter. Therefore, to compute the $W^N_{i\to j}$ we have to calculate the arrival rate from IC_i^N to OC_j^N ($\lambda^N_{i\to j}$), and also first and second moments of the service time of OC^N_j $\left(\bar{S}^N_j, \overline{\left(s^N_j\right)^2}\right)$. In the following two subsections, packet arrival rate and channel service time are computed.

Assuming the network is not overloaded, the arrival rate from IC_i^N to OC_j^N can be calculated using the following general equation

$$\lambda^N_{i\to j} = \sum_S \sum_D \lambda^S \times P^{S\to D} \times R\left(S \to D, IC^N_i \to OC^N_j\right) \tag{2.2}$$

In Eq. 2.2, the routing function $R(S \to D, IC_i^N \to OC_j^N)$ equals 1 if a packet from IP^S to IP^D passes from IC_i^N to OC_j^N; it equals 0 otherwise. Note that we assume a deterministic routing algorithm, thus the function of $R(S \to D, IC_i^N \to OC_j^N)$ can be predetermined, regardless of topology and routing algorithm. After that, the average packet rate to OC_j^N can be easily determined as

$$\lambda^N_j = \sum_i \lambda^N_{i\to j} \tag{2.3}$$

After estimating the packet arrival rates, now we focus on the estimation of the moments of channel service times. At first, we assign a positive integer index to each output channel. Let D_j^N be the set of all possible destinations for a packet which passes through OC_j^N. The index of OC_j^N is equal to the maximum of distances among N and each M where $M \in D_j^N$. Obviously, the index of a channel is between 1

Table 2.2 Parameter notation

t_r	Time spent for packet routing decision (*cycles*)
t_s	Time spent for switching (*cycles*)
t_w	Time spent for transmitting a flit between two adjacent routers (*cycles*)
m	Average size of packets (*flits*)
L_b	The latency of body flits
$L^{S\to D}$	Average packet latency from IP^S to IP^D (*cycles*)
L	Average packet latency in the network (*cycles*)
IP^N	The IP core located at address N
R^N	The router located at address N
IC_i^N	The ith input channel in router R^N
OC_j^N	The jth output channel in router R^N
IB_i^N	Capacity of the buffer in IC_i^N (*flits*)
OB_j^N	Capacity of the buffer in OC_j^N (*flits*)
$P^{S\to D}$	Probability of a packet is generated in IP^S and is delivered to IP^D $\left(\sum_S\sum_D P^{S\to D}=1\right)$
λ^N	Average packet injection rate of IP^N (*packets/cycle*)
$\lambda^N_{i\to j}$	Average packet rate from IC_i^N to OC_j^N (*packets/cycle*)
λ_j^N	Average packet rate to OC_j^N (*packets/cycle*) $\left(\lambda_j^N=\sum_i\lambda^N_{i\to j}\right)$
$P^N_{i\to j}$	Probability of a packet entered form IC_i^N to be exited from OC_j^N
μ_j^N	Average service rate of the OC_j^N (*packets/cycle*)
$C_{S_j^N}$	Coefficient of variation (CV) for service time of the OC_j^N
C_A	CV for interarrival time of packets
ρ_j^N	The fraction of time that the OC_j^N is occupied
$W^N_{i\to j}$	Average waiting time for a packet from IC_i^N to OC_j^N (*cycles*)

and diameter of the network. In addition, the index of all ejection channels is supposed to be 0. After that, all output channels are divided into some groups based on their index numbers, so that group k contains all channels with index k.

Determination of the channel service time moments starts at group 0 (ejection channels) and works in ascending order of group numbers. Therefore, the waiting time from lower numbered groups can then be thought of as adding to the service time of packets on higher numbered groups. In other words, to determine the waiting time of channels in group k, we have to calculate the waiting time of all channels in group $k-1$. This approach is independent of the network topology and works for all kinds of deterministic routing algorithm, whether minimal or non-minimal.

In the ejection channel of R^N, the head flit and body flits are accepted in t_s+t_w and L_b cycles, respectively. Therefore, we can write $\bar{S}_1^N=t_s+t_w+L_b$ and since the standard deviation of packet size is known, we can easily compute $C_{S_1^N}$. Now, by using Eq. 2.1, the waiting time of input channels for ejection channel, $W^N_{i\to j}$, can be determined for all nodes in the network, where $2\le i\le p$.

Although the moments of service time can be computed simply for all ejection channels, service time moments of the other output channels cannot be computed in a direct manner by a general formula, and we have to use a more complicated approach. Now, we can estimate the first moment or average service time of ${OC_i}^M$ as

$$\bar{S}_i^M = \sum\nolimits_{k=1}^{q} P_{j\to k}^N \left(t_s + t_w + t_r + W_{j\to k}^N + \bar{S}_k^N - \left(IB_j^N + OB_k^N \right) \times max\left(t_s, t_w\right) \right) \quad (2.4)$$

$$\overline{\left(s_i^M\right)^2} = \sum\nolimits_{k=1}^{q} P_{j\to k}^N \left(t_s + t_w + t_r + W_{j\to k}^N + \bar{S}_k^N - \left(IB_j^N + OB_k^N \right) \times max\left(t_s, t_w\right) \right)^2 \quad (2.5)$$

where $P_{i\to j}^N$ is the probability of a packet entered form ${IC_j}^N$ to be exited from ${OC_k}^N$ and equals

$$P_{j\to k}^N = \lambda_{j\to k}^N / \lambda_i^M \quad (2.6)$$

Here, we should remind that to calculate $\bar{S}_i^M$ and $\overline{\left(s_i^M\right)^2}$ all values of $\bar{S}_k^N \; (1 \le k \le q)$ must be computed before. Finally, the CV of channel service time for ${OC_i}^M$ can be given by

$$C_{S_i^M}^2 = \overline{\left(s_i^M\right)^2} / \left(\bar{S}_i^M\right)^2 - 1 \quad (2.7)$$

Now, we are able to compute the average waiting time of all output channels using Eq. 2.1. After computing $W_{i\to j}^N$ for all nodes and channels, the average packet latency between any two nodes in the network, $L^{S\to D}$, can be calculated. The average packet latency is the weighted mean of these latencies.

$$L = \sum\nolimits_S \sum\nolimits_D P^{S\to D} \times L^{S\to D} \quad (2.8)$$

where $P^{S\to D}$ is the probability of a packet is generated in IP^S and is delivered to IP^D. LAR framework uses the simulated annealing heuristic to minimize the average packet latency L as described briefly in the next subsection.

2.3.5.3 Simulated Annealing

Simulated Annealing is a stochastic computational method for solving the global optimization problem in a large search space. It is often used when the search space is discrete. For instance, simulated annealing has been applied to some computer-aided design (CAD) problems such as module placement [21] and packet routing [15]. For such problems, simulated annealing may be more efficient than exhaustive enumeration. While simulated annealing is unlikely to find the optimum solution, it can often find an acceptably good solution in a fixed amount of time. Simulated annealing was independently proposed as an optimization technique in 1983 [17] and 1985 [3]. This technique stems from thermal annealing in metallurgy which aims to increase the size of crystals and reduce their defects by heating a material and then slowly lowering the temperature to give atoms the time to attain the lowest energy state.

To simulate the physical annealing process, the simulated annealing algorithm starts with an initial solution and then at each iteration, a trial solution is randomly generated. The algorithm accepts the trial solution if it lowers the objective function (better solution), but also, with a certain probability, a trial solution that raises the objective function (worse solution). Usually the Metropolis algorithm [2] is used as the acceptance criterion in which worse solution are allowed using the criterion that

$$e^{-\Delta E/T} > R(0,1), \tag{2.9}$$

where ΔE is the difference of objective function with current and trial solutions (negative for a better solution; positive for a worse solution), T is a synthetic temperature, and $R(0,1)$ is a random number in the interval [0,1]. Typically this step is repeated until the system reaches a state that is good enough for the application, or until a given computation budget has been exhausted. By accepting worse solutions, the algorithm avoids being stuck at a local minimum in early iterations and is able to explore globally for better solutions. Detailed information about simulated annealing approach can be found in [17].

As mentioned in Sect. 2.3.5.1, objective function is the average packet latency and decision variables are represented by the routing set $X = \{x_1, x_2, \ldots, x_f\}$ where x_i is the path number for flow i $(1 \le x_i \le n_i)$. Let $X = \{x_1, x_2, \ldots, x_r, \ldots, x_f\}$ be the initial routing set. To choose a trial routing set $X_{new} = \{x_1, x_2, \ldots, {x_r}^{new}, \ldots, x_f\}$, we generate a random number r where $1 \le r \le f$ to choose a flow, and then generate another random number ${x_r}^{new}$ where $1 \le {x_r}^{new} \le n_r$ and ${x_r}^{new} \ne x_r$ to choose another path for flow r. Using analytical model describe in Sect. 2.3.5.2, we estimate the average packet latency for current and trial routing set.

2.4 Experimental Results

To evaluate the capability of the proposed framework, we developed a discrete-event simulator that mimics the behavior of routing algorithm in the networks at the flit level. Due to the popularity of the mesh network in NoC domain, our analysis focuses on this topology but LAR framework can be equally applied for other topologies without any change. Through all the experimental results, DOR routing is considered as the initial solution for the simulated annealing algorithm. We compare the performance of LAR with DOR which becomes XY routing algorithm in 2D mesh networks.

To achieve a high accuracy in the simulation results, we use the batch means method [20] for simulation output analysis. There are ten batches and each batch includes 1,000 up to 1,000,000 packets depending on the workload type, packet injection rate, and network size. Statistics gathering was inhibited for the first batch to avoid distortions due to the startup transient. The standard deviation of latency measurements is less than 1.8% of the mean value. As a result, the confidence level and confidence interval of simulation results are 0.99 and 0.02, respectively.

For the sake of comprehensive study, numerous validation experiments have been performed for several combinations of workload types and network size. In what follows, the capability of LAR will be assessed for both synthetic and realistic traffic patterns. Since their applications differ starkly in purpose, these classes of NoC have substantially different traffic patterns.

2.4.1 Synthetic Traffic

Synthetic traffic patterns used in this research include *uniform, transpose, shuffle, bit-complement,* and *bit-reversal* [5]. After developing models describing spatial traffic distributions, we should use an appropriate model to model the temporal traffic distribution. In the case of synthetic traffics, we use the Poisson process for modeling the temporal variation of traffic. It means that the time between two successive packet generations in a core is distributed exponentially. The Poisson model widely used in many performance analysis studies, and there are a large number of papers in many application domains that are based on this stochastic assumption.

The average packet latencies in the 4×4 and 8×8 mesh networks are plotted against offered load in the network in Figs. 2.5 and 2.6, respectively. We observe that under uniform and bit-complement traffic patterns LAR converges to DOR, because in such traffic patterns the average packet latency is minimum for DOR. It means that the simulated annealing algorithm is not able to find better routes and the final solution is the same as initial solution. This result is consistent with other results reported in [4, 9, 12, 19]. The main reason is that the DOR distributes packets evenly in the long term [9]. Previous works, Odd-Even [4], turn model [9], DyAD [12], and APSRA [19] indicate that in the case of uniform traffic, their proposed approaches underperform DOR. However, as can be seen in Figs. 2.5a and 2.6a, our proposed framework has the same performance as DOR for different traffic loads.

Figure 2.5b, c compare the latency of DOR and LAR in 4×4 mesh network under transpose and bit-reversal workloads, respectively. It can be vividly seen that LAR considerably outperforms DOR. Also, in the case of 8×8 mesh network, LAR has better performance than DOR as shown in Fig. 2.6b, c.

Figures 2.5d and 2.6d reveal that under shuffle traffic pattern LAR slightly outperforms DOR. Table 2.3 shows the maximum sustainable throughput of the network for each workload and for each routing algorithm in 4×4 and 8×8 mesh networks. It also shows the percentage improvement of LAR over DOR and reveals that on average LAR outperforms DOR. The maximum load that the network is capable of handling using LAR is improved by up to 205%.

Also, the performance of LAR framework is compared against DyAD routing scheme [12] which combines deterministic and adaptive routing algorithms. We

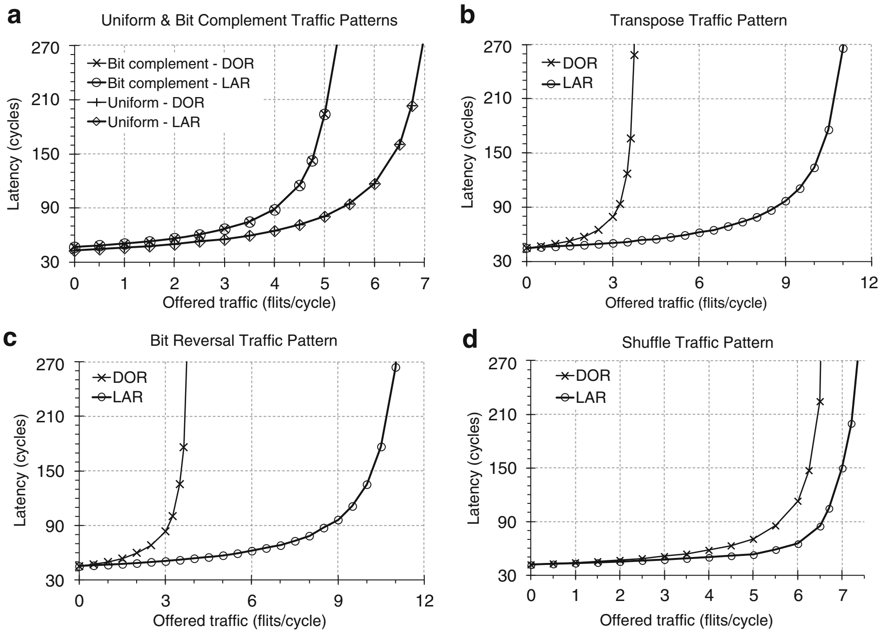

Fig. 2.5 Average packet latency under (**a**) uniform and bit-complement, (**b**) transpose, (**c**) bit-reversal, and (**d**) shuffle traffic patterns in 4×4 mesh network

Table 2.3 Improvement in maximum sustainable throughput of LAR as compared to DOR for different synthetic workloads

	4×4 mesh network			8×8 mesh network		
Workload	DOR	LAR	Impr.	DOR	LAR	Impr.
Uniform	7.4	7.4	0	15.9	15.9	0
Transpose	3.8	11.6	205%	7.7	10.5	36%
Bit-comp.	5.6	5.6	0	8.8	8.8	0
Bit-rev.	3.8	11.6	205%	7.6	9.2	21%
Shuffle	6.6	7.4	12%	12.2	13.4	10%

simulate the uniform and transpose workloads on the similar architecture (6×6 mesh network) and compare their improvement over DOR. Table 2.4 shows the percentage improvement of DyAD and LAR over DOR. In case of uniform workload, DyAD underperforms DOR while LAR has the same performance as DOR. In case of transpose traffic pattern, DyAD and LAR give about 62% and 60% improvement over DOR, respectively. This means that our deterministic routing policy can compete with adaptive routing policies (DyAD switches to adaptive mode under high traffic load) and meanwhile guarantees in-order packet delivery.

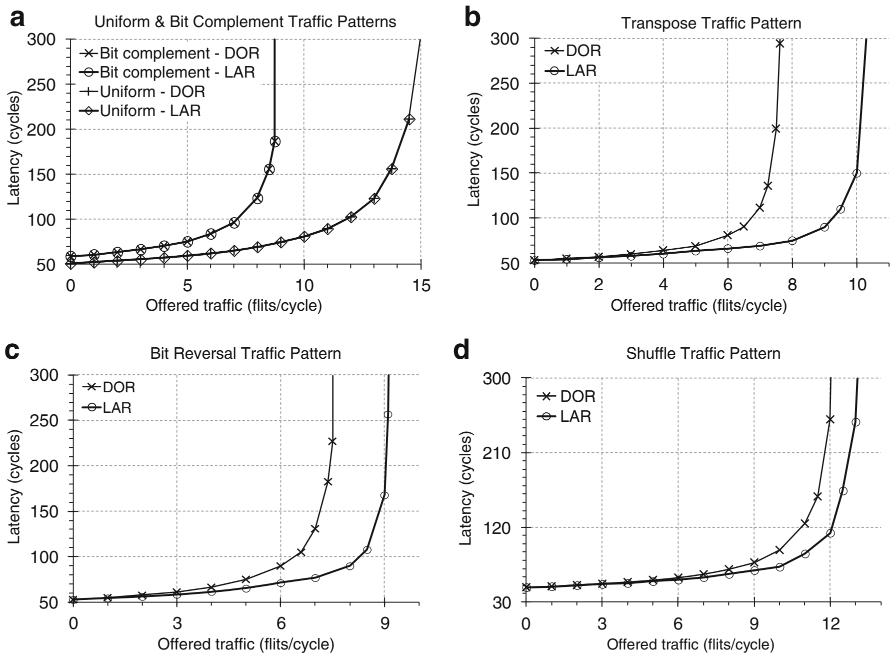

Fig. 2.6 Average packet latency under (**a**) uniform and bit-complement, (**b**) transpose, (**c**) bit-reversal, and (**d**) shuffle traffic patterns in 8×8 mesh network

Table 2.4 Improvement in maximum sustainable throughput of DyAD and LAR over DOR

	Improvement over DOR	
Workload	DyAD	LAR
Uniform	−21%	0
Transpose	62%	60%

2.4.2 Realistic Traffic

In case of realistic traffic, we consider two virtual channels for links to show the consistency of proposed framework with multiple virtual channel routing. As realistic communication scenarios, we consider a generic multimedia system (MMS) and the video object plane decoder (VOPD) application. MMS includes an H.263 video encoder, an H.263 video decoder, an mp3 audio encoder, and an mp3 audio decoder [13]. The communication volume requirements of this application are summarized in Table 2.5. VOPD is an application used for MPEG-4 video decoding and its communication graph is shown in Fig. 2.3. Several studies reported the existence of bursty packet injection in the on-chip interconnection networks for multimedia traffic [22, 25].

Table 2.5 MMS application traffic requirement [13]

src	dst	vol. (bytes)	src	dst	vol. (bytes)
ASIC1	ASIC2	25	DSP2	DSP1	20,363
ASIC1	DSP8	25	DSP3	ASIC4	38,016
ASIC2	ASIC3	764	DSP3	DSP6	7,061
ASIC2	MEM2	640	DSP3	DSP5	7,061
ASIC2	ASIC1	80	DSP4	DSP1	3,672
ASIC3	DSP8	641	DSP4	CPU	197
ASIC3	DSP4	144	DSP5	DSP6	26,924
ASIC4	DSP1	33,848	DSP6	ASIC2	28,248
ASIC4	CPU	197	DSP7	MEM2	7,065
CPU	MEM1	38,016	DSP8	DSP7	28,265
CPU	MEM3	38,016	DSP8	ASIC1	80
CPU	ASIC3	38,016	MEM1	ASIC4	116,873
DSP1	DSP2	33,848	MEM1	CPU	75,205
DSP1	CPU	20,363	MEM2	ASIC3	7,705
DSP2	ASIC2	33,848	MEM3	CPU	75,584

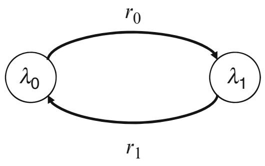
$$\begin{tikzcd} \lambda_0 \arrow[r, bend left, "r_0"] & \lambda_1 \arrow[l, bend left, "r_1"] \end{tikzcd}$$

Fig. 2.7 Two-state MMPP model

Poisson process is not the appropriate model in case of bursty traffic; consequently, we used Markov-modulated Poisson process (MMPP) model as stochastic traffic generators to model the bursty nature of the application traffic [5, 8]. MMPP has been widely employed to model the traffic burstiness in the temporal domain [8]. Figure 2.7 shows a two-state MMPP in which the arrival traffic follows a Poisson process with rate λ_0 and λ_1. The transition rate from state 0 to 1 is r_0, while the rate from state 1 to state 0 is r_1.

Since in such systems, there are various types of cores with different bandwidth requirements, placement of tasks on a chip has strong effect on the system performance. To find a suitable mapping of these applications, we formulate another optimization problem to prune the large design space in a short time and then again use the simulated annealing heuristic to find a suitable mapping vector. Initially, we map task i to node i and then try to minimize the average packet latency through the simulated annealing approach. Figure 2.8a shows that in the case of MMS application and DOR, for the initial mapping M1, average packet latency equals 87 and after a certain number of tries, the mapping vector converges to the mapping M4 with average packet latency $=41$. Furthermore, average packet latency values for mappings M2 and M3, which are two local minimum points in simulated annealing process, are shown in the figure.

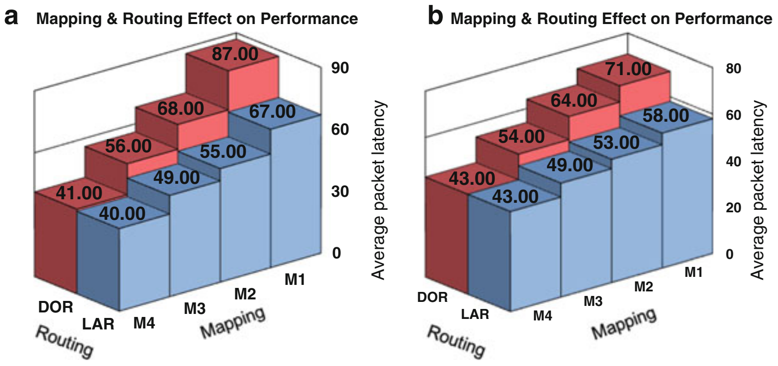

Fig. 2.8 The effect of mapping and routing on the performance of (**a**) MMS application and (**b**) VOPD application

After the mapping phase, we apply the LAR framework to these four mapping vectors. Figure 2.8a reveals that in case of mapping M1, LAR can significantly reduce the average packet latency from 87 to 67. However, for more efficient mapping vectors (M2, M3, and M4), we achieve less improvement. Specially, in the case of best mapping (M4), average packet latency is reduced insignificantly from 41 to 40. It is reasonable that DOR is latency-aware for the best mapping, because during the mapping problem solving process, we fix the routing policy to DOR and strive to minimize average packet latency for this routing policy. Likewise, as shown in Fig. 2.8b, for the VOPD application, the analysis result is the same as MMS application.

Figure 2.8 reveals that in case of application-specific traffic patterns, the improvement in the performance of the routing schemes highly depends on how the application tasks are mapped to the topology. This fact was not considered in the related works such as [16]. Nowadays, in embedded systems-on-chip there are several different types of cores including DSPs, embedded DRAMs, ASICs, and generic processors which their places are fixed on the chip. On the other hand, such a system hosts several applications with completely different workload. Furthermore, modern embedded devices allow users to install applications at run-time, so a complete analysis of such systems is not feasible during design phase. As a result, it is not feasible to map all applications such that the load is balanced for all of them with specific routing algorithm and we should balance the load in routing phase.

In this section we used the LAR framework to find low latency routes in the mesh network. Due to simplicity, regularity, and low cost merits of 2D mesh topology, it is the most popular one in the field of NoC. However, for large and 3D NoCs, which will be popular in the future, the communication in mesh architecture takes a long time. In the next subsection we use LAR to find deadlock-free paths in an arbitrary topology.

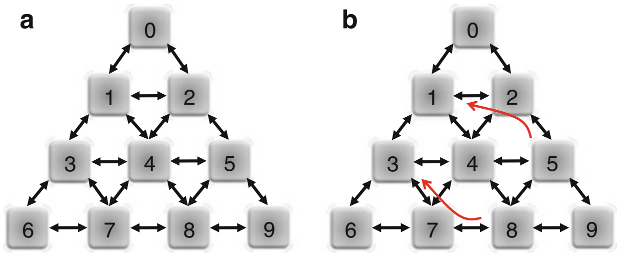

Fig. 2.9 (**a**) A custom topology and (**b**) prohibited turns

Table 2.6 Routing table for node 0 of topology in Fig. 2.8a

dst.	route	dst.	Route
0	No packet	5	SE, SE, EJ
1	SW, EJ	6	SW, SW, SW, EJ
2	SE, EJ	7	SE, SW, SW, EJ
3	SW, SW, EJ	8	SW, SE, SE, EJ
4	SW, SE, EJ	9	SE, SE, SE, EJ

2.4.3 Find Routes in an Arbitrary Topology

To show the capability of LAR framework to find deadlock-free routes in an arbitrary topology, we consider the topology shown in Fig. 2.9a. LAR reports that under uniform traffic pattern there are two cycles in the corresponding CDG and by prohibiting turns 52–21 and 87–73 (shown in Fig. 2.9b) the deadlock-freedom is guaranteed.

Table 2.6 shows the routing table for node 0 of the topology in Fig. 2.9a. Each route in the table specifies a path from node 0 to a given destination as channels name. SE, SW, and EJ specify South East, South West, and ejection channels, respectively. To route a packet, the routing table is indexed by destination address to look up the pre-computed route by LAR. This route is then added to the packet. Since there are seven channels in this network (E, S, NE, NW, SE, SW, and EJ), they can be encoded as 3-bit binary numbers. Also, there are techniques to reduce the size of routing tables [5, 19].

2.5 Conclusion

On-chip packet routing is extremely crucial because it heavily affects performance and power. This calls for a great need of routing optimization. However, due to the diverse connectivity enabled by a network and the interferences in sharing network buffers and links, determining good routing paths, which are minimal and deadlock free for traffic flows, is nontrivial. In this chapter, we have addressed the latency-aware routing problem. Using an analytical model, we first estimate the average packet latency in the network, and then embed this analysis technique into the loop of optimizing routing paths so as to quickly find deterministic routing paths for all traffic flows while minimizing the latency.

The proposed framework is appropriate for reconfigurable embedded systems-on-chip which run several applications with regular and repetitive computations on large set of data, e.g., multimedia and computer vision applications. LAR can not only design minimal and deterministic routing, but also can implement non-minimal routing without virtual channels in arbitrary topology.

References

1. K. Bondalapati, V.K. Prasanna, Reconfigurable computing systems. Proc. IEEE **90**(7), 1201–1217 (2002)
2. O. Catoni, Metropolis, simulated annealing, and iterated energy transformation algorithms, theory and experiments. J. Complex. **12**(4), 595–623 (1996)
3. V. Cerny, Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. J. Opt. Theory. Appl. **45**, 41–45 (1985)
4. G.-M. Chiu, The odd-even turn model for adaptive routing. IEEE Trans. Parall. Distr. Syst. **11**(7), 729–738 (2000)
5. W.J. Dally, B. Towles, *Principles and practices of interconnection networks*, 1st edn. (Morgan Kaufmann, San Francisco, 2004)
6. W.J. Dally, C.L. Seitz, Deadlock-free message routing in multiprocessor interconnection networks. IEEE Trans. Comput. **36**(5), 547–553 (1987)
7. J. Duato, C. Yalamanchili, L. Ni, *Interconnection Networks: An Engineering Approach* (IEEE Computer Society Press, Los Alamitos, Morgan Kaufmann, San Francisco, 2003)
8. W. Fischer, K. Meier-Hellstern, The Markov-Modulated Poisson Process (MMPP) cookbook. Perform. Evaluat. **18**(2), 149–171 (1993)
9. C.J. Glass, L.M. Ni, The turn model for adaptive routing. J. Assoc. Comput. Mach. **41**(5), 874–902 (1994)
10. P. Guerrier, A. Greiner, A generic architecture for on-chip packet-switched interconnections, in *Proceedings of the Design, Automation, and Test in Europe* (Paris, France, 2000), pp. 250–256
11. A. Hemani et al., Network on a chip: An architecture for billion transistor era, in *Proceedings of the IEEE NorChip* (Turku, Finland, 2000), pp. 166–173
12. J. Hu, R. Marculescu, DyAD – smart routing for networks-on-chip, in *Proceedings of the Design Automation Conference* (San Diego, 2004), pp. 260–263
13. J. Hu, R. Marculescu, Energy- and performance-aware mapping for regular NoC architectures. IEEE Trans. Comp. Aid. Des. Integr. Circuit. Syst. **24**(4), 551–562 (2005)
14. A.E. Kiasari, Z. Lu, A. Jantsch, An analytical latency model for networks-on-chip. IEEE Trans. VLSI Syst. **21**(1), 113–123 (2013)
15. A.E. Kiasari, A. Jantsch, Z. Lu, A framework for designing congestion-aware deterministic routing, in *Proceedings of the 3rd International Workshop on Network-on-Chip Architectures (NoCArc), Held in conjunction with the 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-43)* (Atlanta, 2010), pp. 45–50
16. M.A. Kinsy et al., Application-aware deadlock-free oblivious routing, in *Proceedings of the ISCA* (Austin, 2009), pp. 208–219
17. S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
18. S. Murali et al., Analysis of error recovery schemes for networks on chips. IEEE Des. Test Comp. **22**(5), 434–442 (2005)

19. M. Palesi et al., Application specific routing algorithms for networks on chip. IEEE Trans. Parall. Distr. Syst. **20**(3), 316–330 (2009)
20. K. Pawlikowski, Steady-state simulation of queueing processes: A survey of problems and solutions. ACM Comput. Surv. **22**(2), 123–170 (1990)
21. C. Sechen, A. Sangiovanni-Vincentelli, The TimberWolf placement and routing package. J. Sol. State Circuit. **SC-20**, 510–522 (1985)
22. V. Soteriou, H. Wang, L.-S. Peh, A statistical traffic model for on-chip interconnection networks, in *Proceedings of the MASCOTS* (Monterey, 2006), pp. 104–116
23. W. Trumler et al., Self-optimized routing in a network-on-a-chip. IFIP World Comp. Cong. **268**, 199–212 (2008)
24. E.B. van der Tol, E.G. Jaspers, Mapping of MPEG-4 decoding on a flexible architecture platform. SPIE **4674**, 1–13 (2002)
25. G. Varatkar, R. Marculescu, Traffic analysis for on-chip networks design of multimedia applications, in *Proceedings of the Design Automation Conference* (New Orleans, 2002), pp. 795–800