Meta-Modelling of Interconnection Networks with Meta-Modelling Tool $ATOM^3$

Sina Meraji
McGill University, Montreal, Quebec, Canada
smeraj@cs.mcgill.ca

School of Computer Science

Abstract. This paper shows how Meta-Modelling is used to generate an appropriate input for interconnection network simulators which is Xmulator [8] in our case. We develop a Meta-Model for different hypercube topologies and also a Meta-Model for each node in hypercube topology. We use the Entity-Relationships formalism in the Meta-Modelling tool $AToM^3$ [1] to design our Meta-models and then use the Model-Transformation to generate the appropriate XML file for the Xmulator. Finally, the paper illustrates some of the results which got from Xmulator.

1 Introduction

A set of processors with private local memories where communicate through message passing is named as interconnection networks. There are three main approaches for performance evaluation of interconnection networks. The first one is monitoring the behaviour of the actual system; it can capture the effects of low-level design choices, but restricts experimentation since it can be prohibitively expensive and time-consuming to change these features. Modeling is the second approach for performance evaluation of interconnection network. It provides a cost-effective way to explore design issues but there are simplifying assumptions in analytical models that degrade the accuracy of the evaluation. The last way to evaluate the performance of an interconnection network is to model its topology and then use some simulation environment which can provide an extensible framework for evaluating the performance of the corresponding model. Simulation has many features and advantages over the other two methods;it can Model different switching and routing algorithms, different flow control policies and interconnection topologies with high accuracy [6].

The use of models in system engineering is increased very fast. As an example models are used in analysing the behaviour of system, complexity management, safety and reliability analysis to bridge conceptual differences between domains and facilitate the construction of Co-Design hardware/software systems. While scientific modeling shows the process of generating abstract, conceptual, graphical or mathematical models, Meta-Modelling is the process of designing languages through Meta and Meta-Meta notations. Meta-Modeling is helping the

handling and combination of different formalisms (e.g. State-charts, Petri Nets and DEVS) through modeling of a model. Meta-Models are also useful to ensure syntactically correct specifications as well as in the construction of customizable designs. A design is a model of the desired system which exhibits structure through the use of Architecture and Class Diagrams [11, 12].

Any design that allows its architect to create highly extensible, portable and exchangeable information in multiple ways is a type of Meta-Model. XML and UML are good examples of a design that utilize Meta-Modelling. The main ideas of Meta-Modelling are i) interchanging data between different tools which facilitates the connection between different hardware/software systems. ii) Constraint Based Structural Analysis of Specifications that leads to development of domain specific languages and ease of specification analysis [10].

In this paper we combine the concept of Meta-Modeling and simulation to evaluate the performance of hypercube family (one of the most famous interconnection topologies) which consists of binary n-cube, necklace hypercube and stretched hypercube topologies. We use $ATOM^3$ [1] which is a visual tool for Meta-Modelling and Model-Transforming to design a Meta-Model for hypercube family in which user can define different hypercube topologies. The $ATOM^3$'s Meta-layer allows a high-level description of models, based on which $ATOM^3$ can automatically generate a tool tailored to the family of those models. Afterward we use model transformation which converts the defined topology to an XML file. The generated XML file can be parsed by Xmulator [8] which is a simulator for interconnection networks. The final results are produced as an XML file, then the user can change different parameters in the Meta-Model and see their effects on the performance of the network.

The rest of the paper is organized as follows in section 2 we introduce two famous hypercube subset topologies. Section 3 shows the Meta-Modelling of hypercube family. In section 4 we show the Model Transformation to XML file which is the input of the Xmulator. Section 5 shows the performance evaluation of the hypercube family. Finally in section 6 we conclude this study.

2 Hypercube Family

The most famous topology of hypercube family is the binary n-cube interconnection network which is introduced in [3]. Here we introduce other two subsets of this family.

2.1 Necklace Hypercube

Hypercube has many desirable properties, however it is not scalable, i.e. when adding some few nodes to it, we must duplicate the network size to reach the next specified network size. The necklace hypercube, introduced in [7], is a new

interconnection network based on hypercube network which constructed by appending a necklace of processors (or nodes, interchangeably) to each edge. We connect i^{th} dimension neighbours by an array of nodes, as a necklace. The necklace length may be fixed or variable for different edge necklaces. In the former, there are fixed number of nodes between each two adjacent neighbours on a necklace. The network is called regular necklace hypercube, and can be defined as $(n, k) - RNH$, where n is the number of dimensions and k is the necklace size. It has many desirable properties such as its scalability and efficient VLSI layout that make it more attractive than the hypercube network [7]. Actually necklace-hypercube is an undirected graph $G = (V, E)$, where $u \in V$ is defined as $u = (b, d, i)$ with b (denoting the Base Vertex) being the hypercube node address of the node with smaller address in its dimension, d (difference), $0 \leq d \leq n$, is the dimension of the necklace (containing the node), and i (as the index) is the index of the node in its necklace [4]. Note that d for the hypercube vertices (or base vertices) is zero. A $(3, 4) - RNH$ is shown in figure 1.

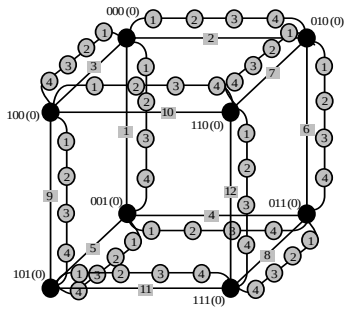


Fig. 1. A regular necklace hypercube with n=3 and k=4

Here we also show a deterministic routing algorithm for the necklace hypercube which is based on e-cube [3] routing algorithm. Consider a minimal one-to-one routing from node $u = (b_1, d_1, i_1)$ to node $v = (b_2, d_2, i_2)$. Let us start with an example in the $(3, 4) - RNH$ network (figure 1) with source node $u = (100, 10, 001)$ and destination node $v = (011, 11, 100)$. Thus, the base vertices for the source node are $s_1 = 100$ and $s_2 = 110$, and for the destination node are $d_1 = 011$ and $d_2 = 111$. Let for any node $x, NBV(x)$ be the distance to the neighbour base vertex. We have $NBV(s_1) = 1, NBV(s_2) = 2, NBV(d_1) = 2, NBV(d_2) = 1$. Note that $H(s_1, d_1) = 3$ indicates a distance of $3 + 1 + 2 = 6$, $H(s_1, d_2) = 2$ indicates a distance of $2 + 1 + 1 = 4$, $H(s_2, d_1) = 1$ indicates a distance of $2 + 2 + 2 = 6$, and $H(s_2, d_2) = 1$ indicates a distance of $1 + 2 + 1 = 4$ hops. Thus, the minimum distance is 4 hops, and the optimal path is $(u, 100, 110, 111, v)$. Figure 2 shows the routing algorithm between nodes $u = (b_1, d_1, i_1)$ and $v = (b_2, d_2, i_2)$ in pseudo code. An adaptive routing algorithm for necklace hypercube topology is proposed in [7].

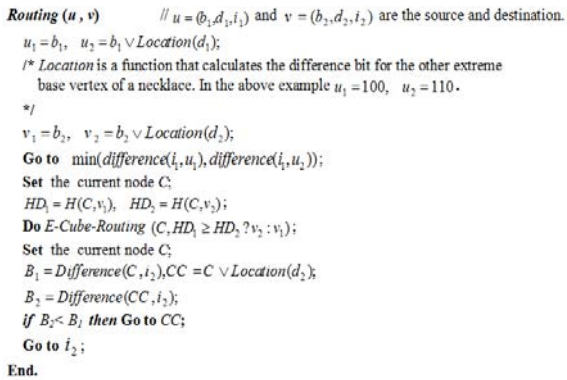


Fig. 2. One-to-one routing algorithm for Necklace Hypercube

2.2 Stretched Hypercube

Stretched hypercube [9] is also tried to overcome the scalability problem by placing some processor nodes on edges of hypercube graph, thus achieving a far more scalable interconnection network. Stretched hypercube networks possess lower average degree than the same size hypercube networks. Accordingly, stretched hypercube networks are of lower average node degree in comparison with hypercube networks when equal network degree is considered.

Definition 1. *Let $G = (V_G, E_G)$ be a hypercube graph, the Regular Stretched hypercube network, $RS_rH = (V_{RS}, E_{RS})$, is an undirected graph based on G, where each edge of G is replaced by an array of r nodes. That is $VRS = (b, b', i)|(b = b'\&i = 0)or(b < b'\& < label^{-1}(b), label^{-1}(b') > E_G\&0 < i \leq r)$, Where label is a bijective function as label: $V_G \rightarrow [|V_G|] = 1, 2, \cdots, |V_G|$.*

From now on, we shall refer to every label value as a base graph node. For each $u = (b, b', i) \in V_{RS}, b$ (base vertex) and b' (last vertex) are two adjacent nodes in the base hypercube, and $i, 0 \leq i \leq r$, represents the index of node u in the array. Conventionally, we apply zero to the index of the base graph nodes and we set $b' = b$ for such nodes; as a result, the base graph vertices can be addressed uniquely. The edge-set of the stretched hypercube can be defined as $E_{RS} = \{\langle u, v\rangle \,|u = (b_1, b'_1, i_1), v = (b_2, b'_2, i_2) \in V_G\}$where nodes u and v must satisfy one of the following conditions:

- Array edges: $b_1 = b_2, b'_1 = b'_2, |i_1 - i_2| = 1$
- Junction edges: $b_1 = b_2, i_1 = 0, i_2 = 1; or\ b'_1 = b_2, i_1 = r, i_2 = 0$

Figure 3 shows an example of regular stretched hypercube. One deterministic and one adaptive routing algorithm for stretched hypercube is presented in [5]. The main idea is using the basic structure of deterministic routing algorithm in necklace hypercube. We omit the details because of the page limit.

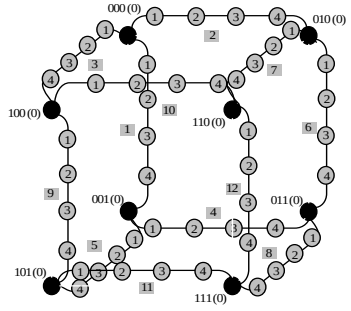


Fig. 3. A regular stretched hypercube with n=3 and k=4

3 Meta Models for Interconnection Networks

A Meta-Model of a given formalism specifies the syntax aspect of the formalism by defining the language constructs and how they are built-up in terms of other constructs. To construct Meta-Models for interconnection networks, we used Entity Relationships (ER) diagram which is a default Meta-Formalism of $ATOM^3$. Important properties of each construct(Entity) are *Cardinality*, *Attributes* and *Appearance*. Cardinality determines the possible number of incoming and outgoing connections of a construct. We populate each construct's Attributes property (of collection type) with a minimum set of regular attributes that supports the semantics of the construct alone and in combination with other constructs [2].

The main elements of a processing element in an interconnection network are shown in figure 4. We divide these parts to two main categories and develop a Meta-Model for each of them. These two Meta-Models are called Hypercube and SwitchInfo. The details of each Meta-Model are described in the following subsections.

3.1 Meta-Hypercube

This Meta-Model consists of three main elements of figure 4 which are essential to build an interconnection topology. These three elements are Switch, physical channel sender(PCS) and physical channel receiver(PCR). The main principle of the switch is to do switching between sender and receiver physical channels. We have two kinds of switches to distinguish main and necklace nodes of a hypercube topology. Different hypercube topologies can be constructed by connecting these elements together. Figure 5 shows the relation between different constructs of this Meta-Model. While each main switch can have N PCSs and PCRs, the necklace switches can only have one PCS and PCR because of the necklace and stretched hypercube structures. Each PCS can connect to a PCR and vice versa. All of these constructs have a name attribute which is unique.

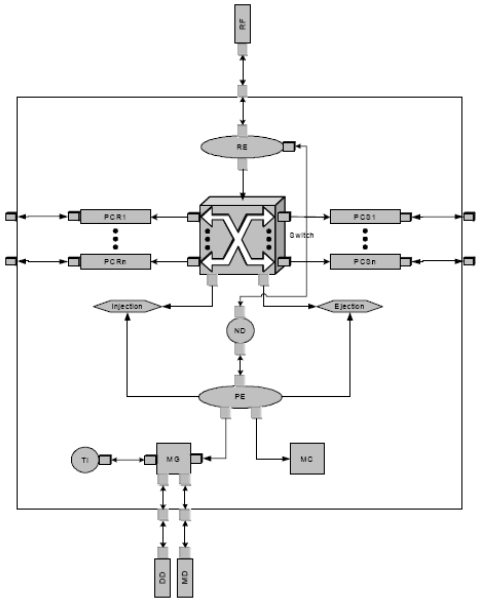


Fig. 4. Connection of different parts of a node in Xmulator

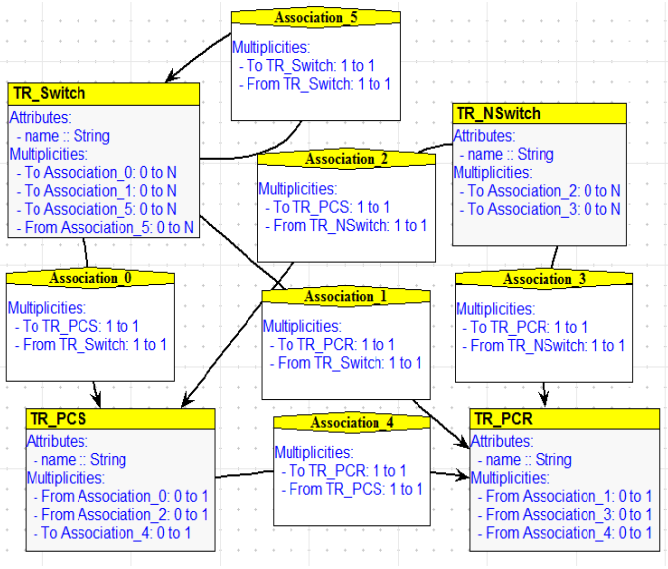


Fig. 5. Meta-Hypercube class diagram

3.2 Meta-SwitchInfo

This Meta-model describes important components of each switch which are shown in figure 4. These components are constant for each switch in the constructed topology in the Meta-Hypercube. The details of each construct are described here.

Switchinfo shows the simulation parameters of each simulation. Some of the main attributes of this construct are as follows:

- disableLogs: Tells to simulator that suppress logging activity. Logging is a very time consuming operation and is recommended for debugging purposes only.
- msgBlockSize: The batch number, it shows the number of messages that must be received by a node to calculate the statistical operation.
- maxSimulationEvents: When the number of events that are created in the simulation reaches this value the simulation will be stopped.
- maxSimulationEventsIncreaseRatio: When we want to run the simulation more than one time, the generation rate must increase in each run and as a result the maxSimulationEvents must be increased. This value shows the increased rate of maxSimulationEvents.
- nRuns: It shows the number of iterations for each simulation.
- initialGenRate: It shows the initial generation rate of each node.
- genRateStep: The generation rate in each run of the simulation is increased by this value.
- outputPath: It shows the output path for generating the simulation results.

Node shows some physical properties of each switch in the topology. Following are some of the main attributes of this construct.

- msgLen: It shows the length of the each message.
- vcN: It shows the number of virtual channels for each physical channel.
- vcSize: It shows the size of each virtual channel.
- pcDelay: It shows the delay of each physical channel.
- swDelay: The time that be consumed by the switch to connect a PCS to a PCR and vice versa.
- injVcN and ejVcN: They show the number of virtual channels for injection and ejection links respectively.
- injPcDelay and ejPcDelay: They show the communication delay of injection and ejection physical channels respectively.

Topology which shows the constructed topology of the user and can be a member of hypercube family (binary n-cube, necklace hypercube and stretched hypercube)
Destination Distribution which shows the distribution that will be used in each node to generate the destination of each message. In the current model we

have exponential and normal distributions.
Routing Element which shows the routing algorithm that used in each node to find the path to the destination. For each topology we implement two routing algorithms. One of them is deterministic and the other one is adaptive.
Injection which shows the input physical channel to the processing element of each node.
Ejection which shows the output physical channel from the processing element of each node.

The main construct in this Meta-Model is Switchinfo. While each Switchinfo must have one Destination Distribution, Routing Element, Node and Topology, we can have more than one Injection or Ejection physical channel for each Switchinfo. The Topology construct must contain one of the main hypercube topologies. The Routing Element must also have one of the instances of Deterministic or Adaptive constructs. As mentioned above for Destination Distribution we can have Normal or Exponential distributions. The class diagram of SwitchInfo Meta-Model is shown in figure 6.

Given the two Meta-Models described above, we can now generate in $ATOM^3$ a Meta-Specification, which, when loaded into the Meta-Level of $ATOM^3$, turns it into a new modeling environment for the modeled Hypercube interconnection network formalism. $ATOM^3$ also has a so-called Buttons formalism which creates a button for each construct of the formalisms. Using all these formalisms together we can construct different hypercube topologies and assign various properties to each node of the constructed topology.

4 Model Transformation

Model transformation is the process of taking as input a model conforming to a given Meta-Model and producing as output another model conforming to a given Meta-Model. As mentioned in [3] we use *XMulator* (XML + Simulator) to simulate the behaviour of interconnection networks. XMulator is a flexible simulation framework for performance evaluation of multicomputer networks implemented in *C#* language. It uses XML format for defining topologies, parameters, and outputs that is very flexible and user friendly and provides high level of interoperability. By enforcing strict boundaries between different components of a network (using object-oriented design), XMulator facilitates multi-factor experiments that independently explore network design issues. Using listener-based integration in company with multi-layered architecture enables XMulator to be used for various systems from interconnection networks to logic circuits. Developing new packages or new tools for existing packages as well as defining routing algorithms is accomplished easily due to its multi-level architecture and object-oriented design. Moreover, using log4net gives a great logging capability to XMulator which enables the user to trace in step by step fashion and monitor the movement of each flit in the network and to log all the parameters of each

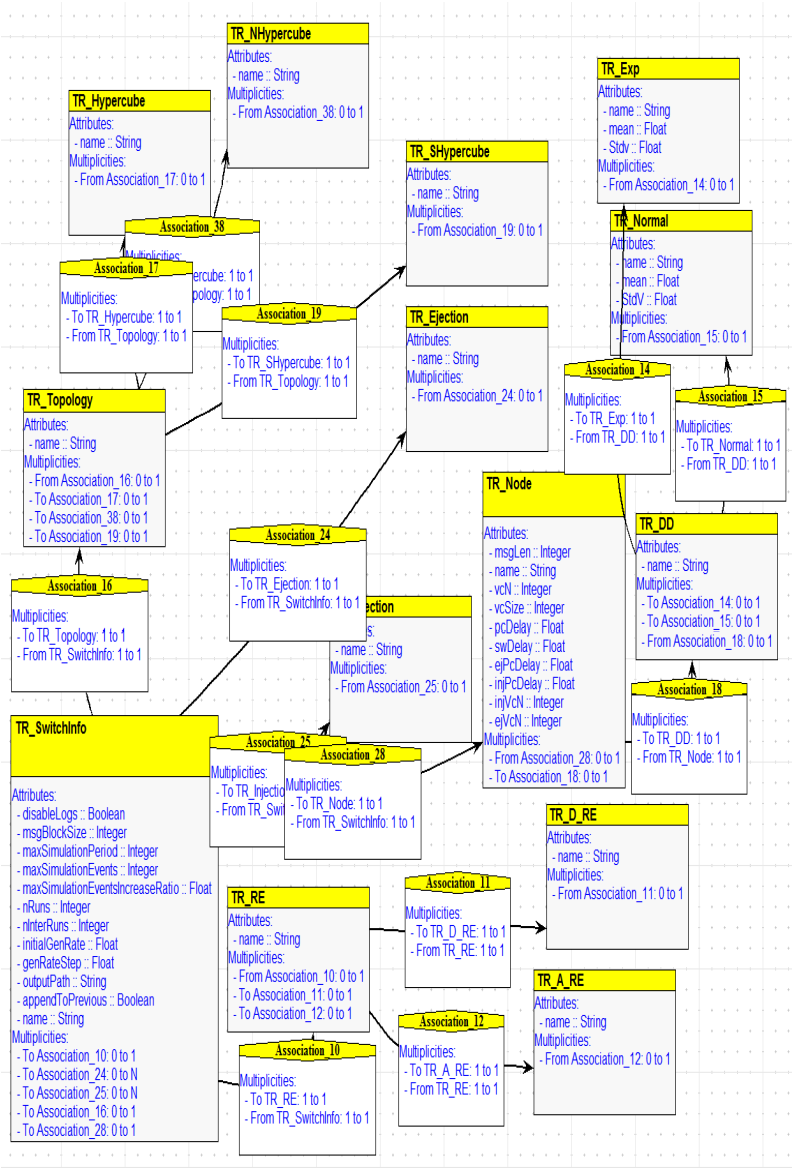


Fig. 6. Meta-SwitchInfo class diagram

component such as channels, routers, message generators, etc [8].

In order to generate the appropriate input for the Xmulator, we added one extra button, which on click applies the transformation to the XML file which is

the input of Xmulator. We have a so-called XML-Generator python code which will be executed on click of this button. The main purpose of this function is to generate the appropriate XML file. The first task of this function is to check the correctness of the user defined topology. As mentioned in section 3-2, in the SwitchInfo Meta-Model we determine the topology of the network. According to this pre-determined topology we can check the constructed network of the user. As a simple example, the number of main switches must be a power of two in hypercube family topologies. If the topology is necklace hypercube then we can count the number of necklace switches in respect to the number of main switches. The number of PCSs and PCRs must be equal and for each necklace switch we can only have two PCSs and PCRs.

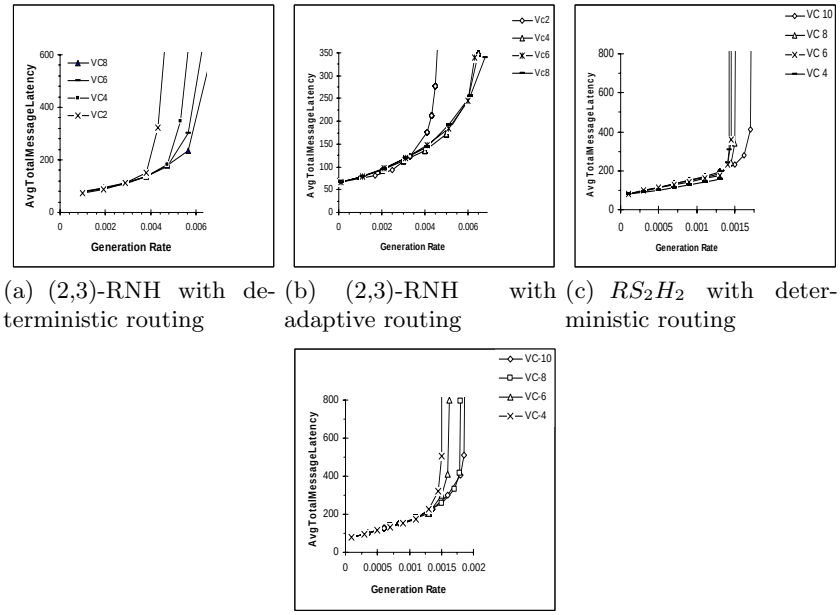
If the constructed topology passes all the tests then the XML-Generator code starts generating the XML code. It counts the number of instances of each construct and generates the appropriate XML tag for that construct. Some of the XML tags will be generated by parsing the attributes of constructs. The generated XML file will be feed to the Xmulator. The Xmulater then does the simulation and generates the results as an XML file. According to the generated results the user can change the topology of the network or different attributes of the simulation. Hence, we have both top-down and bottom-up phases to achieve the final results. In the following we demonstrate an example of generated results by Xmulator.

5 Results

In this section we show the effect of changing different parameters on the overall performance of the network. As an example in Figures 7-a and 7-b, we show the effect of the number of virtual channels for deterministic and adaptive routing algorithms on the overall performance of a (2,3)-RNH when the message length is assumed to be 64 flits. First, we set the number of virtual channels to 2 and then increased this value up to 10. As can be seen in the figures, increasing the number of virtual channels results in the better performance of the network. This is a rational effect because when we increase the number of virtual channels, there are some extra ways for the messages to continue their journey to the destination nodes, and hence the generation rate at which network saturation happens is increased. Another important point is the big difference between the saturation rates when 2 and 4 virtual channels are used, while the same difference for 6 or 8 virtual channels is small. Thus, the user can find that the first virtual channels play an important role on the performance of the system [4–6].

In figures 7-c and 7-d we consider the effect of the message length on the overall performance of a RS_2H_2 using the deterministic and adaptive routing algorithms respectively. To this end, we fix the number of virtual channels per physical channel to 6 channels in our SwitchInfo Meta-Model. As can be seen in the figures, increasing the message size increases the average message latency

and results in earlier saturation of the network [4–6].



(a) (2,3)-RNH with deterministic routing

(b) (2,3)-RNH with adaptive routing

(c) RS_2H_2 with deterministic routing

(d) RS_2H_2 with adaptive routing

Fig. 7. Illustrating Xmulator results

6 Conclusion

In this paper, we use $Atom^3$ to develop a Meta-Model for hypercube interconnection family. We have a Meta-Hypercube Meta-Model with which we can define the topology of the network (binary n-cube, necklace hypercube, stretched hypercube) and also a Meta-SwitchInfo Meta-Model which specifies the structure of each node in the constructed topology. We also use the Model-Transformation to generate an XML file from the constructed topology. This XML file is the input to the Xmulator which is a simulator for interconnection networks. Finally, the user can see the results and change the topology or different parameters of each node to achieve better results. As a future work, we can add the Meta-Models for different interconnection network topologies.

References

1. J. de Lara and H. Vangheluwe. Atom3: A tool for multi-formalism modelling and meta-modelling. In *European Conferences on Theory And Practice of Software Engineering ETAPS02, Fundamental Approaches to Software Engineering (FASE)*, pages 174–188. Springer-Verlag, 2002.
2. A. Levytskyy and E. J.H. Kerckhoffs. Meta-modelling of data flow processes with meta-modelling tool atom3. pages 729–733. ACM, 2005.
3. S. Meraji. An introduction to interconnection networks and xmulator. In *School of Computer Science, McGill University*, MOntreal, Quebec, Canada, 2008.
4. S. Meraji, A. Nayebi, and H. Sarbazi-Azad. Simulation-based performance evaluation of deterministic routing in necklace hypercubes. *aiccsa*, 0:343–350, 2007.
5. S. Meraji and H. Sarbazi-Azad. Simulation-based performance evaluation of stretched hypercubes. *Fina*, 0, 2008.
6. S. Meraji, H. Sarbazi-Azad, and A. Patooghy. Performance modelling of necklace hypercubes. *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8, 26-30 March 2007.
7. M. Monemizadeh and H. Sarbazi-Azad. The necklace-hypercube: a well scalable hypercube-based interconnection network for multiprocessors. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 729–733, New York, NY, USA, 2005. ACM.
8. A. Nayebi, S. Meraji, A. Shamaei, and H. Sarbazi-Azad. Xmulator: A listener-based integrated simulation platform for interconnection networks. *Modelling and Simulation, 2007. AMS '07. First Asia International Conference on*, pages 128–132, 27-30 March 2007.
9. P. Shareghi and H. Sarbazi-Azad. Topological properties of stretched graphs. In *AICCSA '06: Proceedings of the IEEE International Conference on Computer Systems and Applications*, pages 647–650, Washington, DC, USA, 2006. IEEE Computer Society.
10. www.devhood.com/tutorials/tutorialdetails.aspxtutorialid=698.
11. www.moncs.cs.mcgill.ca/people/mosterman/campam/cca01/index.html.
12. www.wikipedia.com.