Empowering Study of Delay Bound Tightness with Simulated Annealing

Xueqian Zhao and Zhonghai Lu
Department of Electronic Systems, School for ICT
KTH Royal Institute of Technology, Stockholm, Sweden
{xueq, zhonghai}@kth.se

***Abstract*—Studying the delay bound tightness typically takes a practical approach by comparing simulated results against analytic results. However, this is often a manual process whereas many simulation parameters have to be configured before the simulations run. This is a tedious and time-consuming process. We propose a technique to automate this process by using a simulated annealing approach. We formulate the problem as an online optimization problem, and embed a simulated annealing algorithm in the simulation environment to guide the search of configuration parameters which give good tightness results. This is a fully automated procedure and thus provide a promising path to automatic design space exploration in similar contexts. Experiment results of an all-to-one communication network with large searching space and complicated constraints illustrate the effectiveness of our method.**

I. Introduction

Quality-of-Service (QoS) is an important aspect in Network-on-Chip (NoC) design [1] in order to provide guaranteed packet delivery service for applications that have strict timing requirements on communication delay, e.g. multimedia streams. In recent years, *Network Calculus* [2][3] based formal analysis has been successfully applied to calculate the delay bounds for on-chip networks [4][5]. However, systematic study of delay bound tightness is seldom touched.

Tightness is a metric to measure how tight the calculated result can bound the worst case communication delay in practical working scenarios. Studying tightness has practical significance for designing NoC as well as validating the formal analysis. From the analysis perspective, it is studied as a reachability problem, because the formal model covers all the cases whereas the practical delay depends on implementation and parameter configuration. Simulation captures more practical implementation details than the abstract formal analysis, and is a pragmatic approach to validate the delay bound tightness as used in [4] and [5]. However, simulation is hard to cover all the system states [6]. Moreover, the worst case delay bound may be only reached in some extreme case that might be very difficult to occur in simulations. This brings about the challenge for the effectiveness of simulation based validation. Furthermore, we are often more interested in finding the system configurations that lead the communication delay close to the delay bound from the design perspective.

Unfortunately, the search space of the system configuration may easily become extremely large due to the combinatorial explosion of the system parameters. Manual configuration has poor scalability and is not competent in finding the configuration for best tightness. For example, let us consider a simple system with 20 parameters whose scale is very easy to be reached in a real NoC design, and each parameter has just only 5 possible values. The whole search space then has over 9.5×10^{13} cases. Assume that each simulation takes 5 seconds, the time to cover all the cases will exceed 1.5×10^7 years. Apparently this is unrealistic for brute-force enumeration, let alone manual configuring. Even though we can manually configure the system based on some analysis of the worst case conditions, the complex contention patterns among flows still make it require tremendous effort and time. Thus an effective computer aided method is essential for searching the worst case delay, especially when the system scales.

Simulated Annealing (SA) [7][8] inspired by annealing in metallurgy is a meta-heuristic method for global optimization. It is widely used in a large range of applications to search for a good approximation to the global optimum of a given objective function (or cost function). SA is capable of handling complex problems with multi-dimensional searching space, especially those with discontinuous or discrete variables. Moreover, the cost function is not necessarily an explicit function with respect to the specified parameters. These characteristics of SA enable the computer aided configuration and enable to embed the simulation into this heuristic method for automatic searching.

In this work, we present an SA aided tightness study to search for the configuration that leads to best tightness. The problem is formulated as an optimization problem with a set of constraints on the system's parameters. Then we propose an SA algorithm to search for the global optimum or suboptimum solution for this well defined problem. The tightness is formulated as an implicit function in the searching space, and the mapping between the tightness and the parameters is set up by simulations. Our experimental results show that this method can effectively find the configuration for best tightness and is capable of dealing with large scale systems.

The paper's contributions can be summarized as follows.

(1) We formulate the delay bound tightness problem as an optimization problem with the objective of maximizing the tightness.

(2) We propose an SA algorithm to solve the formalized problem for global optimum or suboptimum solution, and thus to reach the best tightness.

(3) By studying the case of an all-to-one communication pattern in a binary tree topology, we show the effectiveness of the proposed method to study delay bound tightness, especially for handling a complex system with discrete variables and multi-dimensional search space.

The rest of the paper is organized as follows. We outline related work in Section II. In Section III, we introduce the symbols and the delay bound calculation techniques based on network calculus. The problem formulation of searching for best tightness is presented in Section IV after the discussion of the challenge in the tightness study. We then propose an SA algorithm in Section V. Experimental results are reported in Section VI. Finally, we conclude in Section VII.

II. Related Work

The network calculus theory was originated in 1991 by Cruz [9]. As it develops, network calculus has been applied to on-chip micro-networks. In [5], Qian et al. identifies three basic flow contention

978-3-9815370-2-4/DATE14/©2014 EDAA

patterns and calculate the per-flow *equivalent service curve* (ESC) to calculate the delay bound. Furthermore, in [4] more sophisticated resource sharing such as link sharing, buffer sharing and control sharing are studied. By resolving these resource sharings, the worst-case delay bound analysis technique is developed.

In [10], the validation of the delay bound is also studied as a reachability problem as we do in the paper. They proved that the delay bound is achievable by setting up specific scenarios and assumptions. However, it becomes increasingly difficult and complicated with the increasing complexity of flow contention and resource sharing in the network. In this work, we develop an SA aided method rather than manual analysis to investigate the achievable best tightness for a complex system.

SA is widely used to search for global optimum solutions for those NP-Hard (Nondeterministic Polynomial-time Hard) problems. It is also applied in optimizing the NoC design in the design space to make trade-offs among various specifications. Lu et al. [11] develops a clustering technique with SA to map cores to a 2D mesh NoC. In [12], SA is used to optimize the power consumption and to design energy efficient network. Besides, SA is also applied to optimize the test schedule for NoC [13] within the thermal and power constraints. All these applications practically require an explicit analytical function as the objective.

In our work, we express tightness as an *implicit* function of the system parameters, and embed the simulation process into the algorithm to set up the mapping between tightness and configuration space. To our knowledge, we are the first to apply a heuristic algorithm, in this case, SA, to the study of delay bound tightness.

III. Delay Bound Analysis Technique Using Network Calculus

A. Notations and Definitions

We first define the symbols and terms for network calculus and the problem formulation, as listed in Table I.

TABLE I
SYMBOLS AND DEFINITIONS

Symbol	Explanation
f_i	The ith flow
α_i	Arrival curve of f_i with burstiness b_i and rate r_i
β_i	The ith service curve
β_{C_i,T_i}	Latency-rate service curve with rate C_i and latency T_i
ϕ_i	The weight of flow f_i with weighted round robin arbitration
β_i^{sys}	The end-to-end *equivalent service curve* (*ESC*) provided by the system to f_i
C_i^{sys}	The system equivalent service rate for flow f_i
p_j	The jth parameter used to configure the system
l_j	Lower bound of parameter p_j
h_j	Upper bound of parameter p_j
$\mathcal{P}$	The set of parameters, $\mathcal{P} = \{b_i, r_i, C_i, T_i, \phi_i \mid i \in \mathcal{I}\}$, where $\mathcal{I}$ is the index set
$\mathcal{S}$	The configuration space. Each element $s \in \mathcal{S}$ is a set of parameters such that $s = \{p_j \mid l_j \leq p_j \leq h_j,\ j \in \mathcal{I}\}$ represents one configuration of the system.
d_{max}	The maximum packet delay obtained from the simulation under a specific configuration s
D_{max}	The maximum packet delay defined on $\mathcal{S}$
$\bar{D}$	Flow's delay bound calculated by closed-form formula
$\wedge$	The *minimum* operation, e.g. $a \wedge b = \min\{a, b\}$
$[x]^+$	The maximum of x and 0, namely $[x]^+ = \max\{x, 0\}$

B. The Delay Bound Analysis Technique

1) Concepts: In network calculus, *arrival curve* and *service curve* are two fundamental concepts. Arrival curve defines the upper bound of the injected traffic within any time interval. One of the widely used curves is the linear arrival curve $\gamma_{b,r} = rt + b$ when $t > 0$, otherwise 0, in which r is the sustainable arrival rate and b the burstiness. Service curve models a network element, e.g. a router, representing the lower bound of its service capability. One commonly used service curve is the latency-rate function $\beta_{C,T} = C[t - T]^+$, where C is the minimum service rate and T the maximum processing latency. The delay bound of a flow f_i is calculated by finding the greatest horizontal distance between its arrival curve α_i and the system ESC β_i^{sys}.

2) The analysis procedure: Our goal is to derive a closed-form formula to compute the upper delay bound for a flow. We call this flow *tag flow* to contrast other contention flows. The key idea is to compute its end-to-end ESC considering all resource sharing. According to [4][5], the delay bound analysis procedure can be summarized as follows:

Step 1 Resolve link sharing and calculate the ESC at each node;
Step 2 Resolve buffer sharing and calculate the left-over service curve of the tag flow;
Step 3 Compute the system ESC for the tag flow;
Step 4 Derive the delay bound of the tag flow.

Next, we give an example for the delay bound analysis.

3) An Illustrative Example: Figure 1 shows a simple example in which three flows f_1, f_2, f_2 aggregate through two routers R_1 and R_2. At router R_1, the three flows share the output channel and go to buffer B_2 in the first-in-first-out (FIFO) order. The injection of each flow is bounded by a linear arrival curve $\alpha_i = r_i t + b_i$ for $i = 1, 2, 3$. The weight of each flow is ϕ_i when arbitrating for the shared output channel of R_1. The service curves of R_1 and R_2 are $\beta_i = \beta_{C_i,T_i} = C_i[t - T_i]^+$ ($i = 1, 2$), respectively.

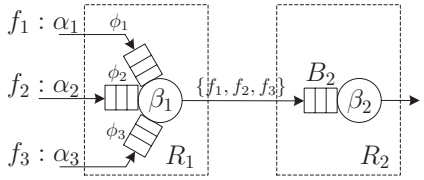

Fig. 1. Three flows aggregate with link sharing and buffer sharing.

Suppose that f_1 is the tag flow. We are to calculate its delay bound from the input of R_1 to the output of R_2. Following the analysis procedure without giving details, the closed-form formula of delay bound can be deduced after resolving link sharing and buffer sharing. Under the stability condition, we have

$$\bar{D} = T_1 + \phi_2 + \phi_3 + T_2 + \frac{b_{2,3}^*}{C_2} + \frac{b_1}{\frac{\phi_1 C_1}{\phi_1+\phi_2+\phi_3)} \wedge (C_2 - r_2 - r_3)}, \quad (1)$$

where $b_{2,3}^* = b_2 + r_2(T_1 + \phi_1 + \phi_3) + b_3 + r_3(T_1 + \phi_1 + \phi_2)$.

IV. Problem Formulation for Tightness Study

We first expose the challenge in the tightness study and then give the tightness problem formulation.

A. Challenge in Tightness Study

1) The Study of Delay Bound Tightness: The delay bound calculated by the network calculus analysis model covers all the system state. However, in realistic on-chip network designs, we are more interested in how close the real simulated maximum delay can approach to the bound under practical configurations. Thus we define the tightness as the ratio of the real simulated maximum delay to the calculated delay bound as

$$\xi = \frac{D_{max}}{\bar{D}} \times 100\%. \quad (2)$$

This definition works well and is used in several works (e.g. [4][5]) to validate the developed network calculus model. One may think that the tightness metric seems not to be so good when both D_{max} and $\bar{D}$ are very small even though D_{max} is very close to $\bar{D}$. For example, we get both 80% of tightness with $4/5$ and $40/50$ whereas the latter has a much larger difference. We do need to examine the difference between D_{max} and $\bar{D}$ when we evaluate the model especially when the delay is small. However, it does not affect the use of this definition to search for best tightness with the method proposed in this work.

2) The problem exemplification: Consider to find a parameter configuration of the small example in Figure 1 to reach the worst case delay. There are 13 parameters to be configured including b_i, r_i and ϕ_i for flow f_i, $i = 1, 2, 3$ and T_j, C_j for service curve β_j, $j = 1, 2$. Assume that each parameter has just only 5 candidate values, there would be 5^{13} ($> 10^9$) combinations in total. It is apparently unrealistic to use brute-force search. One may consider to analyse the conditions of causing the worst case of link sharing and buffer sharing, and set up the relations between the parameters. However, for the whole system, the packet with worst case delay may not experience worst delays in both link sharing and buffer sharing. Moreover, manual analysis and parameter setup still demand significant effort and time. Thus it is non-trivial at all to manually find the configuration which results in the worst case delay.

In the following, we formalize this configuration searching as an optimization problem.

B. Formalization as An Optimization Problem

Let ζ be the closed-form formula to calculate the delay bound for the tag flow in the system. We have $\bar{D} = \zeta(\mathcal{S})$ where $\mathcal{S}$ is the configuration space. However, for the simulated maximum delay D_{max}, there is no explicit function to set up the relation between D_{max} and the configuration space $\mathcal{S}$. We use function $g(D_{max}, \mathcal{S}) = 0$ to implicitly define D_{max} as a function of $\mathcal{S}$. We have the optimization problem defined below to find a parameter configuration s such that we maximize the tightness $\xi(\mathcal{S})$ as defined in Equation 2 under this configuration. Let the cost function be

$$J(\mathcal{S}) = 100\% - \xi(\mathcal{S}). \tag{3}$$

Input: The configuration space $\mathcal{S}$ which is defined by the parameter constraints.

Output: (1) A specific configuration $s \in \mathcal{S}$ which leads to best tightness; (2) The tightness value $\xi(s)$ with respect to s.

$$Objective: \min J(\mathcal{S}) \tag{4}$$

s.t.

$$\begin{cases} l_j \le p_j \le h_j, & \forall p_j \in \mathcal{P} \\ r_i \le C_i^{sys}, & \forall i \in \mathcal{I} \\ b_i \in \mathcal{Z}^+, & \forall i \in \mathcal{I} \\ T_i \in \mathcal{Z}^+, & \forall i \in \mathcal{I} \\ \phi_i \in \mathcal{Z}^+, & \forall i \in \mathcal{I} \end{cases}$$

In the formulation, function J is the *cost function* with respect to the configuration space $\mathcal{S}$. $\mathcal{Z}^+$ is the set of positive integer; $\mathcal{P}$ is a set of parameters, and $\mathcal{I}$ is an index set. For each parameter $p_j \in \mathcal{P}$, we constrain its value between its lower bound l_j and upper bound h_j. The inequality $r_i \le C_i^{sys}$ ensures the stability condition for the system, namely the equivalent system service rate for flow f_i should not be less than the flow's sustainable arrival rate r_i. Moreover, the flow burstiness b_i should be a positive integer, because we count the traffic with discrete data unit such as *flit* or *packet* for on-chip networks. So are the constraints for router service latency T_i and flow service weights ϕ_i which take cycles as the unit in digital systems. Using processing time, i.e. cycles, as the flow weight is widely used in arbitrations such as weighted-round-robin (WRR). As the arbitration policy may vary with different designs, we may change the constraint on ϕ_i accordingly.

The problem is thus abstracted as a programming problem with the objective of maximizing the tightness. However, this is an integer programming as b_i, T_i and ϕ_i are all integers. Moreover, the cost function $J(\mathcal{S})$ is an implicit function with respect to the configuration space $\mathcal{S}$. Thus we propose to solve this programming problem with simulated annealing algorithm, which does not require the explicit information of the cost function. Note that the implicit function g is actually determined in cycle accurate simulations. We can always find a maximum packet delay d_{max} from each simulation under a specific configuration s, thus to set up the mapping between the configuration space $\mathcal{S}$ and the maximum delay D_{max}.

V. SA Aided Configuration Search

A. Overview

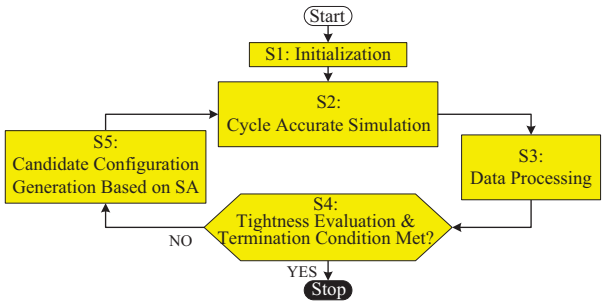

Fig. 2. Overview of the SA aided configuration search.

The SA aided configuration search comprises five main steps to maximize the tightness as shown in Figure 2.

1) We start from assigning an initial set of parameters to the system. This parameter configuration is selected randomly or according to some empirical results.
2) Then, the cycle accurate simulation is conducted to collect the packet timing information such as the packet injection time and ejection time.
3) We process the obtained data and get the maximum packet delay for this simulation. Each time this step is executed, we find a point in the solution space of the equation $g(D_{max}, \mathcal{S}) = 0$, which implicitly defines the relation between D_{max} and $\mathcal{S}$.
4) We evaluate the tightness ξ for current iteration, and judge whether the termination condition is met. If it is met, stop and output the found tightness and the corresponding configuration. Otherwise, continue to the next step.
5) The next set of parameters (candidate configuration) is generated according to the SA algorithm. We return to the second step and continue the process above to evaluate the tightness under the new configuration.

B. Simulated Annealing Algorithm

SA is a meta-heuristic method to help find global optimum or suboptimum solutions for optimization problems. It mainly consists of three aspects, namely the *state generation method*, the *acceptance criterion* and the *annealing scheduling*. With decades of development, SA has been developed into a family of probabilistic algorithms with variants on the three aspects. The discussion on SA improvements is not our concern of this work, and we will focus on applying SA to solve the optimization problem defined in Section IV-B.

TABLE II
SYMBOLS AND DEFINITIONS FOR THE SA ALGORITHM

Symbol	Explanation
$Y(\cdot,\cdot)$	Function to generate the next state (configuration).
$J(\cdot)$	Cost function of the SA algorithm, defined as an implicit function on $\mathcal{S}$.
T	Temperature of the annealing process.
dE	Energy difference when transiting to the next state.
$random(0,1)$	Function to generate a random number between 0 and 1 with uniform distribution.
r	Cooling rate of the annealing process, $0 < r < 1$.
T_{min}	Threshold temperature to stop the annealing process.
s	Current parameter configuration, $s \in \mathcal{S}$.
J_c	Cost value of the current configuration (or state).

Algorithm 1 reports the SA proposed in this work. We summarize the symbols and notations used in the algorithm in Table II. The algorithm requires an initial state s_0 to set the start point of the algorithm, an initial temperature T_0 and a termination temperature T_{min} to define the termination condition, and a cooling rate r for the annealing scheduling. The algorithm finally outputs the configuration s for best tightness and the corresponding cost function value $J(s)$.

The outer *while-loop* determines the annealing schedule, which cools down the temperature with rate r ($0 < r < 1$) in each iteration. The inner *for-loop* goes through the candidate states under the same temperature. Assume that there are n parameters, forming an n-dimensional space. Hence the configuration $s = \{p_1, p_2, \ldots, p_n\}$. Each time the state generation function $Y(s, p_j)$ generates a state neighbour to the current state s by moving on the dimension of p_j with a random step. The generated new state s_1 is then used to solve the implicit function $g(d_{max}, s_1) = 0$ by simulating with configuration s_1. Then s_1 is evaluated to judge whether to accept it or not. We adopt *Metropolis Criterion* to determine the probability of accepting the new state. Even though the new state results in a worse cost value, we still accept it with probability determined by $\exp(dE/T) >$ random$(0,1)$. This trick helps the algorithm to jump out of the local optimum and search more states to find the global optimum solution. The inner *for-loop* actually ensures the algorithm to move along the dimension with lowest cost value under the *Metropolis Criterion*. The finally selected state s_{tmp} in the *for-loop* becomes the new state in the next annealing iteration.

The time complexity of SA depends on how the annealing schedule is designed. SA can have worse time complexity than exhaustive enumeration if we force it to find the global optimum with probability 1. Consider Algorithm 1, the outer loop runs for $N = \lfloor \log_r(T_{min}/T_0) \rfloor + 1$ times. The inner *for-loop* executes n times. Let $\mathcal{O}_g$ be the time complexity of solving $g(D_{max}, s_1) = 0$. The time complexity of Algorithm 1 is $\mathcal{O}(nN) \cdot \mathcal{O}_g$.

C. Determine the SA Inputs

In the SA algorithm, higher temperature can help jump out of local optimum points. When the temperature cooled down, lower temperature leads to better search precision around the possible global optimum solution. Thus we make the searching process jump among different local optimum points at the early stage (when the temperature is high). When the temperature is low, we make the searching more likely to try the points around the possible global optimum point.

Since acceptance probability is determined by the *Metropolis Criterion* "$\exp(dE/T) > random(0,1)$", we determine T_0 and T_{min} by solving equations

$$\exp(\frac{\overline{dE}}{T_0}) = P_h, \qquad \exp(\frac{\overline{dE}}{T_{min}}) = P_l, \tag{5}$$

Algorithm 1: Simulated annealing to find the best tightness and the corresponding parameter configuration.

Input: Initial annealing temperature T_0, initial parameter configuration s_0, termination temperature T_{min} and the cooling rate r, $0 < r < 1$.
Output: Optimized parameter configuration s and the minimized cost value $J(s)$.
Initialize: $T = T_0$, $s = s_0$, $J_c = J(s_0)$

```
while T ≥ T_min do
    foreach p_j ∈ s do
        s_1 = Y(s, p_j) ; // Generate a candidate state
        // Obtain the maximum packet delay
        Do simulation to solve g(d_max, s_1) = 0 ;
        Calculate J(s_1) ; // Candidate cost value
        dE = J_c − J(s_1) ; // Energy difference
        // If the cost value is decreased, accept
           the state
        if dE > 0 then
            s_tmp = s_1 ; // Transit the state
            J_c = J(s_1) ; // Update the cost value
        else if exp(dE/T) > random(0,1) then // Judge with
        the Metropolis Criterion
            s_tmp = s_1 ;
        end
    end
    s = s_tmp ; // Transit the state
    T = r * T ; // Cool down the temperature
end
return s, J_c ;
```

where $\overline{dE}$ is an estimated mean value of dE, obtained by calculating the average value from the statistics of $\overline{dE}$ obtained from a simulation; P_h is a positive real number close to 1, e.g. 0.95; P_l is a positive real number close to 0, e.g. 0.05. The value of T_0 and T_{min} can also use the empirical value directly, e.g. $T_0 = 100$ and $T_{min} = 1$.

The cooling rate r can be set as an empirical value, e.g. 0.95, or be determined by the number of iterations we expect to run. Suppose that we expect to run the outer loop for N times, then r is calculated by $r = (T_{min}/T_0)^{1/N}$. As for the initial state s_0, it can be selected randomly or set using data from previous simulations. The initial value of the cost function J then is calculated by these given initial configurations.

VI. EXPERIMENTS AND RESULTS

A. Experimental Purpose and Setting

We design an experiment to show the whole process of applying the proposed SA aided method to search for best tightness. We first introduce the experiment setup and give the closed-form delay bound formula of the tag flow. Then we set the SA parameters and finally show the results obtained by the SA aided method.

We consider an all-to-one communication case which is a typical communication pattern for parallel applications, as illustrated in Figure 3. The routers are organized in the topology of a 4-layer binary tree. The tree includes three types of nodes, namely the root node R_1, the trunk nodes $R_2 \sim R_7$ and the leaf nodes $R_8 \sim R_{15}$. Except for the root R_1, each node R_i has a local traffic injection $f_i, i = 2 \ldots 15$ with arrival curve $\alpha_i = r_i t + b_i$, and transfers packets to the father node $R_{\lfloor i/2 \rfloor}$ with a latency-rate service curve $\beta_i = C_i[t - T_i]^+$. The root R_1 is the common destination of all the injected flows and sinks packets with a service curve of $\beta_1 = C_1[t - T_1]^+$. A trunk node R_i transfers packets from two son-nodes and meanwhile has a local flow injection. It outputs an aggregate flow to its father node. As for the leaf node, each one only has local flow injection. To simplify the

analysis and formulas, we set that each input port of a router has a sufficiently large FIFO buffer to avoid dropping packets.

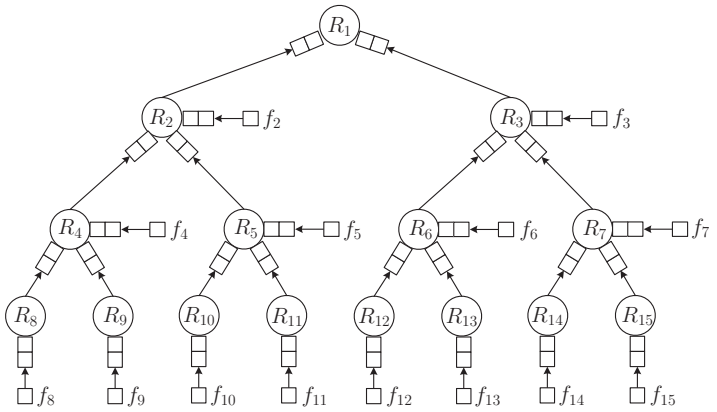

Fig. 3. The topology of the binary tree of depth 4.

Denote Ω_i the sub-tree with node R_i as the root, f_{Ω_i} the output flow at the root of Ω_i. Thus f_{Ω_i} is an aggregate flow for $\Omega_2 \sim \Omega_7$ and a single flow for $\Omega_8 \sim \Omega_{15}$. For example, we have $f_{\Omega_8} = f_8$ at the output of R_8, and $f_{\Omega_4} = \{f_8, f_9, f_4\}$, $f_{\Omega_2} = \{f_{\Omega_4}, f_{\Omega_5}, f_2\}$ as aggregate output flows at R_4 and R_2, respectively. The flows aggregate upstream into the input buffer of the father nodes layer by layer and share the output channels of trunk nodes and the sink of the root node. We use weighted round-robin (WRR) to arbitrate the sharing of the output channels and sink.

B. Closed-form Delay Bound Formula

We deduce the closed form delay bound for the flows injected at the bottom layer. Due to the topological symmetry, we select f_8 as the tag flow. The delay bounds of the other bottom layer flows can be calculated similarly.

Let $\mathcal{K} = \{1, 2, \ldots, 15\} \cup \{\Omega_1, \Omega_2, \ldots, \Omega_{15}\}$ be the index set, ϕ_i be the weight of f_i in processing time, and w_i be the allocated bandwidth ratio for $i \in \mathcal{K}$. Each of the subtrees $\Omega_8 \sim \Omega_{15}$ has only one flow injection, thus we have $\phi_{\Omega_i} = \phi_i$ for $i = 8 \ldots 15$. We can then resolve the link sharing and buffer sharing by the techniques in [4][5]. For example, f_{Ω_8} has link sharing with f_{Ω_9} and f_4 at node R_4, the ESC of R_4 for f_{Ω_8} is calculated as $\hat{\beta}^4_{\Omega_8} = w_{\Omega_8}\beta_4 \otimes \delta_{\phi_{\Omega_9}+\phi_4}$, where $w_{\Omega_8} = \phi_{\Omega_8}/(\phi_{\Omega_8} + \phi_{\Omega_9} + \phi_4)$ is the bandwidth ratio for f_{Ω_8}.

The example in Figure 1 is a basic structure of the binary tree in Figure 3. Following the method in Section III-B2, we first resolve the link sharing at R_1 to obtain the ESC for subtree Ω_2. Then resolve the link sharing and the buffer sharing among f_{Ω_4}, f_{Ω_5} and f_2 at R_2, thus we get the system ESC for f_{Ω_4}. Continue to resolve the link sharing and the buffer sharing at R_4, and we obtain the system ESC for the tag flow f_8. Finally, we have the closed-form delay bound formula as

$$\bar{D} = T_8^{sys} + \frac{b_8}{C_8^{sys}}, \tag{6}$$

where $T_8^{sys} = T_8 + T_2 + \phi_2 + \phi_{\Omega_5} + T_1 + \phi_3 + [b_{\Omega_5} + r_{\Omega_5}(T_2 + \phi_2 + \phi_{\Omega_4}) + b_2 + r_2(T_2 + \phi_{\Omega_4} + \phi_{\Omega_5})]/(w_{\Omega_2}C_1) + r_8[b_9 + r_9T_9 + r_9(T_4 + \phi_{\Omega_8} + \phi_4) + b_4 + r_4(T_4 + \phi_{\Omega_8} + \phi_{\Omega_9})]/[w_{\Omega_4}C_2 \wedge (w_{\Omega_2}C_1 - r_5 - r_{10} - r_{11} - r_2)]$, C_8^{sys} is the system service rate for f_8 calculated in Formula 7 (See Section VI-C2).

C. Parameter Setup for SA

We set the parameters for the experiment following the problem formulation in Section IV-B and the method of determining the SA inputs in Section V-C.

1) Objective Function: We use Formula 4 as the objective function. The function $g(\cdot,\cdot)$ is implemented by cycle accurate simulation with Verilog-HDL to obtain D_{max} with respect to the configuration space. The parameter set $\mathcal{P} = \{b_i, r_i, \phi_i, \phi_{\Omega_j}, C_j, T_j | i = 2 \ldots 15, j = 1 \ldots 15\}$, thus the configuration space $\mathcal{S} = \big\{s = \{p_j\} | p_j \in \mathcal{P}, l_j \le p_j \le h_j\big\}$, where the constraint for parameter p_j is specified as follows.

2) Parameter Constraints: The configuration parameters of the system consist of two categories. One is for fixed parameters and the other is for variable parameters with constraints. We set the service curve of the leaf nodes and the trunk nodes with fixed rate of 1 packet/cycle and no latency, namely $C_i = 1, T_i = 0$ for $i \neq 1$. This setting means that these routers transfer the arrived packet immediately with rate 1.0 packet/cycle. The weights $\phi_{\Omega_i} = \phi_i$, for $i = 8 \ldots 15$, because these leaf nodes just output a single flow rather than an aggregate flow.

In the case study, there are 50 variables in total to be determined by SA. The constraints for these parameters are listed in Table III.

TABLE III
PARAMETER CONSTRAINTS OF THE BINARY TREE FOR SA

Parameters	Constraints
Arrival rate r_i	$0.01 \le r_i \le 0.22$, for $i = 2 \ldots 15$
Flow burstiness b_i	$1 \le b_i \le 16$, and $b_i \in \mathcal{Z}^+$, for $i = 2 \ldots 15$
Flow weight ϕ_i	$1 \le \phi_i \le 5$, and $\phi_i \in \mathcal{Z}^+$, for $i = 2 \ldots 15$
Subtree weight ϕ_{Ω_i}	$1 \le \phi_{\Omega_i} \le 15$, and $\phi_{\Omega_i} \in \mathcal{Z}^+$, for $i = 2 \ldots 7$
Service rate C_1	$0.3 \le C_1 \le 0.99$
Service latency T_1	$2 \le T_1 \le 25$, and $T_1 \in \mathcal{Z}^+$
System ESC rate C_i^{sys}	$r_i \le C_i^{sys}$, for $i = 2 \ldots 15$

The system stability condition ensures that the service provided by the root node is sufficient to serve all the injected flows. The system equivalent service rate C_i^{sys} for f_i is

$$C_i^{sys} = \begin{cases} w_i C_i \wedge (w_{\Omega_i} C_{\lfloor \frac{i}{2} \rfloor} - r_{\Omega_{2i}} - r_{\Omega_{2i+1}}), & i = 2, 3 \\ w_i C_i \wedge \big(w_{\Omega_i} C_{\lfloor \frac{i}{2} \rfloor} \wedge (w_{\Omega_{\lfloor \frac{i}{2} \rfloor}} C_{\lfloor \frac{i}{2^2} \rfloor} \\ -r_{\Psi(\Omega_i)} - r_{\lfloor \frac{i}{2} \rfloor}) - r_{\Omega_{2i}} - r_{\Omega_{2i+1}}\big), & i = 4 \ldots 7 \\ C_i \wedge w_{\Omega_i} C_{\lfloor \frac{i}{2} \rfloor} \wedge \big(w_{\Omega_{\lfloor \frac{i}{2} \rfloor}} C_{\lfloor \frac{i}{2^2} \rfloor} \wedge \\ (w_{\Omega_{\lfloor \frac{i}{2^2} \rfloor}} C_{\lfloor \frac{i}{2^3} \rfloor} - r_{\Psi(\Omega_{\lfloor \frac{i}{2} \rfloor})} - r_{\lfloor \frac{i}{2^2} \rfloor}) \\ -r_{\Psi(\Omega_i)} - r_{\lfloor \frac{i}{2} \rfloor}\big), & i = 8 \ldots 15 \end{cases} \tag{7}$$

where r_{Ω_i} is the arrival rate of f_{Ω_i} which equals the sum of all the arrival rates injected in Ω_i; $\Psi(\Omega_i)$ is a function to calculate the brother of Ω_i, for example $\Psi(\Omega_5) = \Omega_4$. Taking the calculation for f_4 as an example, $C_4^{sys} = w_4 C_4 \wedge \big(w_{\Omega_4} C_2 \wedge (w_{\Omega_2} C_1 - r_{\Omega_5} - r_2) - r_{\Omega_9} - r_8\big)$.

3) SA Inputs: We first run the SA algorithm with randomly selected T_0 and T_{min}, and obtain $\overline{dE} = 7.2$. Set $p_h = 0.95$ and $p_l = 0.01$, we obtain the initial temperature $T_0 = 140.4$ and the termination temperature $T_{min} = 1.6$ by Formula 5, respectively. Set $N = 200$ as the number of annealing iterations, we get the cooling rate $r = 0.98$. As for the initial configuration s_0, the parameters are set as $r_i = 0.02$, $b_i = 1$, $\phi_i = 1$ for $i = 2 \ldots 15$; $\phi_{\Omega_j} = 1, 1, 3, 3, 3, 3$ for $j = 2 \ldots 7$, respectively, and $C_1 = 0.8$, $T_1 = 5$.

D. Results and Analysis

We run the experiments on a 64-bit Linux platform with 4 Intel Xeon 5150 CPUs working at 2.66 GHz. Every Verilog-HDL simulation is run with Modelsim-SE 6.6d for 500,000 cycles. Each configuration in the SA algorithm requires about 8 seconds to do the simulation, data processing and corresponding I/O operations. The whole process costs about 22.2 hours.

The configuration found by SA is listed in Table IV, including the arrival rate r_i, the burstiness b_i, the flow weight ϕ_i of flow f_i for $i = 2 \dots 15$, the weight ϕ_{Ω_i} of subtree Ω_i for $i = 2 \dots 7$ and the calculated system service rate C_i^{sys} for f_i.

TABLE IV
CONFIGURATION OF THE BINARY TREE FOUND BY SA FOR BEST TIGHTNESS

i	r_i	b_i	ϕ_i	ϕ_{Ω_i}	C_i^{sys}
2	0.048	3	3	9	0.114
3	0.041	4	1	7	0.125
4	0.055	13	2	8	0.066
5	0.023	10	2	13	0.066
6	0.024	3	4	1	0.029
7	0.033	15	2	6	0.185
8	0.108	1	3	-	0.119
9	0.038	11	2	-	0.049
10	0.021	4	2	-	0.064
11	0.195	1	3	-	0.238
12	0.038	14	4	-	0.043
13	0.058	14	2	-	0.063
14	0.016	1	4	-	0.168
15	0.032	1	3	-	0.184

TABLE V
RESULTS WITH FIXED WEIGHTS AND SINK SERVICE

i	r_i	b_i	C_i^{sys}
2	0.054	15	0.143
3	0.087	13	0.143
4	0.085	5	0.183
5	0.058	1	0.183
6	0.059	1	0.153
7	0.104	2	0.153
8	0.026	1	0.124
9	0.028	14	0.126
10	0.072	3	0.197
11	0.037	2	0.162
12	0.067	15	0.161
13	0.044	9	0.138
14	0.014	10	0.063
15	0.035	8	0.084

The service rate and the service latency of the root node are $C_1 = 0.846$ and $T_1 = 14$, respectively. The maximum simulation delay D_{max} is 104 cycles obtained from over 100,000 packets as shown in Figure 4, and the delay bound $\bar{D}$ equals 107.3 cycles by Formula 6. Thus, the tightness under this SA-found configuration is $\xi = D_{max}/\bar{D} \times 100\% = 96.9\%$.

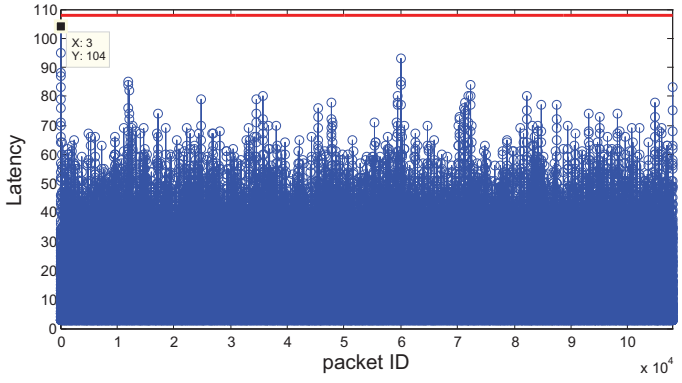

Fig. 4. Simulation result with the SA found configuration. The maximum simulated delay is 104 cycles, and the calculated delay bound is 107.3 cycles.

The results above indicate that the SA based method effectively find the configuration to reach great tightness and can handle very large parameter search space, e.g. 50 dimensions in this experiment. The parameters in Table IV make good sense in the view of worst case occurrence. The contention flows of f_8 includes f_9, f_4, f_{Ω_5}, f_2, f_{Ω_2}, and f_{Ω_3}. It can be verified that the burstiness of these contention flows are all greater than their weights, e.g. $b_9 > \phi_9$. This helps to make the worst case occur when the flows content for the output channels.

E. Further Results and Discussion

The SA based method can also help evaluate the analytical model in real designs with some fixed parameters. For example, the arbitration scheme may fix the flow weights, and the service capability of sinking packets may also have been determined in the design. Taking the binary tree as an example, we may have all the flow weights fixed, $\phi_i = 1$ for $i = 2 \dots 15$ and $\phi_{\Omega_j} = 1, 1, 3, 3, 3, 3$ for $j = 2 \dots 7$, and have the sinking service fixed, $C_1 = 0.8$ and $T_1 = 10$. We keep the same constraints of r_i and b_i as in Table III, and use the same initial values as previously. Thus the quantity of variable parameters decreases to 22. The searching process of SA is reduced to about 9.8 hours. The tightness reaches $\xi = 95.8\%$ with $\bar{D} = 85.6$ cycles and $D_{max} = 82$ cycles. The resulting flow rates r_i and burstiness b_i are listed in Table V.

Our method is topology independent and can also be applied to the case employing flow control, though we do not consider flow control for simplicity in the case study. One just needs to accordingly revise the derivation of the closed-form delay bound formula in Section III-B, and then apply our method.

VII. CONCLUSION

The tightness study of delay bound is an essential and non-trivial problem for QoS design of NoC. The large searching space of the system configuration increases the difficulty of investigating worst case delay and validating the formal analysis. In this paper, we present a simulated annealing aided method for tightness study. The tightness study problem is abstracted as an optimization problem to enable the computer aided configuration search for best tightness. We also propose the simulated annealing algorithm to solve the formulated problem which embeds the simulation into the evaluation of the objective function. The experiment results of an all-to-one communication tree, in which 50 parameters are involved, indicate that our method can effectively handle the complex system with discrete variables and multi-dimensional search space. Not limited to the study of delay bound tightness, this method can also be used for other applications of the similar nature.

In the future, we will apply this method to study the tightness of the buffer backlog bound. This will extend the problem definition from a single-objective to a multi-objective optimization as there are many buffers in the system optimization space.

ACKNOWLEDGMENT

The research is sponsored in part by Intel Corporation through a research gift. In particular, we thank Alexander Gotmanov from Intel Corporation for valuable discussions and advices.

REFERENCES

[1] R. Marculescu, U.Y. Ogras, Li-Shiuan Peh, N.E. Jerger, and Y. Hoskote. Outstanding research problems in NoC design: System, microarchitecture, and circuit perspectives. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 28(1):3–21, 2009.

[2] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, Berlin, Heidelberg, 2001.

[3] Yuming Jiang and Yong Liu. *Stochastic Network Calculus*. Springer, 1 edition, 2008.

[4] Yue Qian, Zhonghai Lu, and Wenhua Dou. Analysis of worst-case delay bounds for best-effort communication in wormhole networks on chip. In *Proc. 3rd ACM/IEEE Int. Symp. Networks-on-Chip*, pages 44–53, 2009.

[5] Yue Qian, Zhonghai Lu, and Wenhua Dou. Analysis of worst-case delay bounds for on-chip packet-switching networks. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 29(5):802–815, 2010.

[6] William James Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufman Publishers, 2004.

[7] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[8] VLADIMÍR Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *IEEE Trans. Inf. Theory*, 45(1):41–51, 1985.

[9] Rene L. Cruz. A calculus for network delay, part I: Network elements in isolation; part II: Network analysis. *IEEE Trans. Inf. Theory*, 37(1), Jan. 1991.

[10] Luciano Lenzini, Linda Martorini, Enzo Mingozzi, and Giovanni Stea. Tight end-to-end per-flow delay bounds in FIFO multiplexing sink-tree networks. *Perform. Eval.*, 63(9):956–987, Oct. 2006.

[11] Zhonghai Lu, Lei Xia, and A. Jantsch. Cluster-based simulated annealing for mapping cores onto 2D mesh Networks on Chip. In *Proc. 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, pages 1–6, Apr. 2008.

[12] D. Gebhardt, Junbok You, and K.S. Stevens. Design of an energy-efficient asynchronous NoC and its optimization tools for heterogeneous SoCs. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 30(9):1387–1399, 2011.

[13] H. Salamy and H. Harmanani. An effective solution to thermal-aware test scheduling on network-on-chip using multiple clock rates. In *IEEE 55th Int. Midwest Symp. Circuits and Systems*, pages 530–533, 2012.