Hindawi Publishing Corporation
VLSI Design
Volume 2007, Article ID 90941, 15 pages
doi:10.1155/2007/90941

Research Article

Network Delays and Link Capacities in Application-Specific Wormhole NoCs

Zvika Guz, Isask'har Walter, Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny

Electrical Engineering Department, Technion–Israel Institute of Technology, Technion city, Haifa 32000, Israel

Received 15 November 2006; Accepted 6 February 2007

Recommended by Maurizio Palesi

Network-on-chip- (NoC-) based application-specific systems on chip, where information traffic is heterogeneous and delay requirements may largely vary, require individual capacity assignment for each link in the NoC. This is in contrast to the standard approach of on- and off-chip interconnection networks which employ uniform-capacity links. Therefore, the allocation of link capacities is an essential step in the automated design process of NoC-based systems. The algorithm should minimize the communication resource costs under Quality-of-Service timing constraints. This paper presents a novel analytical delay model for virtual channeled wormhole networks with nonuniform links and applies the analysis in devising an efficient capacity allocation algorithm which assigns link capacities such that packet delay requirements for each flow are satisfied.

1. INTRODUCTION

Network-on-Chip (NoC) is a novel communication paradigm for MultiProcessor Systems-on-Chip (MPSoCs). NoCs provide enhanced performance and scalability, in comparison with previous communication architectures (e.g., dedicated point-to-point signal wires, shared buses, or segmented buses with bridges) [1, 2]. The advantages of NoC are achieved thanks to efficient sharing of wires and a high level of parallelism. Many MPSoCs use specialized application-specific computational blocks, and they require heterogeneous, application-specific communication fabrics. These systems operate under typical precharacterized information-flow patterns, which can often be classified into a few "use-cases" [3] where throughput and delay requirements are specified for data-flows from sources to destinations in the system.

Application-specific NoC generation has been addressed previously [4–10]. The NoC can be customized for a particular application-specific MPSoC through an automatic network design phase [4, 10–14]. Most previous research on NoC customization deals with the selection of network topology, module placement and routing scheme to accommodate the expected application-specific data traffic patterns, assuming that all links in the NoC are identical (e.g., [15–18]). In this paper, the application-specific customization is extended to include allocation of bandwidth for each link in the network according to expected traffic flows and their delay requirements. It should be noted that the problem of capacity allocation is independent of the aforementioned NoC challenges and can be used in conjunction with other optimization procedures.

The use of links with uniform capacities, typical of multicomputer interconnect, is unsuitable for application-specific NoC. In multicomputer systems, there is often little or no knowledge about the dynamic bandwidth and timing requirements of the tasks being executed, or about their assignment to a physical processing unit. In addition, in an off-chip network dynamic routing has an acceptable cost and loads can be equally distributed among multiple paths.

SoC traffic is usually more heterogeneous, in terms of both bandwidth and delay requirements. SoC is also subjected to detailed specifications describing the traffic and timing restrictions of each communicating pair, hence in many cases all communication characteristics are known at design time. At the same time, due to more restrictive cost considerations, solutions such as dynamic routing and load distribution are less appealing and instead fixed shortest-path routing is typically preferred in order to minimize router and network interface logic and communication power dissipation [4, 10, 12, 13, 15, 19]. As a result, different links in the NoC must carry significantly different data

rates and therefore different capacities should be allocated to them. In addition, different latency requirements for different data flows should affect the allocation of extra capacity to some critical links.

The goal of capacity allocation is to minimize the network *cost* (in terms of *link area* and *power*) while maintaining acceptable *packet delays* for the specific system communication demands [4]. This step in the design process is similar to timing closure in traditional chip design, where critical path drivers are often upsized and noncritical drivers are downsized for saving power. However, in an NoC, critical timing paths cannot be uniquely matched to dedicated signal wires, since multiple messages share the network links. In an NoC, timing closure must be achieved by adequate bandwidth allocation to each of the network links: insufficient allocation will not meet performance requirements, while lavish allocation will result in excessive power consumption and area. During the design process, the network architect can control the physical link capacity by setting the number of parallel wires it is composed of, or by tuning its clock frequency. If the network is implemented asynchronously, repeaters can be placed to regulate the throughput of the link.

Wormhole routing [20] is an increasingly common interconnect scheme for NoC as it minimizes communication latencies, requires small buffer space and is relatively simple to implement. However, performance evaluation and customization process of wormhole-based NoCs heavily rely on simulations as no existing analysis accounts for the combination of heterogeneous traffic patterns and virtual channels [21]. Unfortunately, these are both fundamental characteristics of NoC interconnect. The use of simulations makes the task of searching for efficient capacity allocation computationally extensive and does not scale well with the size of the problem. On the other hand, a detailed exact analytical solution of a complex NoC is intractable and simplistic approximations may lead to inaccurate results.

Two contributions are described in this paper. First, a novel delay analysis for wormhole-based NoCs is presented. The model approximates the network behavior under a wide range of loads. Given any system (in terms of topology, routing, and link capacities) and its communication demands (in terms of packets length and generation rate), the analysis estimates the delay experienced by every source-destination pair. To the best of our knowledge, this is the first analysis of a wormhole network with nonuniform link capacities. Second, an algorithm that applies the delay analysis for efficiently allocating capacities to network links is described. Simulation is only used for final verification and fine tuning of the system. This methodology considerably decreases the total NoC cost and significantly improves the speed of the customization process.

The rest of this paper is organized as follows: in Section 2, the capacity allocation problem is formalized. In Section 3, we present and evaluate a delay model for heterogeneous wormhole networks with multiple virtual channels and different link capacities. The capacity allocation algorithm is discussed in Section 4, while Section 5 presents design examples and simulation results. Finally, Section 6 concludes the paper. In the appendix, we describe QNoC, a QoS-oriented architecture for on-chip networks that was used for evaluation of our analysis and capacity allocation technique.

Given:
F
$\forall f \in F$: m^f, λ^f, T^f_{REQ}
$\forall$ *link* j, assign link capacity (C_j) s.t.:
$\forall f \in F$: $T^f \leq T^f_{\text{REQ}}$
$\sum C_j$ is minimal

Figure 1: Capacity assignment minimization problem.

2. THE LINK CAPACITY ALLOCATION PROBLEM

If all links in an application-specific NoC are given the same capacity, some of them might be much larger than needed, consuming unnecessary power and area. Using an analogy to timing closure in classical VLSI design, this resembles instantiating the strong drivers that are needed in the critical path, throughout the entire chip.

Alternatively, an efficient capacity allocation assigns just enough capacity to each of the network links such that all flows in the system meet their delay requirements while the total capacity invested in the network is minimized.

In order to formalize the minimization problem, we introduce the following notation.

F = *The set of all flows, from every source module.* $1 \leq s \leq N$ *to every destination module* $1 \leq d \leq N$.
f^i = *A flow from the set* F.
m^i = *The mean packet length (number of flits) of flow.* $f^i \in F$ *[flits]*.
λ^i = *Average packet generation rate of flow* $f^i \in F$. *[packets/second]*.
T^f_{REQ} = *The required mean packet delivery time for flow f.*
C_j = *Capacity of link j [bits/second]*.

The capacity assignment minimization problem can be formalized as in Figure 1.

Naïve allocations, based only on flow bandwidth requirements, do not yield good solutions. For example, if the capacity assigned to each link is equal to the average data rate of the flows it carries, any temporary deviation from the average will cause network congestion and unacceptable delays. Therefore, the links must be over.3provisioned with bandwidth and should be designed to operate at a fairly low average utilization rate. If utilization rates are too high, network delays will grow and the latency of some flows will be too long. Assignment of link capacities such that the utilization rates of all links are equal may be a reasonable allocation heuristic for homogeneous systems, in which all flows have similar characteristics. However, such an allocation might lead to extreme waste of resources in a heterogeneous scenario. For example (Figure 2), consider a flow f^1 which injects 1 Gb/s into some network link (00→01),

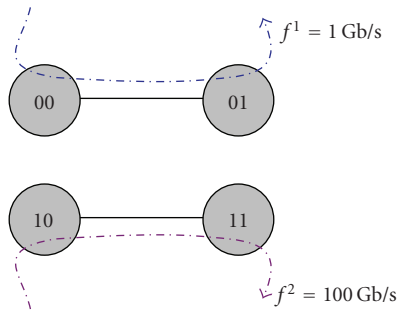


Figure 2: Simple example of a system in which equal utilization capacity assignment is wasteful.

but has a critical latency, such that a bandwidth of 10 Gb/s is required in that link (10% utilization). Assume that another flow f^2 injects 100 Gb/s into another link, and its latency requirement permits 200 Gb/s (50% utilization) in that link (10→11). Enforcing equal link utilization throughout the network will require 10% utilization everywhere and will unnecessarily increase the second link capacity to 1000 Gb/s.

Intuitively, the capacity of some links can be reduced based on latency requirements so that delivery times are stretched to the maximum allowed. Therefore, any efficient capacity allocation algorithm must be latency aware. Hence, an efficient model for evaluating the delays of packets through the network is required. The static delay model derived in Section 3 will be used to direct our capacity allocation algorithm.

3. WORMHOLE DELAY MODEL

In a wormhole network, packets are divided into a sequence of flits which are transmitted over physical links one by one in pipeline fashion. A hop-to-hop credit mechanism assures that a flit is transmitted only when the receiving port has free space in its input buffer. Performance of such networks has been studied extensively in the past. While several analytical studies have evaluated the mean latency of packets in wormhole networks (e.g., [22–29]), no previous work analyzes wormhole networks with heterogeneous link capacities, which is a fundamental characteristic of NoC that makes it cost effective [4]. Moreover, existing models evaluate the mean latency over all flows in the system. Since each SoC flow must meet its delay requirement, a perflow analysis is necessary. In addition, no previous work analyzes networks with both virtual-channels and nonuniform source-destination communication patterns. For example, [22] presents an analytical model of a wormhole network with an arbitrary number of virtual channels per link. Though comprehensive, it assumes that messages destinations are uniformly distributed. In [23], the model does reflect nonuniform traffic but does not address networks with multiple virtual channels.

Consequently, a new wormhole delay analysis is called for, one that captures the above fundamental characteristic of SoC. In the following subsections, we present our wormhole analysis. Section 3.1 introduces the network model and notations, Section 3.2 presents the analysis and Section 3.3 evaluates the analysis.

3.1. The network model

The time to deliver a packet between a specific source-destination pair is composed of the source queuing time and the time it take the packet to traverse the network (hereafter referred to as *network time*). In a wormhole network, network time is composed of two components [29]: the time it takes the head flit to reach the destination module (*path acquisition time*) and the time it takes the rest of the packet to exit the network (*transfer time*); path acquisition time is affected by the complex interaction among different flows in the system and transfer time is affected by other flows sharing the same links (link capacity is time multiplexed among all virtual channels sharing the link).

Since NoC delay/cost tradeoffs are different from those of off-chip networks and since performance is a key issue, we assume that NoCs will be designed to operate under a relatively moderate load. Consequently, for the sake of simplicity and computational efficiency, our analysis addresses low to medium loads and does not attempt to achieve high accuracy under very high utilizations.

Previous research [21] has showed that adding virtual channels increases the maximal network throughput. We assume that physical links are split into an adequate number of virtual channels. In particular, we assume that a head flit can acquire a VC instantaneously on every link it traverses.

Our analysis focuses on the transfer of long packets, that is, packets which are composed of a number of flits significantly larger than the number of buffers along their path. From the simulations presented in [4], it is clear that such packets (termed the Block Transfer class of service) are the ones that place the most stringent demand on NoC resources and hence dominate the bandwidth requirements. A delay analysis capturing the latencies of short packet in wormhole NoCs which is valuable for the system architect is left for future work.

We consider a wormhole deadlock-free fixed routing network that is composed of N routers connected by unidirectional links. The packets that constitute the traffic of each source-destination pair identify a *flow*.

Our model uses the following assumptions.

(1) Each flow generates fixed length packets using a Poisson process (bursty traffic can be modeled by using artificially larger packet size).
(2) Sources have infinite queues, and sinks immediately consume flits arriving at their destination.
(3) Routers have a single flit input queue per virtual channel.
(4) The propagation delay of flits through links and routers is negligible.
(5) Back pressure credit signal is instantaneous.

We will use the following additional notation to characterize the network:

l = *flit size [bits]*,
π^i = *the set of links composing the fixed path of flow* f^i,
π^i_j = *the set of links that are subsequent to link* j *on, flow* i*'s path (a suffix of the path* π^i*)*.

The following notation is used in order to analyze the packets delay:

T^i = *the mean packet delivery time of packets of flow* f^i (*the average time elapsed since a packet is created until its last flit arrives at its destination*),
Q^i = *mean source queuing time of packet of flow* f^i (*the average time elapsed since the packet is created until it enters the network*),
T^i_{network} = *the mean network time of packets of flow* f^i (*the average time elapsed since the packet is inserted into the network until its last flit is received by the destination module*),
t^i_j = *the mean time to deliver a flit of flow i over link* j *(waiting for transmission and transmission times)*,
Λ^i_j = *the total flit injection rate of all flows sharing link* j, except flow f^i *[flits/second]*.

3.2. Wormhole delay analysis

We approximate the source queuing time using the M/D/1 model [30]:

$$Q^i = \frac{1}{2 \cdot (1/T^i_{\text{network}} - \lambda^i)} - \frac{T^i_{\text{network}}}{2} \quad . \tag{1}$$

Clearly, when a flow does not share any of its links with other flows, (1) is the exact mean queuing time, since the time required to deliver a packet through the network (T^i_{network}) is deterministic. When a packet might be interleaved with other flits within the network, the service time is not deterministic anymore, and the assumptions of the M/D/1 model do not hold. However, thorough simulation (presented in Sections 3.3 and 5) show that (1) is a good approximation for the queuing time even for flows that are frequently multiplexed.

The network time of a packet in a wormhole network resembles a pipeline traversal. When the number of parts composing the packet is considerably larger than the number of pipeline stages, the latency (the time it takes the first bits to exit the pipe) is insignificant compared with the total time, which in this case is mostly affected by the pipeline's throughput. Since packets are assumed to be considerably longer than the buffers along their path, and since each head flit is assumed to instantaneously acquire a virtual channel on every link it arrives at, we ignore path acquisition time and approximate the transmission time only.

As in a classic pipeline, the transfer time is dominated by the stage with the smallest service rate. Since flits of different flows are interleaved on links, the approximation of t^i_j should account for the transmission time of other flits on the same link. We use a modification of the basic M/M/1 modeling [30] as an approximation of the flit interleaving delay,

$$t^i_j = \frac{1}{1/l \cdot C_j - \Lambda^i_j} \quad , \tag{2}$$

where Λ^i_j is the bandwidth consumed by all flows other than flow i on link j. Formally,

$$\Lambda^i_j = \sum_{f \mid j \in \pi^f \wedge f \neq i} \lambda^f \cdot m^f \quad . \tag{3}$$

Equation (2) models the mean interleaving delay experienced by flits of flow i on link j as a simple queue, without accounting for the bandwidth consumed by packets of flow i itself. This modification is based on the observation that flits of a flow are interleaved on a physical link due to the delivery of packets that belong to other flows only.

By substituting (3) into (2) we get

$$t^i_j = \frac{l}{C_j - l \cdot \sum_{f \mid j \in \pi^f \wedge f \neq i} \lambda^f \cdot m^f} \quad . \tag{4}$$

The total network time, which is dominated by the hop with the longest delay, can then be written as

$$T^i_{\text{network}} \simeq m^i \cdot \max\left(t^i_j \mid j \in \pi^i\right) \quad . \tag{5}$$

The above approximation does not capture interlink dependencies and is generally too optimistic for medium and high loads. Wormhole link loads affect each other mainly by the back-pressure mechanism: a flit must wait for the arrival of a credit for its virtual channel from the following link. Therefore, flit delivery time over a link (t^i_j) is affected by the delivery time in subsequent (downstream) links on the flow's path. To reflect the effect of flit delay on other (upstream) links, we replace (4) by the following expression which accounts for these interlink dependencies. Our simulations (Sections 3.3 and 5) show that this simple expression successfully estimates the resulting link delay:

$$\widetilde{t}^i_j = t^i_j + \sum_{k \mid k \in \pi^i_j} \frac{l \cdot \Lambda^i_k}{C_k} \cdot \frac{t^i_k}{\text{dist}^i(j,k)} \quad , \tag{6}$$

where $\text{dist}^i(k, j)$ is the distance (measured in number of hops) between link j and k on the path of flow i. Formally written as

$$\text{dist}^i(k,j) = \left| \frac{\pi^i_j}{\pi^i_k} \right| \quad . \tag{7}$$

Equation (6) approximates the delay experienced by flits of flow i on link j by adding to the basic flit delay (t^i_j) a term that takes into account the cumulative effect of the delays of subsequent links along the path. Each subsequent link delay is weighted by two factors: the links' distance from link $j(\text{dist}^i(k, j))$ and the factor by which the link is utilized by flows other than flow i itself ($l \cdot \Lambda^i_k / C_k$).

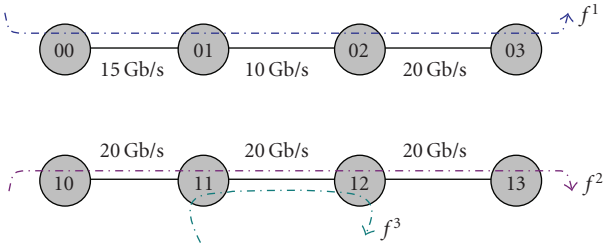


Figure 3: NoC simple example.

As explained above, the mean total network time of each flow is calculated using the longest interleaving delay on its path. Therefore, (5) is replaced by

$$T^i_{\text{network}} \simeq m^i \cdot \max\left(\tilde{t}^i_j \mid j \in \pi^i\right) . \tag{8}$$

Finally, the total packet delivery time is equal to

$$T^i = Q^i + T^i_{\text{network}} . \tag{9}$$

3.3. Model characterization

In this section, we evaluate the accuracy of the wormhole network delay model by comparison with detailed network simulation of example systems. First, Section 3.3.1 presents a simple example to demonstrate the analysis. Then, Section 3.3.2 considers a classic scenario of 4×4 homogeneous systems. We take QNoC (described in the appendix) as an example of an NoC architecture. The OPNET simulator was extended to include a full NoC simulation based on the QNoC architecture, including all complex dependencies between different flows in wormhole networks (due to particular virtual channel assignment schemes, finite router queues, etc.).

3.3.1. Simple example

To demonstrate the analysis we present a simple and intuitive example, depicted in Figure 3.

The example system consists of three flows: flow f^1 does not share any of its links with any other flow. Its path (from source 00 to destination 03) is composed of links with different capacities as shown. Flow f^2, running from source 10 to destination 13, shares one of its links (11→12) with flow f^3 (running from source 11 to destination 12). Consequently, while the service time for flow f^1 is constant, the transmission times of flow f^2 and f^3 are affected by the contention on their shared link. All of the links on their path have the same capacity of 20 Gb/s.

Figure 4 presents the normalized average packet delay of the three flows as a function of varying system load, created by changing the generation rate of each flow. As expected, the network delay (path acquisition time plus transfer time) of flow f^1 (Figure 4(a)) is unaffected by the varying utilization, because its packets are never interleaved on a link, and can always use the entire capacity of the links on its path. However, when utilization is high, the packet delay increases due to source queuing latency. Figure 4(a) also shows that the analytical terms for both the network delay (8) and queuing time (1) provide good prediction of the simulation results, as the delivery time is dictated by the path's lowest capacity link and the source queue follows M/D/1 assumptions.

Figures 4(b) and 4(c) show the delays of flows f^2 and f^3, respectively. Unlike flow f^1, the network delays of these flows grow as link utilization increases, since the available residual capacity on the common link decreases. As expected, both flows also experience source queuing delay, which becomes significant at high utilization. For all three flows, the analysis closely predicts the network and queuing delays measured by simulations.

3.3.2. Homogeneous all-to-all example

To demonstrate the accuracy of our analytical model we now consider a homogeneous system in which every module injects the same amount of bandwidth into the network and packet destinations are chosen randomly with uniform distribution. While this is not typical of SoC, this scenario is very often used to evaluate wormhole networks [21, 22, 25–29]. We compare the analytical delay model with simulation results for a varying utilization factor, by using a wide range of uniform capacity allocation vectors.

Our network, illustrated in Figure 5, is comprised of a regular four by four two dimensional mesh with symmetric XY routing [4]. All links have identical capacities; packets of each flow are generated by a Poisson process, with a mean rate $\lambda = 1/0.00048$ [packets/s]; packets consist of 500 flits, each flit is 16 bit long.

As can be seen in Figure 6, although all flows inject the same bandwidth, the different distance and different aggregated load of links along their paths result in a large variation in packet delays. Assuming that all flows have identical requirements, the mean packet delivery time of some flows is much lower than needed. This slack can be trimmed by a more efficient link capacity scheme.

Figure 7 compares the mean end-to-end packet delay predicted by the analytical model with simulation results, as a function of the utilization level of the most utilized link (i.e., for a wide range of uniform capacity allocations). The analytical model closely predicts the simulation results for a wide range of loads, way beyond the loads that are expected in a practical SoC (the mean absolute error reaches 8% when utilization is over 90%). The mean absolute error is presented in Figure 8.

Figure 9 zooms in on two specific flows, depicting the simulations and analysis end-to-end packet delays of flows 00→10 and 00→33, respectively. As can be seen from the figure, flow 00→10, which runs on a single link shared by only two other flows (according to the symmetric XY routing), suffers a moderate increase in its end-to-end delay as system load increases. On the other hand, the delay of flow 00→33 is much more sensitive to the system load, experiencing significant degradation in its delay as the load increases. This

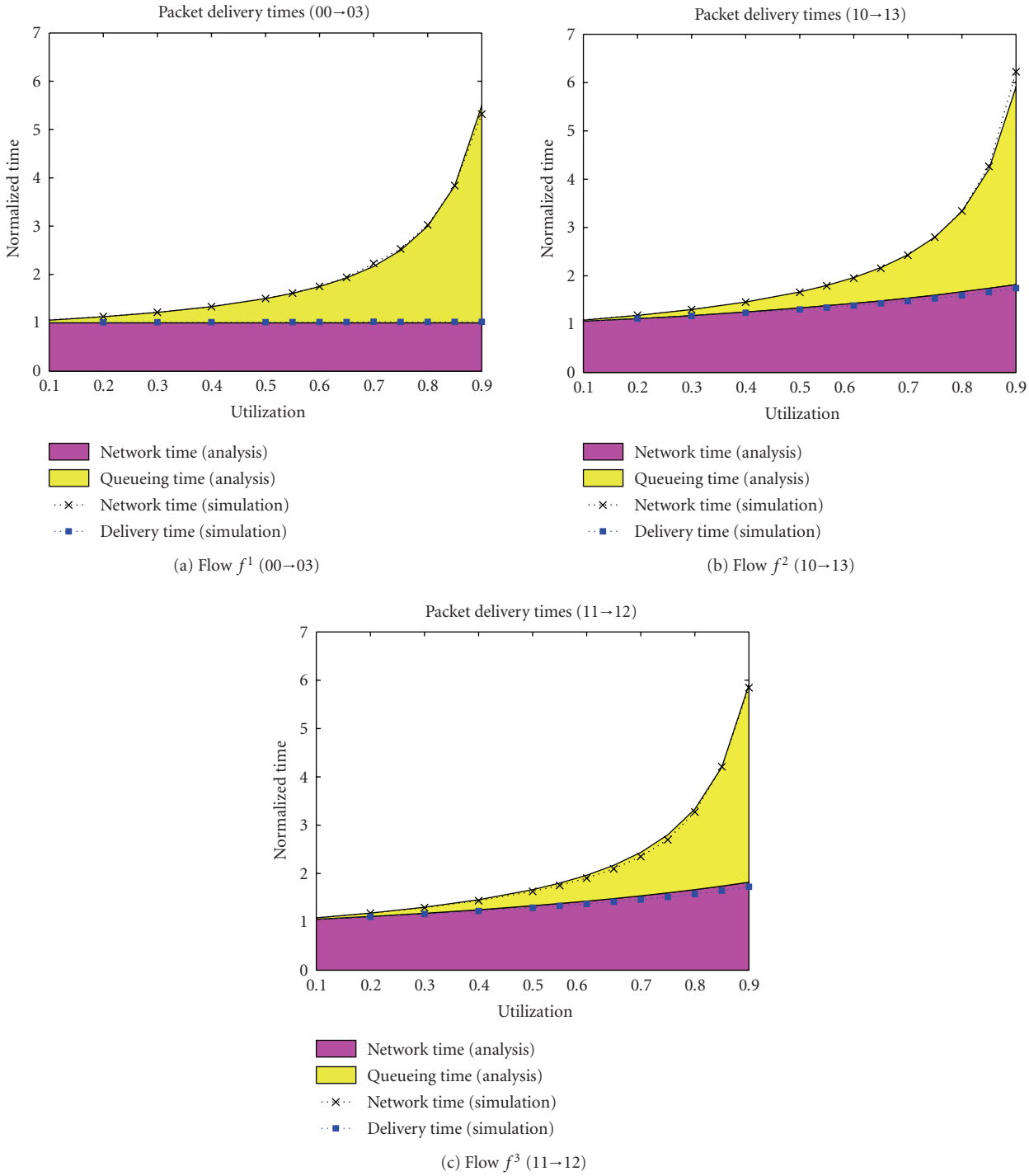


FIGURE 4: Simulation and analytical delay decomposition of flows f^1, f^2, and f^3. Horizontal axis marks the maximum link utilization on the flow's path; vertical axis is the delay normalized by delivery time in an unloaded network.

flow, running on multiple links, among them few of the most loaded links in the system, suffers from many contentions with other flows along its path. The higher loads result in many collisions, and hence significant increase in its total delay. Although these flows have different characteristics, the analysis closely predicts the resulting delays measured in simulations.

4. CAPACITY ALLOCATION

Similar to the TILOS sizing algorithm [31], which uses static timing analysis to optimize transistor sizes, our algorithm minimizes the resources allocated to network paths with relaxed timing requirement, and assigns extra network bandwidth to critical paths that need to be optimized.

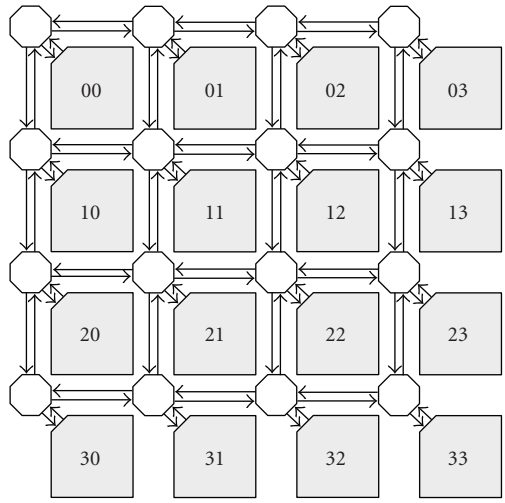


Figure 5: The homogeneous NoC example: a four-by-four mesh with 48 unidirectional interrouter links.

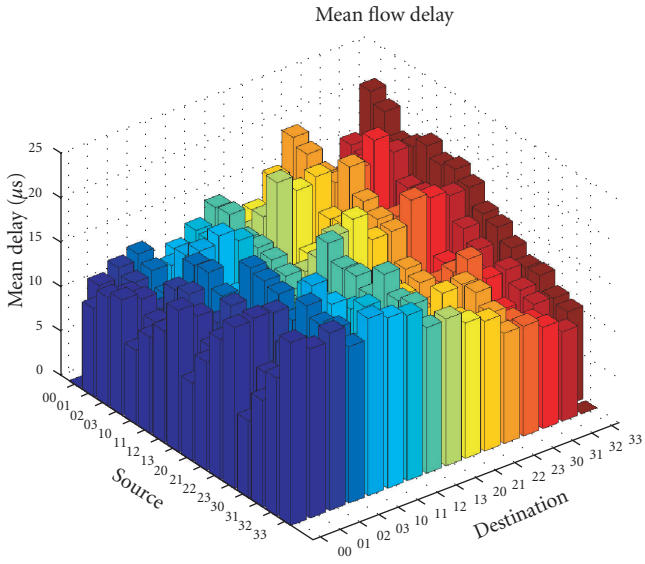


Figure 6: Simulated mean delay in the homogeneous system. Base plane axes mark the source and destinations module addresses as defined in Figure 5, respectively, and the vertical axis is the mean packet delivery time (longest delay is 2.5× longer than the shortest one).

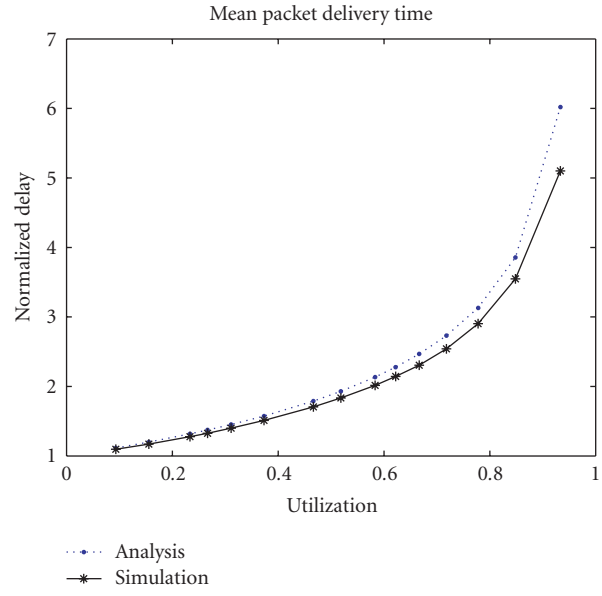


Figure 7: Model estimation and simulation results for the homogeneous system. Horizontal axis is the utilization of the most utilized link and the vertical axis represents the delay normalized by the delay in a zero-loaded system (i.e., where no other flows exist).

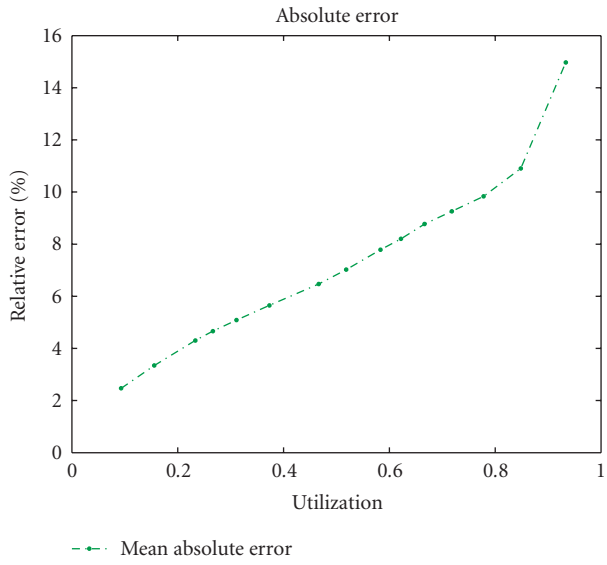


Figure 8: Mean absolute error in the homogeneous system.

However, unlike classical VLSI circuits in which each wire has a unique delay constraint, NoC links are shared by data flows with different timing requirements which must all be met.

Using the model presented in Section 3, all network delays can be computed directly, without resorting to simulation, for any set of network parameters. This computation can be performed in the inner loop of an optimization algorithm which searches for a low-cost capacity allocation for the NoC links such that all delay requirements are satisfied.

If the SoC is to handle multiple use-cases [3], the capacity allocation process should be performed for each case individually. If link capacities can be dynamically adjusted, the appropriate bandwidth should be selected at run-time. Else, the designer can statically set each link's capacity to the maximal bandwidth it requires in all use cases. This will assure that all timing requirements are met in all operating scenarios.

Our capacity allocation algorithm, specified in Algorithm 1, takes a greedy approach. The initialization phase

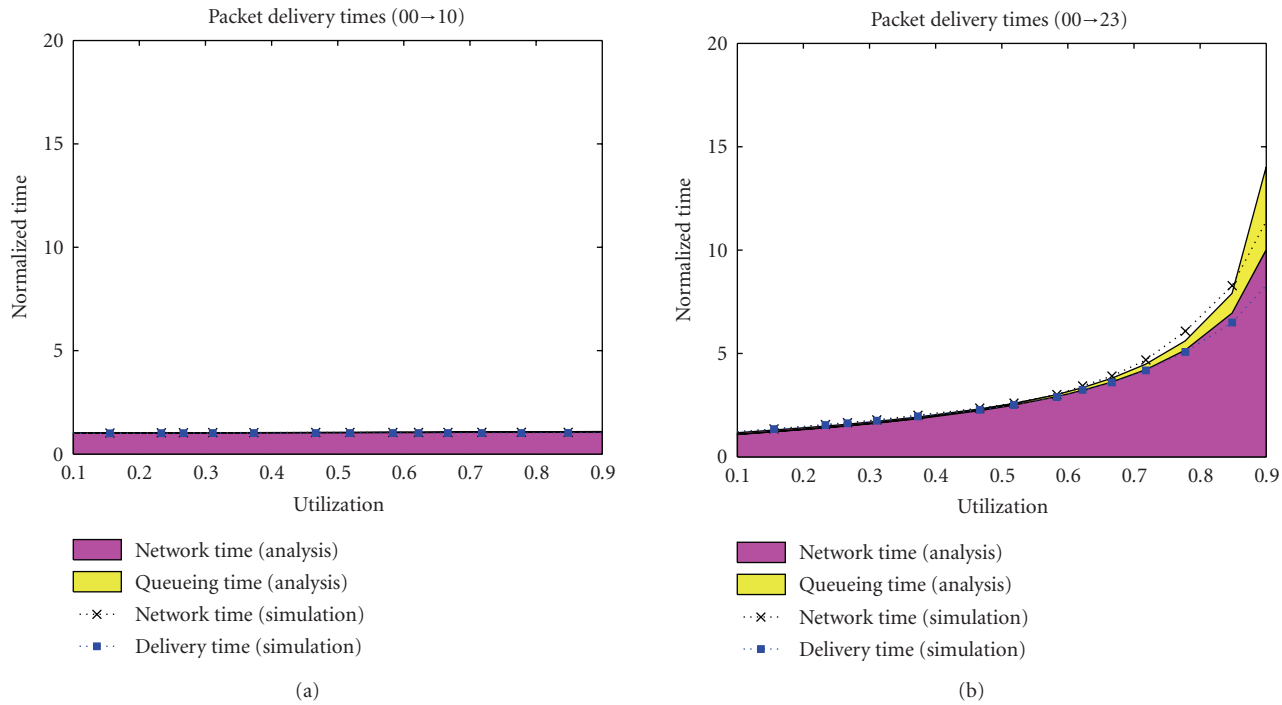


Figure 9: Model estimation and simulation results for the end-to-end delay of flow (a) 00→10 and (b) 00→33 in the homogenous system. The horizontal axis shows utilization of the most utilized link in the system and the vertical axis represents the delay normalized by the delay in an unloaded system.

```
    /* assign initial capacities */
(1) foreach link e:
(2)   C_e ← Σ_{f∈F:e∈π^f} λ^f · m^f · l
(3) end foreach
(4) foreach flow f ∈ F:
    /* evaluate current packet delivery time */
(5)   T^f ← Delay_Model (C, f)
(6)   while (T^f > T^f_REQ)
        /* look for most sensitive link */
(7)     foreach e ∈ π^f:
(8)       ∀ j ≠ e: C̃_j = C_j
(9)       C̃_e = C_e + δ
(10)      T̃^f_e ← Delay_Model (C̃, f)
(11)    end foreach
      /* get most sensitive link */
(12)  e' = argmin_e {T̃^f_e}
      /* increase its capacity */
(13)  C_e' = C_e' + δ
(14)  end while
(15) end foreach
```

Algorithm 1: Capacity allocation algorithm.

allocates a minimal preliminary capacity to each link in the network (lines 1–3). The main loop (lines 4–15) analyses each source-destination flow separately. It first uses the delay model (Section 3) to approximate the flow's delay given the current capacity allocation vector (line 5). If the delay is longer than required (line 6), the algorithm allocates a small, predefined amount (δ) of extra capacity to the flow's path. It first temporarily increases the capacity of all links along the path separately (lines 7–11) and approximates the resulting packet delay. It then searches for the link with the largest gain from bandwidth addition, that is, the link for which adding capacity results in the shortest delay overall (line 12). The extra capacity is added only to that link (line 13). When the algorithm terminates, all flows meet their required delay. Since the algorithm handles flow-by-flow, considering for every flow only the links along its path for optimization, and since every iteration on the flow's links reduces the delay of the weakest link along that flow's path, the algorithm is bound to terminate with locally minimal allocation as its output. Investigation of detailed convergence properties and complexity analysis are left as future work.

In practice, the analytical model does not capture all complex dependencies and effects in a virtual channeled wormhole network. As a result, the mean delay of a few flows may be underestimated and the capacity of some links might be too small. Therefore, the resulting assignment is verified using network simulation, and some extra capacity is added to these flows' path.

5. DESIGN EXAMPLES

Two examples are presented, to demonstrate the delay model and the benefit of using the capacity allocation algorithm. Both examples exhibit nonuniform communication patterns, which are more realistic in SoC than the homogeneous

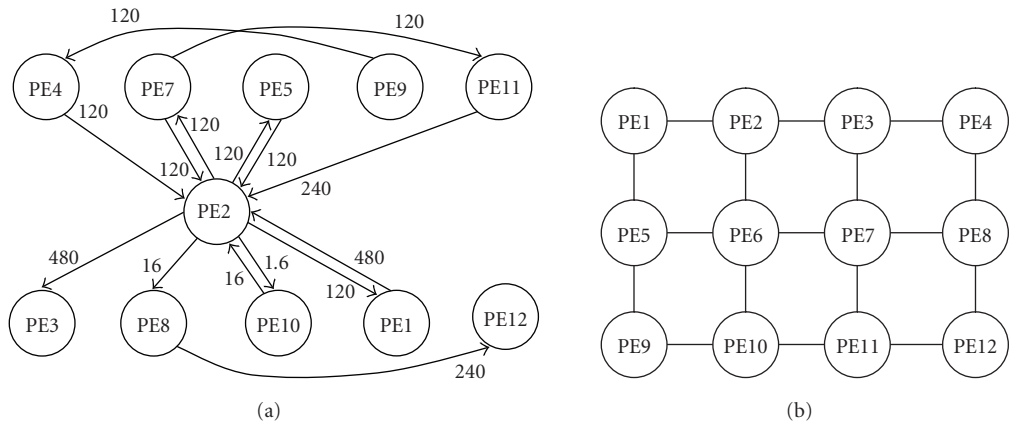


Figure 10: (a) Task graph and (b) placement for the DVD decoder system.

example used in Section 3. Modules communicate with only a subset of all possible destinations and different flows have different bandwidth and different delay requirements. In the first example, some modules send and receive packets from a single module (many-to-one and one-to-many patterns), emulating a SoC in which data is generated and destined at specific modules (e.g., SoCs in which there is a main CPU or a single cache memory, or SoCs that use an off-chip DRAM). The second example considers a video application, in which the main data path is pipelined, that is, data blocks are forwarded from module to module.

For each example, we apply the suggested capacity allocation algorithm and present its benefit over uniform link capacities assignment. We have implemented a tool that automatically assigns link capacities given the network topology, routing scheme, communication demands, and QoS requirements. This tool, which uses the aforementioned delay model and capacity allocation algorithm (described in Section 4), is to be used by the chip interconnect designer to minimize network resources. For simulation, we have used the OPNET-based NoC simulator described in Section 3.3.

Table 1: DVD decoder system.

Flow	Interarrival time (μs)	Packet length (flits)	Required delay (μs)
00→01	16.67	500	5
01→12	66.67	500	10
01→10	66.67	500	10
01→00	66.67	500	5
01→02	16.67	500	10
01→21	5 000.00	500	15
01→13	500.00	500	10
03→01	66.67	500	10
10→01	66.67	500	10
12→01	66.67	500	10
12→22	66.67	500	5
20→03	66.67	500	10
21→01	500.00	500	15
22→01	33.30	500	10
23→13	33.30	500	10

5.1. DVD video decoder example

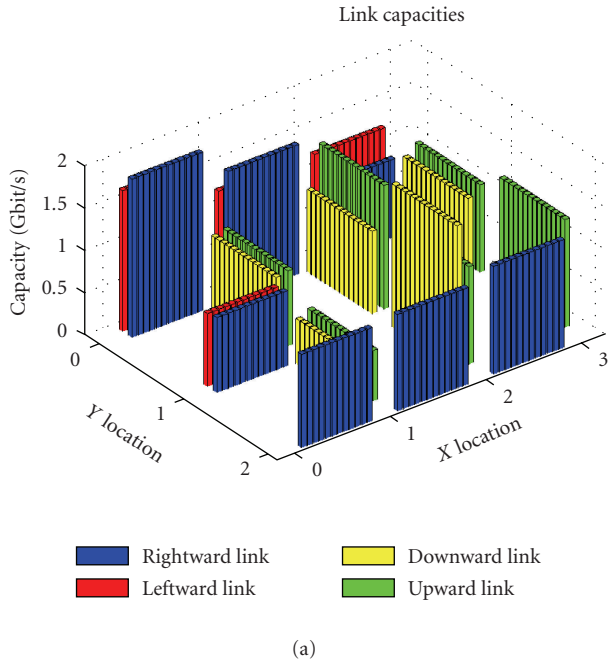
The first heterogeneous network comprises a regular three by four, two dimensional mesh with XY routing, implementing a DVD video decoder. Table 1 describes the different flow characteristics and delay requirements and Figure 10 presents the task graph and module placement.

We have compared the total capacity assigned by the capacity allocation algorithm with a uniform assignment that meets the same requirements. Figure 11 presents the capacities assigned to each link by the algorithm. While the uniform assignment requires a total of 41.8 Gb/s, the algorithm used only 25.2 Gb/s, achieving a 40% reduction of resources. Figure 12 shows that all flows meet their requirements and that the static analytical delay model adequately predicts the simulated latencies.

Figure 13 compares the packets delay slacks (i.e., the difference between the actual packet delay and the delay requirement) in the two allocations. Since in the uniform allocation all links are assigned identical capacity, the mean packet delays of some of the flows are much lower than required. On the other hand, the capacity allocation algorithm adjusts individual link capacities, thus reducing the slack and improving efficiency. Note that some flows may benefit from links with high capacity on their path that is needed to satisfy the requirements of other flows sharing the link. As a result slacks are still possible even in an optimal allocation.

5.2. Video processing application

The second example is a video processing application, based on the video object plane decoder (VOPD) system presented in [5]. The system, illustrated in Figure 14, has 15 data flows. The commutation characteristic (Table 2) and modules' placement (Figure 14(b)) were adapted from [5] and a reasonable set of delay requirements were added.



(a)

Link	Assigned capacity (Gb/s)	Link	Assigned capacity (Gb/s)
00→01	1.87	23→22	0
01→02	1.53	00→10	0
02→03	0.93	10→20	0
10→11	0.89	01→11	0.86
11→12	0	11→21	0.53
12→13	0	02→12	0.97
20→21	1.10	12→22	1.69
21→22	1.14	03→13	0.93
22→23	1.27	13→23	0
01→00	1.66	10→00	0
02→01	1.22	20→10	0
03→02	1.22	11→01	0.89
11→10	0.86	21→11	0.6
12→11	0	12→02	1.46
13→12	0	22→12	1.15
21→20	0	13→03	1.03
22→21	0	23→13	1.27

(b)

Figure 11: Links capacity assigned by the capacity allocation algorithm for the DVD system.

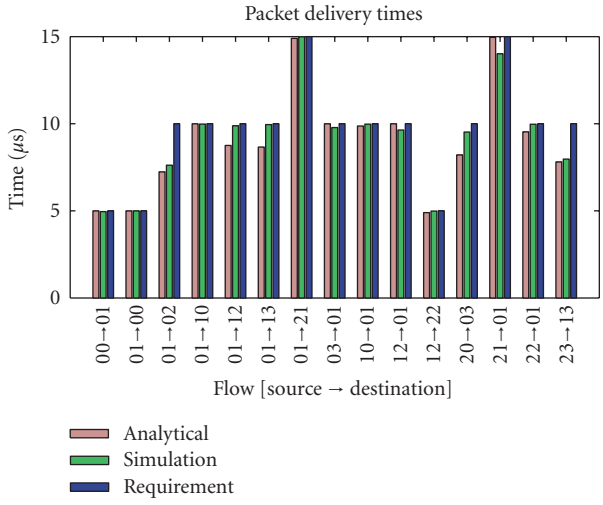


Figure 12: Flow delay (DVD decoder system).

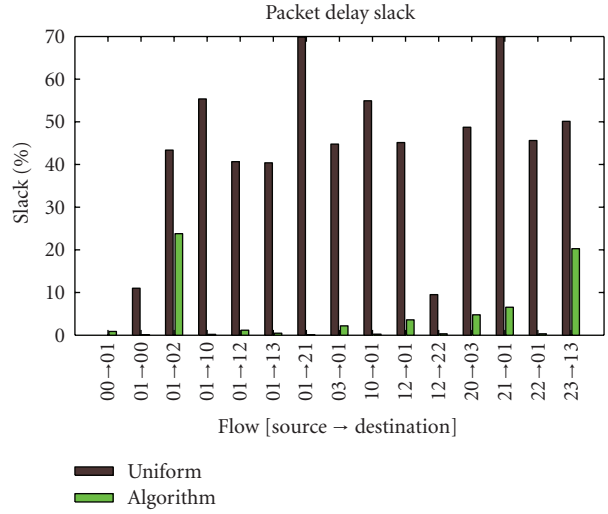


Figure 13: Mean packet delay slack as extracted from simulation, for uniform and algorithm-based allocation.

As in the previous example, we have compared the total capacity assigned by the capacity allocation algorithm with a uniform assignment that meets the same requirements.

Figure 15 presents the capacities assigned by the algorithm to each link and Figure 16 shows the resulting delays. In this example, a uniform allocation that meets all latency requirements consumes a total of 640 Gb/s, while the algorithm used only 369 Gb/s, thus reducing total capacity by 40%. Unlike the DVD decoder example, here there are almost no slacks remaining. This is due to the fact that most links are not shared by multiple flows, and hence flows do not benefit from capacity allocated in order to satisfy the needs of other flows.

In both examples presented above the capacity allocation algorithm was able to achieve significant resource savings thanks to different bandwidth and requirements of the different flows, and thanks to the diversity in link loads. Since some flows have weaker requirements than others, it is possible to allocate relatively low capacity to some of the links, without causing any flow to violate its delay requirement. As

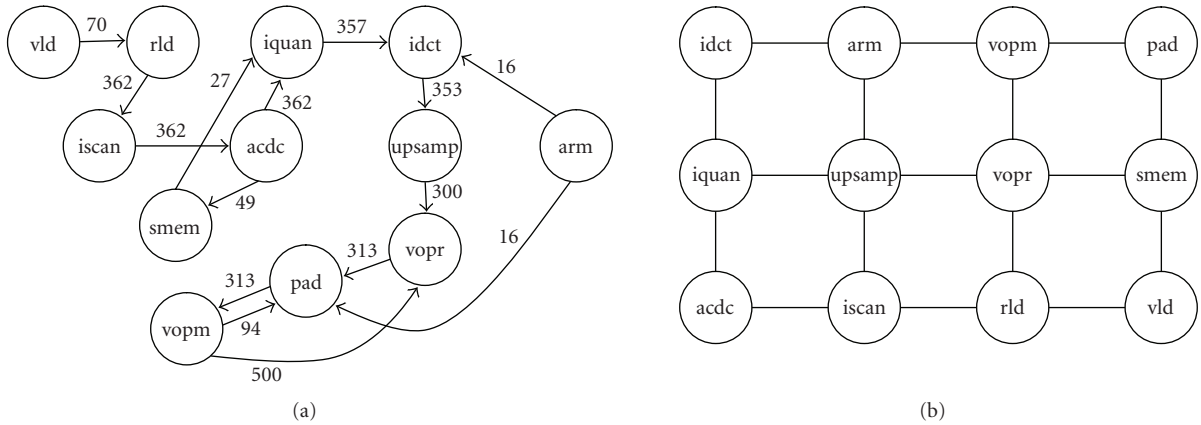
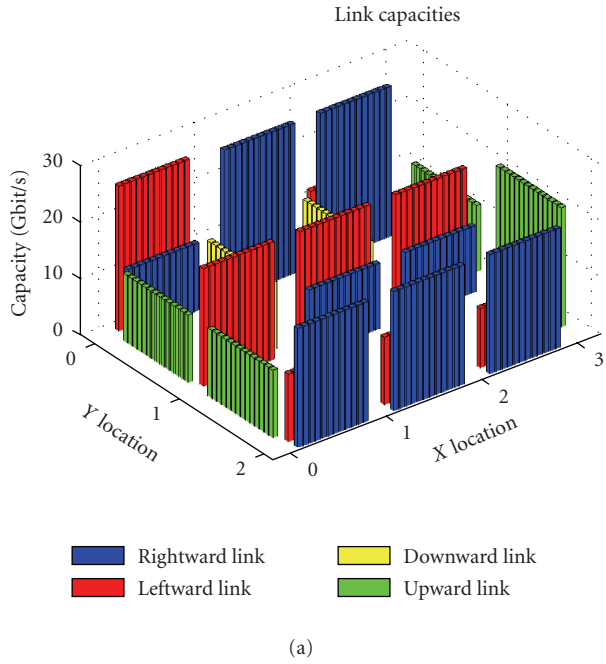


Figure 14: VOPD (a) task graph and (b) placement.



Link	Assigned capacity (Gb/s)	Link	Assigned capacity (Gb/s)
00→01	12.03	23→22	10.54
01→02	26.59	00→10	0
02→03	26.59	10→20	0
10→11	0	01→11	12.03
11→12	11.66	11→21	0
12→13	11.78	02→12	12.75
20→21	21.17	12→22	0
21→22	21.17	03→13	0
22→23	21.17	13→23	0
01→00	25.67	10→00	11.95
02→01	0	20→10	12.01
03→02	11.73	11→01	0
11→10	20.88	21→11	0
12→11	20.88	12→02	0
13→12	20.88	22→12	0
21→20	11.98	13→03	11.78
22→21	11.98	23→13	21.17

(b)

Figure 15: Links capacity assigned by the capacity allocation algorithm for the VOPD system.

a result, heterogeneous systems are more likely for a substantial resource saving than homogeneous ones.

6. SUMMARY

Allocating different capacities to different network links is an important phase in the design process of application-specific NoC-based systems. A good assignment algorithm should allocate network resources efficiently so that QoS and performance requirements are met but total cost is minimized.

The paper made two novel contributions: first, a simple static timing analysis delay model was presented. The analysis captures virtual channeled wormhole networks with different link capacities and eliminates the reliance on simulations for timing estimation. The paper also introduced an allocation algorithm that greedily assigns link capacities using the analytical delay model, so that packets of each flow arrive within the required time. Using design examples, we showed the potential benefit of automated link capacity allocation in a typical NoC-based SoC design, where the traffic is heterogeneous and critical delay requirements vary significantly.

APPENDIX

QNoC ARCHITECTURE

We present QNoC (Quality-of-service NoC) [4] architecture as an example of network characteristics and NoC-based system design. Although we have used the QNoC architecture for evaluation of our analysis and capacity allocation

Table 2: VOPD application.

Flow	Interarrival time (μs)	Packet length (flits)	Required delay (μs)
00→11	0.6916	128	0.2
01→00	15.26	128	0.08
01→03	15.26	128	0.08
02→03	2.597	128	0.1
02→12	0.4883	128	0.2
03→02	0.78	128	0.2
10→00	0.6839	128	0.2
11→12	0.8138	128	0.2
12→03	0.78	128	0.2
13→10	9.042	128	0.1
20→10	0.6744	128	0.2
20→13	4.982	128	0.1
21→20	0.6744	128	0.2
22→21	0.6744	128	0.2
23→22	3.488	128	0.2

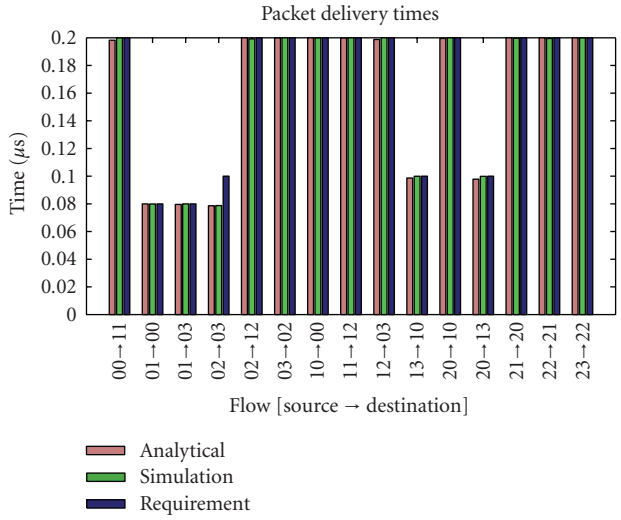


Figure 16: Flow delays (VOPD system).

technique, it should be noted that both the delay analysis (Section 3) and the allocation algorithm (Section 4) are applicable to other architectures, with different topologies and other fixed routing schemes.

Unlike traditional off-chip computer networks, which are built for future growth and compatibility with standards, on-chip networks can be designed and customized for an a priori known set of computing resources, given precharacterized traffic patterns and Quality-of-Service (QoS) requirements, while minimizing VLSI cost [4, 32, 33]. For example, topology generation, module placement, routing path selection and network link capacity allocation are performed at system design time, resulting in an efficient network implementation in terms of area and power.

The generic QNoC architecture is based on QoS wormhole packet routing and a planar irregular mesh topology

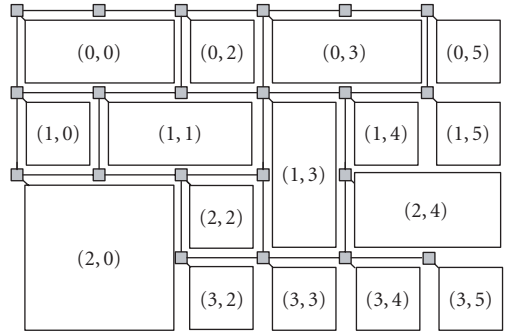


Figure 17: Example of an irregular mesh QNoC.

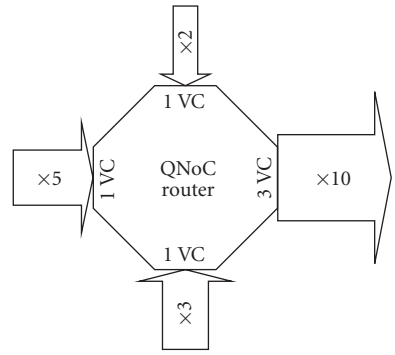


Figure 18: An example of an asymmetric network: multiple VCs are needed on the outgoing link.

(Figure 17). Our definition of an irregular mesh topology is identical to the full mesh including module addresses, except that some routers and links are missing. Wormhole routing [20] is an increasingly common interconnect architecture for NoC as it minimizes communication latencies and reduce buffer space. In QNoC, we enhance regular wormhole routing with the ability to provide different classes of QoS, related to end-to-end delay, throughput or round-trip delay.

The full design cycle of such a network consists of the following stages. First, the traffic and QoS requirements of the target SoC are identified. Next, the network is customized by appropriate module placement and by applying a least cost, shortest path routing function, thus minimizing power dissipation and maximizing network resource utilization [4, 33]. Finally, network load balancing is performed by link bandwidth allocation so that the predefined multiclass QoS requirements of each communication flow are satisfied while reducing the cost to a minimum (see Section 2). Note that this methodology is in contrast with off-chip networks, where the traffic requirements change over time and the routing mechanisms need to balance the load of this changing traffic over given topology and link capacities, which were designed for legacy or unknown loads. This important feature of QNoC allows constructing a heterogeneous wormhole network with interrouter links of different speeds.

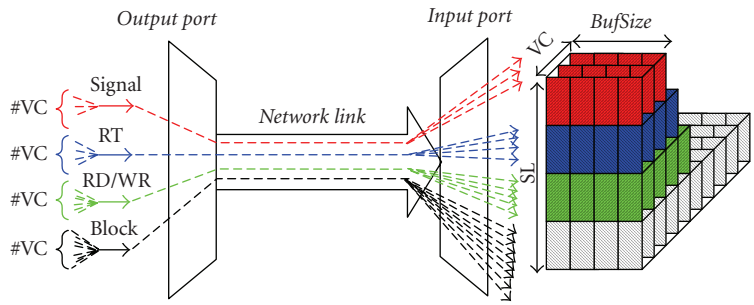


Figure 19: Multiplexing multiple SLs and VCs over a physical QNoC link.

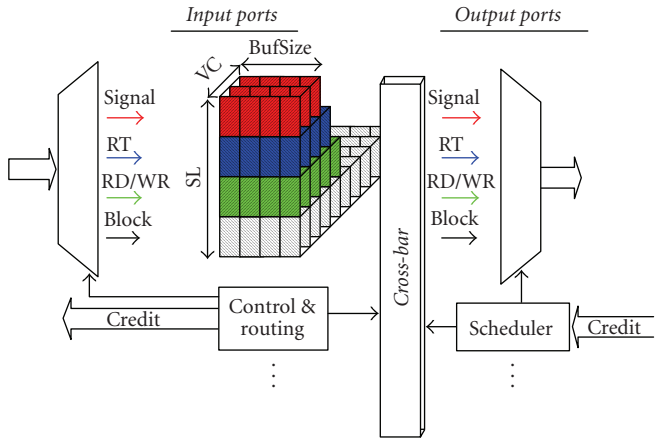


Figure 20: A QNoC router.

QNoC service levels

In order to support different classes of QoS for different kinds of on-chip traffic, we identify different types of communication requirements and define appropriate service levels (SL) to support them. For example, consider the following four different SLs: *Signaling* (urgent short packets that are given the highest priority), *Real-Time* (guaranteed bandwidth and latency to streamed audio and video), *Read/Write* (short memory and register accesses), and *Block-Transfer* (long messages such as DMA transfers).

In QNoC, a priority ranking among different SLs is established. For example, Signaling is given the highest priority and Block-Transfer the lowest. QNoC employs preemptive communication scheduling where data of a higher priority packet is always transmitted before that of a lower service level (a round-robin is employed within service levels). Additional service levels may be defined if desired.

Virtual channels

High performance wormhole-based interconnection networks are often equipped with virtual channels (VCs), which increase link utilization and overall network performance [21]. When links have different capacities, multiple VCs allow better utilization of high bandwidth links by multiplexing several slow flows over the link. Figure 18 depicts a simple example where the capacity of an outgoing link ($\times 10$) equals the overall capacity of the three incoming links. However, the high capacity outgoing link can be fully utilized only by simultaneous multiplexing of the incoming flows. This can be achieved by implementing multiple VCs on this high capacity outgoing link.

In QNoC every SL at each network link can be extended with its own number of VCs. The flits of different SLs/VCs that contend for the link bandwidth are time-multiplexed according to some arbitration policy over a single physical link (Figure 19). Previous research [21] has showed that adding VCs increases the maximal network throughput. We assume that physical links are split into an adequate number of VCs. In particular, we assume that a head flit can acquire a VC instantaneously on every link it traverses.

QNoC routers

QNoC router consists of input and output ports that are interconnected by a crossbar switch (Figure 20). Arriving flits are buffered at the input ports, awaiting transmission by the output ports. There are dedicated buffers for each SL and each VC. Relatively small buffers are allocated to each, capable of storing only a few flits. On the first flit of a packet, the router invokes a routing algorithm to determine to which

output port that packet is destined. The router then schedules the transmission for each flit on the appropriate output port.

Each output port of a router is connected to an input port of a next router via a communication link. The output port maintains the number of available flit slots per each SL and VC in the buffer of the next input port. The number is decremented upon transmitting a flit and incremented upon receiving a buffer-credit from the next router. When a space is available, the output port schedules transmission of flits that are buffered at the input ports and wait for transmission through that output port.

Router arbitration policy

Each output port schedules transmission of flits according to the availability of buffers in the next router and the service level priority of the pending flits. Once a higher priority packet appears on one of the input ports, transmission of the current packet is preempted and the higher priority packet gets through. Transmission of the lower priority packets is resumed only after all higher-priority packets have been serviced.

In QNoC, each SL is further divided into multiple VCs that are scheduled for transmission by the VC arbitration mechanism. VC arbitration consists of two phases. The first phase is VC allocation, in which available output VCs are allocated to the pending packets on the VCs from the input ports. When an output VC is allocated for a packet, this VC status becomes "active" and it can participate in the second phase of VC arbitration. The second phase is scheduling of these "active" VCs for transmission on each output link.

The QNoC architecture was modeled in detail in the OPNET environment [34], and flit-accurate simulations of every network instance can be performed.

ACKNOWLEDGMENTS

This work was partially supported by the Semiconductor Research Corporation (SRC), Intel Corporation, and the iSRC consortium.

REFERENCES

[1] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Proceedings of Design, Automation and Test in Europe Conference ands Exhibition (DATE '00)*, pp. 250–256, Paris, France, March 2000.

[2] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proceedings of the 38th Design Automation Conference (DAC '01)*, pp. 684–689, Las Vegas, Nev, USA, June 2001.

[3] S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. de Micheli, "A methodology for mapping multiple use-cases onto networks on chips," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '06)*, vol. 1, pp. 118–123, Munich, Germany, March 2006.

[4] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *Journal of Systems Architecture*, vol. 50, no. 2-3, pp. 105–128, 2004, special issue on network on chip.

[5] D. Bertozzi, A. Jalabert, S. Murali, et al., "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 2, pp. 113–129, 2005.

[6] J. Henkel, W. Wolf, and S. Chakradhar, "On-chip networks: a scalable, communication-centric embedded system design paradigm," in *Proceedings of the 17th International Conference on VLSI Design (VLSID '04)*, vol. 17, pp. 845–851, Mumbai, India, January 2004.

[7] K. Srinivasan, K. S. Chatha, and G. Konjevod, "An automated technique for topology and route generation of application specific on-chip interconnection networks," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD '05)*, pp. 231–237, San Jose, Calif, USA, November 2005.

[8] M. K.-F. Schäfer, T. Hollstein, H. Zimmer, and M. Glesner, "Deadlock-free routing and component placement for irregular mesh-based networks-on-chip," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD '05)*, pp. 238–245, San Jose, Calif, USA, November 2005.

[9] M. Palesi, S. Kumar, and R. Holsmark, "A method for router table compression for application specific routing in mesh topology NoC architectures," in *Proceedings of the 6th International Workshop on Architectures, Modeling, and Simulation (SAMOS '06)*, pp. 373–384, Samos, Greece, July 2006.

[10] K. Goossens, J. Dielissen, O. P. Gangwal, S. G. Pestana, A. Rădulescu, and E. Rijpkema, "A design flow for application-specific networks on chip with guaranteed performance to accelerate SOC design and verification," in *Proceedings of Design, Automation and Test in Europe (DATE '05)*, vol. 2, pp. 1182–1187, Munich, Germany, March 2005.

[11] J. Hu and R. Marculescu, "Application-specific buffer space allocation for networks-on-chip router design," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD '04)*, pp. 354–361, San Jose, Calif, USA, November 2004.

[12] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "HERMES: an infrastructure for low area overhead packet-switching networks on chip," *Integration, the VLSI Journal*, vol. 38, no. 1, pp. 69–93, 2004.

[13] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini, "XPIPES: a latency insensitive parameterized network-on-chip architecture for multi-processor SoCs," in *Proceedings of the 21st International Conference on Computer Design (ICCD '03)*, pp. 536–539, San Jose, Calif, USA, October 2003.

[14] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch, "The Nostrum backbone-a communication protocol stack for networks on chip," in *Proceedings of the 17th International Conference on VLSI Design (VLSID '04)*, pp. 693–696, Mumbai, India, January 2004.

[15] M. Coenen, S. Murali, A. Ruadulescu, K. Goossens, and G. de Micheli, "A buffer-sizing algorithm for networks on chip using TDMA and credit-based end-to-end flow control," in *Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis*, pp. 130–135, Seoul, Korea, October 2006.

[16] G. Ascia, V. Catania, and M. Palesi, "Multi-objective mapping for mesh-based NoC architectures," in *Proceedings of the 2nd International Conference on Hardware/Software Codesign and Systems Synthesis*, pp. 182–187, Stockholm, Sweden, September 2004.

[17] S. Murali and G. de Micheli, "SUNMAP: a tool for automatic topology selection and generation for NoCs," in *Proceedings of the 41st Design Automation Conference*, pp. 914–919, San Diego, Calif, USA, June 2004.

[18] U. Y. Ogras and R. Marculescu, ""It's a small world after all": NoC performance optimization via long-range link insertion," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 7, pp. 693–706, 2006.

[19] N. Banerjee, P. Vellanki, and K. S. Chatha, "A power and performance model for network-on-chip architectures," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '04)*, vol. 2, pp. 1250–1255, Paris, France, February 2004.

[20] W. J. Dally and C. J. Seitz, "The torus routing chip," *Distributed Computing*, vol. 1, no. 4, pp. 187–196, 1986.

[21] W. J. Dally, "Virtual-channel flow control," in *Proceedings of the 17th Annual International Symposium on Computer Architecture (ISCA '90)*, pp. 60–68, Seattle, Wash, USA, June 1990.

[22] H. Sarbazi-Azad, A. Khonsari, and M. Ould-khaoua, "Performance analysis of deterministic routing in wormhole *k*-ary *n*-cubes with virtual channels," *Journal of Interconnection Networks*, vol. 3, no. 1-2, pp. 67–73, 2002.

[23] S. Loucif and M. Ould-Khaoua, "Modeling latency in deterministic wormhole-routed hypercubes under hot-spot traffic," *Journal of Supercomputing*, vol. 27, no. 3, pp. 265–278, 2004.

[24] C. Roche, P Palnati, M. Gerla, F. Neri, and E. Leonardi, "Performance of congestion control mechanisms in wormhole routing networks," in *Proceedings of the 16th Annual Joint Conference of the IEEE Computer and Communications Societies, Driving the Information Revolution (INFOCOM '97)*, vol. 3, pp. 1365–1372, Kobe, Japan, April 1997.

[25] J. Kim and C. R. Das, "Hypercube communication delay with wormhole routing," *IEEE Transactions on Computers*, vol. 43, no. 7, pp. 806–814, 1994.

[26] R. I. Greenberg and L. Guan, "Modeling and comparison of wormhole routed mesh and torus networks," in *Proceedings of the 9th IASTED Iasted International Conference on Parallel and Distributed Computing Systems*, Washington, DC, USA, October 1997.

[27] B. Ciciani, M. Colajanni, and C. Paolucci, "Performance evaluation of deterministic wormhole routing in *k*-ary *n*-cubes," *Parallel Computing*, vol. 24, no. 14, pp. 2053–2075, 1998.

[28] J. T. Draper and J. Ghosh, "A comprehensive analytical model for wormhole routing in multicomputer systems," *Journal of Parallel and Distributed Computing*, vol. 23, no. 2, pp. 202–214, 1994.

[29] W. J. Dally, "Performance analysis of *k*-ary *n*-cube interconnection networks," *IEEE Transactions on Computers*, vol. 39, no. 6, pp. 775–785, 1990.

[30] L. Kleinrock, *Queuing Systems, Volume 1: Theory*, John Wiley & Sons, New York, NY, USA, 1975.

[31] J. P. Fishburn and A. E. Dunlop, "TILOS: a posynomial programming approach to transistor sizing," in *Proceedings of the IEEE International Conference on Computer Aided Design (ICCAD '85)*, pp. 326–328, Santa Clara, Calif, USA, November 1985.

[32] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Cost considerations in network on chip," *Integration, the VLSI Journal*, vol. 38, no. 1, pp. 19–42, 2004.

[33] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Routing table minimization for irregular mesh NoCs," in *Proceedings of Design Automation and Test in Europe (DATE '07)*, Nice, France, March 2007.

[34] OPNET modeler, http://www.opnet.com/.