

Towards a Framework to Evaluate Performance of the NoCs

Mahmoud Moadeli and Wim Vanderbauhede

{mahmoudm, wim}@dcs.gla.ac.uk

Abstract

On-chip networks (NoC) emerged as a promising approach to provide a packet-based, structured communication infrastructure to handle the increasing communication requirements of the system on chips. Most applications running on the NoC have to meet a stringent quality of service to perform their normal functionality. Assuring that the manufactured system will satisfy these demands requires binding the non-functional issues to the system specification from the early stages of the development. In this paper we have investigated using PEPA and queuing networks to analyze performance of such systems regarding to a widely adopted switching technique, and also offered a few suggestions to enable performance-aware specification of systems in different levels of abstraction and tools to perform cross-level transitions.

1 Introduction

The NoC is a novel approach to handle the design complexity and communication requirements of large system on chips in a structured and modular fashion. The applications running on a NoC typically must meet their service demands with an acceptable quality to perform satisfactorily. These requirements, however, has to be satisfied at the lowest cost in terms of design complexity and the area in order to realize a system of acceptable size and cost. Obviously, to achieve this goal, the performance related aspects has to be augmented to the system specification from the preliminary stages of the system development. Also tools and techniques are required to analyze and compare such systems in respect to different specifications

to trade-off between cross-cutting concerns and yield the performance measures of interest.

Researches performed in the field have revealed that wormhole switching and deterministic routing are the best options to develop an optimized system on a NoC [1] architecture. In wormhole switching, a packet is split to a number of flits. The header flits has the routing information and followed by the remaining flits. Wormhole switching realizes manufacturing fast and compact switches and is the dominant switching technique in parallel processing and the NoC developments.

In this paper we represent a PEPA model for a simple system adopting wormhole switching in the next section, and in the section 3 we have shown how adopting queuing networks may simplify analysis of larger systems. In the section 4 we briefly discuss the requirements of a comprehensive framework to realize the evaluation of large NoCs, and finally in the section 5 we present the conclusion and the trends for the future works.

2 Markovian Models

It is widely known that the performance evaluation tools and methods that involve solving Markov processes (directly or indirectly) are not appropriate for analyzing systems with large state space. To investigate whether these techniques are suitable for evaluation of the NoCs, we have presented the PEPA model for a simple system depicted in Figure 1. The system is consisted of three nodes communicating through the unidirectional links using wormhole switching. We assume having a uniform traffic in which each node chooses the destinations arbitrarily, and there

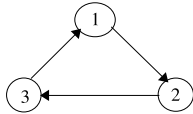


Figure 1: A simple system

is equal *Poisson* traffics between each pair of nodes in the system.

The PEPA [8] model in algorithms 1, 2, 3 and 4 represents the code for node 1, 2, 3, and the interaction between the nodes respectively. Because of the vertex symmetry the PEPA code for all nodes are similar. The code is obviously too complex to easily comprehend. The reason behind this complexity is that to implement the wormhole routing, we are not able to have a compositional view of the system as three communicating components. Each node in the system needs the detailed information about the source and situation of the streams it receives. For example, a node may receive the flits originating from another node which has currently transmitted the last flit of the message and therefore, terminated its collaboration with the destination node. Also, it is important to separate the interaction of the intermediate nodes before and after the establishment of the connection between source and destination. Otherwise, the model may not yield accurate results.

To start transmission two nodes, say x and y need to setup a connection. The action to establish a connection is denoted by *setupxy*. This action requires involvement of both x and y to be accomplished. Having the connection established, two nodes may start transmitting and receiving the flits of a packet. The action to perform transmission is called *sendxy* and again requires co-operation of both nodes. After transmission the nodes perform the action *relxy* collaboratively to release the connection.

To simplify reading our code we have set a number of rules for naming the states. Underlines () separate different connections in the path. Thus, *setupxy_z* denotes setting up a connection between x and y which is a part of the connection between x to z .

To be able to derive precise results from our model

we require to represent any possible states in our system. While a message has established a number of connections of a particular path, it may demonstrate different behavior than when the source and destinations are connected. For example, before the full connection is established non of the intermediate nodes may perform a release action. While, the action is allowed after the full connection is set up.

We represent the situations where a message is in a blocked state with w in the state's name. For example, $N1 - 12w3$ denotes the state of the *node1* where the connection between *node1* and *node2* is established but it requires the connection between *node2* and *node3* to change its state to $N1 - 123$.

Another convention in our naming is using c to represent a canceled connection. For example $Nx - cx - yz$ shows the state of the *nodex* after it sends last flit of the packet and terminates its collaboration in transmission, while *nodey* and *nodez* are still involved in the transmission started by *nodex*. Obviously, in this state *nodex*, is allowed to take part in another transmission. This situation has been shown by two underlines ().

Apparently, the size and complexity of such models prohibitively grows as the number of nodes increases. Thus, modification, verification and debug of such models in a system consisting of tens of nodes is not unlikely to be impractical. More importantly, the current methods and techniques for solving Markovian processes easily end up state explosion while facing such systems.

In the example above we selected PEPA to evaluate our model. Although, it provides a level of abstraction over using Markovian process directly, the model is still too complex to handle and unlikely to scale. It is not very hard to deduce that modeling the system with stochastic Petri nets [9] would be almost as complicated as developing the PEPA model. This is because both PEPA and SPN eventually map to an underlying Markov process. Therefore, to handle the complexity and the size of the real NoC models, which are at least hundreds of times larger and more complicated than our simple model, we require to analyze the system at a higher level of abstraction. In the next section we investigate how adopting the queuing networks to analyze the NoCs may address

existing problems.

3 Analytical methods based on queuing networks

The nature of problems in the NoC domain is to a large extent similar to the traditional problems in parallel processing and distributed systems. Therefore, the results of the research carried out in these fields may be applied to the NoCs with some slight modifications. The objective of the most performance related researches performed in the field typically has been obtaining and comparing measures of interest in respect to adopting a particular topology, routing algorithm, switching technique and traffic behavior.

A widely accepted class of approaches to analyze the systems analytically have been those in which the system is modeled as a queuing network. Evaluating a system using this method involves decomposing the system to a number of components, obtaining detailed traffic information for each component, and finding appropriate queuing systems to model the components. Using this approach to model an interconnection network, channels and their associated buffers are usually reggraded as the servers while the IPs are customer generators.

With such view of the system as long as we assume that system is stable, nodes generate *Poisson* traffic and channels are modeled as $M/M/1$ queues, deriving the performance measures are quite simple. The add/split property of *Poisson* distribution enables calculating the traffic on each individual channel. Consequently applying the results to the well known equations for $M/M/1$ queues yields the measures of interest. However, analyzing the system would not be that trivial if non-Poisson traffic distributions or any queues rather than Markovian queues needed to be taken into account.

An example of adopting non-Markovian queues was the model presented by Ghosh and Draper [2] to calculate average latency in the k -ary n -cube interconnection networks using wormhole routing. In their method they modeled each channel as an $M/G/1$ queue in which the the service time of each

channel depends on the service and waiting times of the subsequent channels. Because of the dependency of intermediate channels on the subsequent channels they had to analyze the paths reversely, i.e. from destination to the source. Each intermediate channel requires detailed traffics coming form other links and some stochastic information regarding the destinations of the streams in order to precisely calculate the expected latency experienced at the channel.

In their method Ghosh and Draper [2] made a numbers of assumptions that are not realistic such as assuming *Poisson* traffic to each channel while channels are modeled as $M/G/1$ queues. Nevertheless, comparing the outcomes of the evaluation with simulation results for k -ary n -cubes and other architectures such as the mesh [7] architecture shows that their analytical evaluation well approximate the simulation results and those assumption do not have a significant impact on the accuracy of their evaluations.

Applying this method to the real systems however, may not be that simple. In the real systems nodes generate message at different rates and distributions; the network does not necessarily have a regular topology; and it is unlikely to find patterns to simplify the analysis. Thus, the traffic analysis and consequently evaluation of the system requires explicit involvement of all channels and nodes. This approach obviously necessitates repeating all calculations should any minor changes are applied to the system. Moreover, the traffic analysis will be much more complicated if virtual channels and adaptive routings also have to be taken into account.

4 A Framework to Evaluate the NoCs

In this section we present the tools and techniques required to develop performance-aware specification of the system at different levels of abstraction and explain briefly the transitions from the abstract levels down to the more detailed levels which is eventually aimed to yield the performance measures. Figure 2 represents different layers and cross-layer transitions that is going to be discussed briefly.

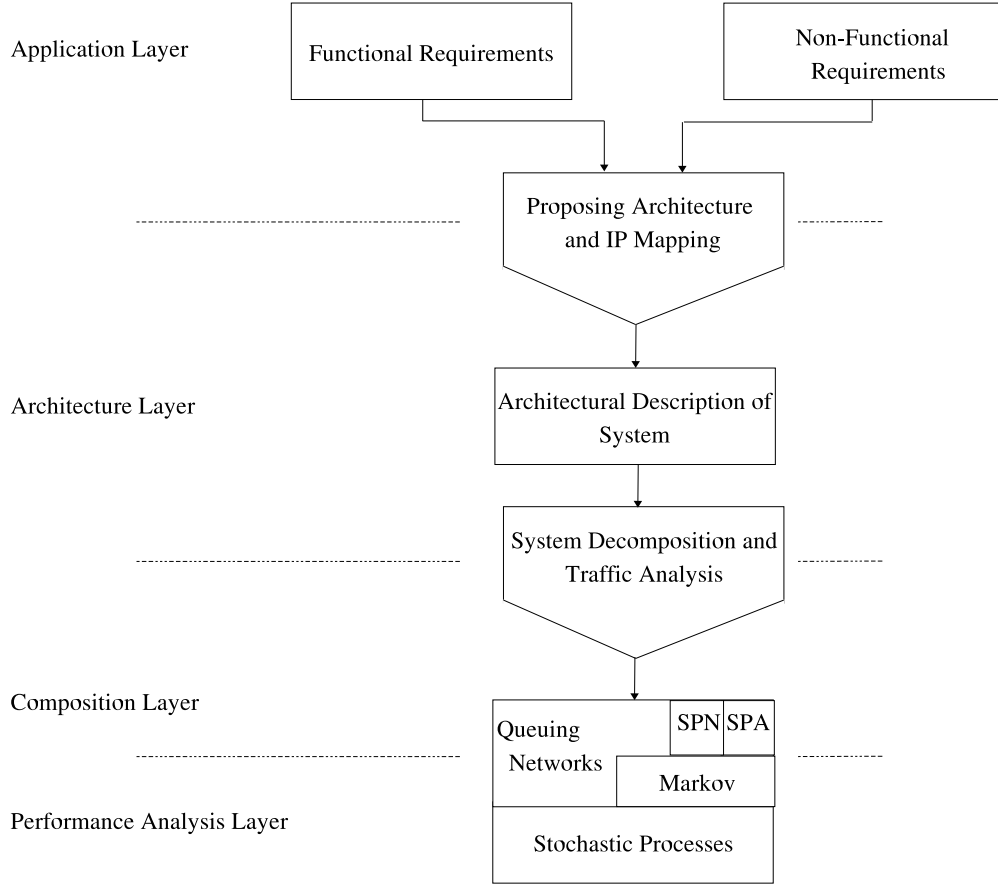


Figure 2: System specification at different levels of abstraction

4.1 Application Layer

At the highest level of abstraction which is called application layer an application may be presented by a graph in which the vertexes are functional IPs and the edges show the rate and the direction of communication between IPs. Obviously, this method of specification does not convey any non-functional aspects such as energy, latency and throughput which an application may intends to be optimized for. Presenting the functional and non-functional specifications of a system together at this level will be crucial for the development of large systems in which normal functionality of the system heavily depends on the quality of non-functional aspects.

A system presented at this level may be analyzed along with the non-functional requirements to produce an architecture that optimize some performance measures. The analysis at this level does not yield detailed performance information and it is mainly about high level features of the system such as adjusting the IP positions in different architectures in order to provide traffic locality, or proposing a particular routing algorithm to minimize the latency.

So far there has been a number of researches that specify the position of IPs in a NoC architecture to optimize energy [6, 5, 4]. More advanced tools may propose the architecture as well as the position of each IP in the architecture in order to minimize en-

ergy consumption [3]. However, to the best of our knowledge, no analytical research has been accomplished to optimize the system regarding to other performance measures such as latency and throughput.

4.2 Architecture Layer

In architecture level we may present the model with more realistic information including switching, routing and topology. Detailed physical characteristics of the communication infrastructure such as the number of virtual channels sharing a physical channel may also be specified in this level. A system at architecture layer may be described by an architectural description language that enable expressing some features of the real system which are required for the lower layers to analyze the system performance. A system presented here could be the results of the analysis tool developed at the application layer which may be manually modified here.

The state of the system at this level is still too high to be handled by available performance evaluation tools. Therefore, it is required to decompose the system to different components in such a way that enables deriving the performance of each components in isolation. The results of the analysis at this stage may feed to a compositional performance evaluation approach such as PEPA or queuing networks at lower levels.

The IPs on a chip may take different routes to communicate and each channel in the path may consisted of a number of virtual channels, a major challenge at this level is on-chip traffic analysis and engineering to assure that lower layers have precise and enough information to analyze each component.

Description and analysis of the system at this level is another topic of research in performance evaluation of large NoCs.

4.3 Composition Layer

As mentioned in the previous sections having a compositional view of the system enables analyzing larger systems. Most widely used performance evaluation tools and techniques such queuing networks, SPA and SPN enforce a compositional view of the system.

Although, SPAs have shown to be powerful tools to derive performance measures of interest in general, some peculiarities such as switching techniques and size of the systems are prohibitive to adopting them in NoC domain. Researches carried out in similar domains such as parallel systems reveals that queuing networks may be considered as an alternative.

4.4 Performance Analysis Layer

The last but not the least layer to be discussed is the performance analysis layer in which the actual performance analysis takes place. As we pointed out, methods that directly map to Markov process may not well scale for the NoC domain. A promising alternative is adopting queuing networks. However, the queuing models presented so far has been verified for the simplest forms of traffics. Testing their credibility for more complicated and realistic traffics and proposing more accurate and realistic queuing models requires extensive research in the field.

5 Conclusion and the Future works

In this paper we discussed using SPA and queuing networks as two widely used methods to model a system adopting wormhole switching. The results of our experiments and researches performed by others reveals that since queuing networks provide a level of abstraction over Markovian models they are more appropriate tools for analysis of large systems such as NoCs. Also we investigated the requirements of a comprehensive performance evaluation framework. We showed that such framework requires languages to express the performance-aware description of a system at different levels of abstraction and synthesis tools are needed to enable cross-layer transitions. Moreover, the performance evaluation techniques, in particular queuing networks has to be researched for the more realistic queuing models. Our future work will be proposing description, synthesis and analysis tools and techniques at different levels of abstraction for the NoC performance evaluation.

References

- [1] Umit Y. Ogras, Jingcao Hu, Radu Marculescu, "Key Research Problems in NoC Design: A Holistic Perspective", International Conference on Hardware - Software Codesign and System Synthesis, September, 2005.
- [2] Jeffrey T. Draper , Joydeep Ghosh, "A comprehensive analytical model for wormhole routing in multicomputer systems", Journal of Parallel and Distributed Computing, v.23 n.2, p.202-214, Nov. 1994.
- [3] S. Murali, G. De Micheli, "SUNMAP: A Tool for Automatic Topology Selection and Generation for NoCs", In Proc. DAC, 2004.
- [4] Giuseppe Ascia, Vincenzo Catania, Maurizio Palesi: "Multi-objective mapping for mesh-based NoC architectures", CODES+ISSS 2004: 182-187.
- [5] S. Murali, G. De Micheli, "Bandwidth-Constrained Mapping of Cores onto NoC Architectures," Design, Automation and Test in Europe Conference and Exhibition Volume II, February, 2004.
- [6] Jingcao Hu, Radu Marculescu, "Energy- and performance-aware mapping for regular NoC architectures". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 24(4), April 2005.
- [7] R. Greenberg and L. Guan, "Modelling and Comparison of Wormhole Routed Mesh and Torus Networks," Proc. Ninth IASTED Int'l Conf. Parallel and Distributed Computing and Systems, 1997.
- [8] Jane Hillston, "A Compositional Approach to Performance Modeling", Cambridge University Press, 1996
- [9] F. Bause, P. S. Kritzinger, "Stochastic Petri Nets – An Introduction to the Theory" Vieweg Verlag, Germany, 2002.

Algorithm 1 The PEPA code for node 1.

```

N1_0 = (nh, λ1).N1_0 + (setup12, λsetup).N1_12 + (setup31, λsetup).N1_31 + (setup12_3, λsetup).N1_12w3 +
(setup31_2, λsetup).N1_31w2 + (setup2_31, λsetup).N1_231;

N1_12 = (send12, λ).N1_12 + (rel12, λrel).N1_0 + (setup31, λsetup).N1_12_31 + (setup2_31, λsetup).N1_231;
N1_31 = (send31, λ).N1_31 + (rel31, λ31).N1_0 + (setup12, λsetup).N1_12_31 + (setup12_3, λsetup).N1_12w3;
N1_12_31 = (send12, λ).N1_12_31 + (rel12, λrel).N1_31 + (send31, λ).N1_12_31 + (rel31, λrel).N1_12;
N1_12w3 = (send12w3, λ).N1_12w3 + (nh, λ1).N1_12w3 + (send12_3, λ).N1_123 + (setup31, λsetup).N1_31_12w3 +
(setup2_31, λsetup).N1_231_12w3;
N1_123 = (send12_3, λ).N1_123 + (rel12_3, λrel).N1_0 + (setup31, λsetup).N1_31_123;
N1_31_12w3 = (send12w3, λ).N1_31_12w3 + (send12_3, λ).N1_31_123 + (send31, λ).N1_31_12w3 + (rel31,
1).N1_12w3;
N1_231_12w3 = (send12w3, λ).N1_231_12w3 + (send2_31, λ).N1_231_12w3 + (send12_3, λ).N1_123__c2_31;

N1_31_123 = (send31, λ).N1_31_123 + (rel31, λ31).N1_123 + (send12_3, λ).N1_31_123 + (rel12_3, λrel).N1_31;

N1_31w2 = (send31w2, λ).N1_31w2 + (setup3_12, λsetup).N1_312 + (nh, 1).N1_31w2;

N1_312 = (send31_2, λ).N1_312 + (rel31_2, λrel).N1_c3_12 + (send3_12, λ).N1_312;

N1_c3_12 = (send3_12, λ).N1_c3_12 + (rel3_12, λrel).N1_0 + (setup31, λsetup).N1_31__c3_12 + (setup31_2,
λsetup).N1_31w2__c3_12;

N1_31__c3_12 = (send3_12, λ).N1_31__c3_12 + (rel3_12, λrel).N1_31 + (send31, λ).N1_31__c3_12 + (rel31,
λrel).N1_c3_12;

N1_31w2__c3_12 = (send31_2, λ).N1_31w2__c3_12 + (rel3_12, λrel).N1_31w2 + (send31, λ).N1_31w2__c3_12;

N1_231 = (send2_31, λ).N1_231 + (rel2_31, λ).N1_0 + (setup12, λsetup).N1_12__231 + (setup12_3,
λsetup).N1_12w3__2_31;

N1_12__231 = (send12, λ).N1_12__231 + (rel12, λrel).N1_231 + (send2_31, λ).N1_12__231 + (rel2_31, λrel).N1_12;

N1_12w3__2_31 = (send2_31, λ).N1_12w3__2_31 + (send12_w3, λ).N1_12w3__2_31 + (send12_3,
λ).N1_123__c2_31;

N1_123__c2_31 = (send2_31, λ).N1_123__c2_31 + (rel2_31, λrel).N1_123 + (send12_3, λ).N1_123 + (rel12_3,
λrel).N1_c2_31;

N1_c2_31 = (send2_31, λ).N1_c2_31 + (rel2_31, λrel).N1_0 + (setup12, λsetup).N1_12__c2_31 + (setup12_3,
λsetup).N1_c2_31__12w3;

N1_12__c2_31 = (send12, λ).N1_12__c2_31 + (rel12, λrel).N1_c2_31 + (send2_31, λ).N1_12__c2_31 + (rel2_31,
λrel).N1_12;

N1_c2_31__12w3 = (send2_31, λ).N1_c2_31__12w3 + (rel2_31, λrel).N1_12w3 + (send12_w3, λ).N1_c2_31__12w3
+ (send12_3, λ).N1_c2_31__123;

N1_c2_31__123 = (send12_3, λ).N1_c2_31__123 + (rel12_3, λrel).N1_c2_31 + (send2_31, λ).N1_c2_31__123 +
(rel2_31, λrel).N1_123;

```

Algorithm 2 The PEPA code for node 2.

```

N2_0 = (nh, 1).N2_0 + (setup23, λsetup).N2_23 + (setup12, λsetup).N2_12 + (setup23_1, λsetup).N2_23w1 + (setup12_3,
λsetup).N2_12w3 + (setup3_12, λsetup).N2_312;

N2_23 = (send23, λ).N2_23 + (rel23, λrel).N2_0 + (setup12, λsetup).N2_23_12 + (setup3_12, λsetup).N2_312;

N2_12 = (send12, λ).N2_12 + (rel12, λrel).N2_0 + (setup23, λsetup).N2_23_12 + (setup23_1, λsetup).N2_23w1;

N2_23_12 = (send23, λ).N2_23_12 + (rel23, λrel).N2_12 + (send12, λ).N2_23_12 + (rel12, λrel).N2_23;

N2_23w1 = (send23w1, λ).N2_23w1 + (nh, 1).N2_23w1 + (send23_1, λ).N2_231 + (setup12, λsetup).N2_12_23w1 + (setup3_12,
λsetup).N2_312_23w1;

N2_231 = (send23_1, λ).N2_231 + (rel23_1, λrel).N2_0 + (setup12, λsetup).N2_12_231;

N2_12_23w1 = (send23w1, λ).N2_12_23w1 + (send23_1, λ).N2_12_231 + (send12, λ).N2_12_23w1 + (rel12, λrel).N2_23w1;

N2_312_23w1 = (send23w1, λ).N2_312_23w1 + (send3_12, λ).N2_312_23w1 + (send23_1, λ).N2_231__c3_12;

N2_12_231 = (send12, λ).N2_12_231 + (rel12, λrel).N2_231 + (send23_1, λ).N2_12_231 + (rel23_1, λrel).N2_12;

N2_12w3 = (send12w3, λ).N2_12w3 + (setup1_23, λsetup).N2_123 + (nh, 1).N2_12w3;

N2_123 = (send12_3, λ).N2_123 + (rel12_3, λrel).N2_c1_23 + (send1_23, λ).N2_123;

N2_c1_23 = (send1_23, λ).N2_c1_23 + (rel1_23, λrel).N2_0 + (setup12, λsetup).N2_12__c1_23 + (setup12_3,
λsetup).N2_12w3__c1_23;

N2_12__c1_23 = (send1_23, λ).N2_12__c1_23 + (rel1_23, λrel).N2_12 + (send12, λ).N2_12__c1_23 + (rel12,
λrel).N2_c1_23;

N2_12w3__c1_23 = (send12_3, λ).N2_12w3__c1_23 + (rel1_23, λrel).N2_12w3 + (send12, λ).N2_12w3__c1_23;

N2_312 = (send3_12, λ).N2_312 + (rel3_12, λrel).N2_0 + (setup23, λsetup).N2_23__312 + (setup23_1,
λsetup).N2_23w1__3_12;

N2_23__312 = (send23, λ).N2_23__312 + (rel23, λrel).N2_312 + (send3_12, λ).N2_23__312 + (rel3_12, λrel).N2_23;

N2_23w1__3_12 = (send3_12, λ).N2_23w1__3_12 + (send23_w1, λ).N2_23w1__3_12 + (send23_1, λ).N2_231__c3_12;

N2_231__c3_12 = (send3_12, λ).N2_231__c3_12 + (rel3_12, λrel).N2_231 + (send23_1, λ).N2_231 + (rel23_1,
λrel).N2_c3_12;

N2_c3_12 = (send3_12, λ).N2_c3_12 + (rel3_12, λrel).N2_0 + (setup23, λsetup).N2_23__c3_12 + (setup23_1,
λsetup).N2_c3_12__23w1;

N2_23__c3_12 = (send23, λ).N2_23__c3_12 + (rel23, λrel).N2_c3_12 + (send3_12, λ).N2_23__c3_12 + (rel3_12,
λrel).N2_23;

N2_c3_12__23w1 = (send3_12, λ).N2_c3_12__23w1 + (rel3_12, λrel).N2_23w1 + (send23_w1, λ).N2_c3_12__23w1 +
(send23_1, λ).N2_c3_12__231;

N2_c3_12__231 = (send23_1, λ).N2_c3_12__231 + (rel23_1, λrel).N2_c3_12 + (send3_12, λ).N2_c3_12__231 + (rel3_12,
λrel).N2_231;

```

Algorithm 3 The PEPA code for node 3.

$N3_0 = (nh, 1).N3_0 + (setup31, \lambda_{setup}).N3_31 + (setup23, \lambda_{setup}).N3_23 + (setup31_2, \lambda_{setup}).N3_31w2 + (setup23_1, \lambda_{setup}).N3_23w1 + (setup1_23, \lambda_{setup}).N3_123;$

$N3_31 = (send31, \lambda).N3_31 + (rel31, \lambda_{rel}).N3_0 + (setup23, \lambda_{setup}).N3_31_23 + (setup1_23, \lambda_{setup}).N3_123;$

$N3_23 = (send23, \lambda).N3_23 + (rel23, \lambda_{rel}).N3_0 + (setup31, \lambda_{setup}).N3_31_23 + (setup31_2, \lambda_{setup}).N3_31w2;$

$N3_31_23 = (send31, \lambda).N3_31_23 + (rel31, \lambda_{rel}).N3_23 + (send23, \lambda).N3_31_23 + (rel23, \lambda_{rel}).N3_31;$

$N3_31w2 = (send31w2, \lambda).N3_31w2 + (nh, 1).N3_31w2 + (send31_2, \lambda).N3_312 + (setup23, \lambda_{setup}).N3_23_31w2 + (setup1_23, \lambda_{setup}).N3_123_31w2;$

$N3_312 = (send31_2, \lambda).N3_312 + (rel31_2, \lambda_{rel}).N3_0 + (setup23, \lambda_{setup}).N3_23_312;$

$N3_23_31w2 = (send31w2, \lambda).N3_23_31w2 + (send31_2, \lambda).N3_23_312 + (send23, \lambda).N3_23_31w2 + (rel23, \lambda_{rel}).N3_31w2;$

$N3_123_31w2 = (send31w2, \lambda).N3_123_31w2 + (send1_23, \lambda).N3_123_31w2 + (send31_2, \lambda).N3_312_c1_23;$

$N3_23_312 = (send23, \lambda).N3_23_312 + (rel23, \lambda_{rel}).N3_312 + (send31_2, \lambda).N3_23_312 + (rel31_2, 1).N3_23;$

$N3_23w1 = (send23w1, \lambda).N3_23w1 + (setup2_31, \lambda_{setup}).N3_231 + (nh, 1).N3_23w1;$

$N3_231 = (send23_1, \lambda).N3_231 + (rel23_1, \lambda_{rel}).N3_c2_31 + (send2_31, \lambda).N3_231;$

$N3_c2_31 = (send2_31, \lambda).N3_c2_31 + (rel2_31, \lambda_{rel}).N3_0 + (setup23, \lambda_{setup}).N3_23_c2_31 + (setup23_1, \lambda_{setup}).N3_23w1_c2_31;$

$N3_23_c2_31 = (send2_31, \lambda).N3_23_c2_31 + (rel2_31, \lambda_{rel}).N3_23 + (send23, \lambda).N3_23_c2_31 + (rel23, \lambda_{rel}).N3_c2_31;$

$N3_23w1_c2_31 = (send23_1, \lambda).N3_23w1_c2_31 + (rel2_31, \lambda_{rel}).N3_23w1 + (send23, \lambda).N3_23w1_c2_31;$

$N3_123 = (send1_23, \lambda).N3_123 + (rel1_23, \lambda_{rel}).N3_0 + (setup31, \lambda_{setup}).N3_31_123 + (setup31_2, \lambda_{setup}).N3_31w2_1_23;$

$N3_31_123 = (send31, \lambda).N3_31_123 + (rel31, \lambda_{rel}).N3_123 + (send1_23, \lambda).N3_31_123 + (rel1_23, 1).N3_31;$

$N3_31w2_1_23 = (send1_23, \lambda).N3_31w2_1_23 + (send31_w2, \lambda).N3_31w2_1_23 + (send31_2, \lambda).N3_312_c1_23;$

$N3_312_c1_23 = (send1_23, \lambda).N3_312_c1_23 + (rel1_23, 1).N3_312 + (send31_2, \lambda).N3_312 + (rel31_2, 1).N3_c1_23;$

$N3_c1_23 = (send1_23, \lambda).N3_c1_23 + (rel1_23, \lambda_{rel}).N3_0 + (setup31, \lambda_{setup}).N3_31_c1_23 + (setup31_2, \lambda_{setup}).N3_c1_23_31w2;$

$N3_31_c1_23 = (send31, \lambda).N3_31_c1_23 + (rel31, \lambda_{rel}).N3_c1_23 + (send1_23, \lambda).N3_31_c1_23 + (rel1_23, \lambda_{rel}).N3_31;$

$N3_c1_23_31w2 = (send1_23, \lambda).N3_c1_23_31w2 + (rel1_23, \lambda_{rel}).N3_31w2 + (send31_w2, \lambda).N3_c1_23_31w2 + (send31_2, \lambda).N3_c1_23_312;$

$N3_c1_23_312 = (send31_2, \lambda).N3_c1_23_312 + (rel31_2, \lambda_{rel}).N3_c1_23 + (send1_23, \lambda).N3_c1_23_312 + (rel1_23, \lambda_{rel}).N3_312;$

Algorithm 4 Interaction between components of the system

$(N1_0 < setup12, rel12, send12, setup12_3, rel12_3, send12_3, setup3_21, rel3_12, send3_12, send12w3 > N2_0) < setup31, setup23, rel31, rel23, send31, send23, setup1_23, rel1_23,$
 $send1_23, setup3_12, rel3_12, send3_12, setup2_31, rel2_31, send2_31, send23w1, send31w2 > N3_0$
