Mathematical Formalisms for Performance Evaluation of Networks-on-Chip

ABBAS ESLAMI KIASARI, AXEL JANTSCH, and ZHONGHAI LU,
KTH Royal Institute of Technology

This article reviews four popular mathematical formalisms—*queueing theory, network calculus, schedulability analysis*, and *dataflow analysis*—and how they have been applied to the analysis of on-chip communication performance in Systems-on-Chip. The article discusses the basic concepts and results of each formalism and provides examples of how they have been used in Networks-on-Chip (NoCs) performance analysis. Also, the respective strengths and weaknesses of each technique and its suitability for a specific purpose are investigated. An open research issue is a unified analytical model for a comprehensive performance evaluation of NoCs. To this end, this article reviews the attempts that have been made to bridge these formalisms.

Categories and Subject Descriptors: C.4 [**Computer Systems Organization**]: Performance of Systems—*Modeling techniques*

General Terms: Design, Performance

Additional Key Words and Phrases: System-on-chip (SoC), network-on-chip (NoC), performance evaluation, analytical modeling

ACM Reference Format:
Kiasari, A. E., Jantsch, A., and Lu, Z. 2013. Mathematical formalisms for performance evaluation of networks-on-chip. ACM Comput. Surv. 45, 3, Article 38 (June 2013), 41 pages.
DOI: http://dx.doi.org/10.1145/2480741.2480755

1. INTRODUCTION

It is essential to gain a solid understanding of a system's performance in advance of the system being implemented in detail and built. Therefore, performance models have been deployed in system design for many decades and, more recently, have been adopted for the study of System-on-Chip (SoC). In modern SoCs, the on-chip communication infrastructure or Network-on-Chip (NoC) is a dominant factor for design, validation, and performance analysis. SoC designers are interested in performance evaluation, given that their goal is either to provide the highest performance at a given cost or to provide a minimum level of performance at the lowest possible cost. In both cases, a reliable measure of performance is indispensible. However, the focus in the first case is typically on average performance, while the worst-case performance is the main metric in the latter case. In real-time systems, such as automotive or avionic applications, the worst-case execution time is of particular concern; it is important to know how much time might be needed in the worst case in order to guarantee that the task will always finish its jobs before the predetermined deadline. However, the worst-case-based design results in resource over-dimensioning. Therefore, average-case-based design methods are usually used for non-time-critical applications in order to achieve a more efficient system.

This work is funded in part by Intel Corporation through a research gift.
Corresponding author's email: kiasari@kth.se.

DOI: http://dx.doi.org/10.1145/2480741.2480755

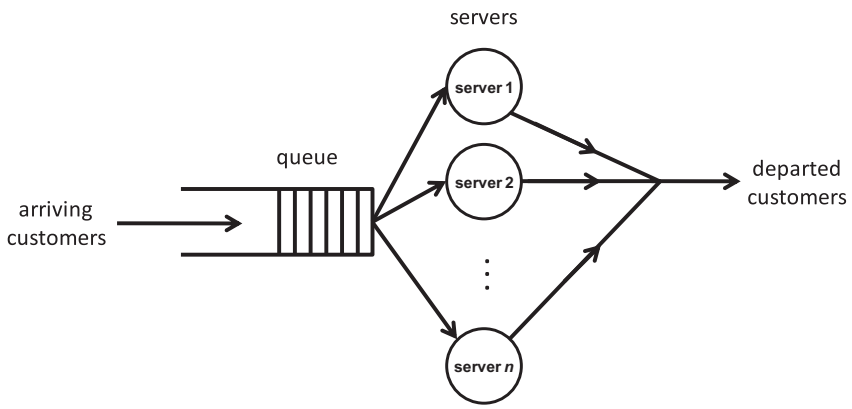


Fig. 1. Model of a queueing system.

Performance estimation tools can be classified in simulation models and mathematical models. SoC designers have explored the design space using detailed simulations in order to tackle performance analysis. Although simulation tools are flexible and accurate, the complexity of modern SoCs imposes a firm limit on what can reasonably be simulated. Another disadvantage of a simulation-based design process is that the nonlinear and nonmonotonic behavior of system performance makes it difficult to draw conclusions from the simulation results regarding how to adapt the system hardware or its programming. It is also difficult to determine the worst-case behavior of the SoC. An alternative approach is to build an abstract analytical model for the architecture of the on-chip communication. An appropriate analytical model can estimate the desired performance metrics very early on in the design phase, in a fraction of the time that simulation would take. Although the use of high-level models conceals a lot of complex technological aspects, it facilitates rapid exploration of the NoC's design space. Also, the analytical models provide not only the timing properties of the system, but also useful feedback about the system's behavior. Consequently, such models can be invoked in any optimization loop for NoCs in order to obtain fast and accurate performance estimations. Therefore, analytical models have a place alongside simulation in SoC performance analysis, and their importance is likely to grow as SoC communication architectures become increasingly complex and irregular. Several popular analysis methods, developed in other context years or even decades ago, have recently been adapted to NoC analysis.

The purpose of this survey is to recapitulate the results from the mathematical formalisms—*queueing theory, network calculus, schedulability analysis*, and *dataflow analysis*—and their application to the analysis of NoCs. For each, we review the basic concepts and results in Sections 2 to 5. Section 6 considers a simple application and shows how these formalisms can be used to evaluate the performance of a system. Section 7 presents attempts to combine these methods. Finally, their respective strengths, weaknesses, and suitability for a specific purpose are summarized in Section 8.

2. QUEUEING THEORY

2.1. Overview

Queueing theory is a branch of probability theory. As shown in Figure 1, in a queueing system, a population of *customers* enters a *service facility*, at some time, which includes one server or multiple servers, in order to obtain service. If a new customer arrives and

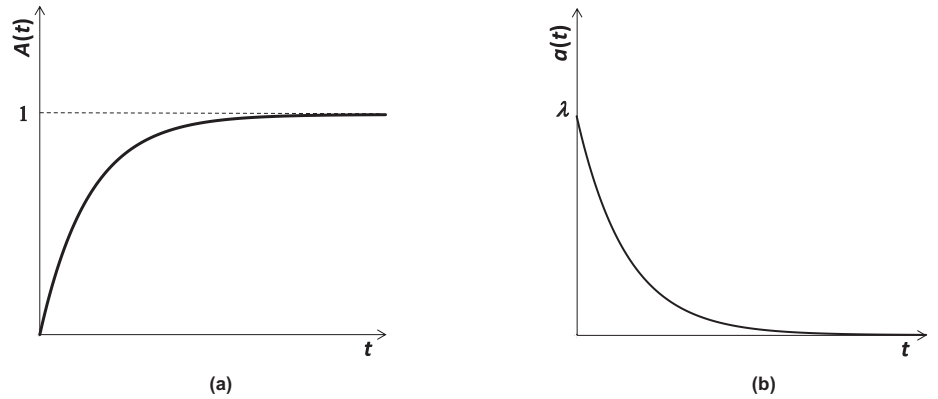


Fig. 2. (a) CDF and (b) pdf of an interarrival time with exponential distribution.

all servers are busy, it enters a *queue* and waits until one server becomes available. Therefore, in order to analyze such a system, we must identify the arrival process as well as the structure and discipline of the service facility.

In queueing theory, the arrival process and service time are specified probabilistically. Generally, the arrival process is described in terms of the *cumulative distribution function* (*CDF*) of the *interarrival times* of customers (i.e., the time between two successive arrivals) and is denoted $A(t)$, where

$$A(t) = P\{\text{interarrival time} \leq t\}. \tag{1}$$

The notation $P\{X\}$ denotes the probability of the event X. $A(t)$ is a nonnegative and nondecreasing function of t. Also, the *probability density function* (*pdf*) of interarrival times is

$$a(t) = \frac{d}{dt}A(t). \tag{2}$$

For instance, if the interarrival time of customers has exponential distribution with parameter $\lambda(\lambda > 0)$, then its CDF is given by

$$A(t) = \begin{cases} 1 - e^{-\lambda t} & t \geq 0, \\ 0 & t < 0, \end{cases} \tag{3}$$

which is sketched in Figure 2(a). The corresponding pdf of the interarrival time is

$$a\,(t) = \begin{cases} \lambda e^{-\lambda t} & t \geq 0, \\ 0 & t < 0, \end{cases} \tag{4}$$

which is sketched in Figure 2(b).

The assumption in queueing systems is that these interarrival times are independent and identically distributed random variables. Similarly, *service time,* that is, the length of time that a customer spends in the service center, is considered as another continous random variable whose CDF and pdf, respectively, are

$$B(x) = P\{\text{service time} \leq x\}, \tag{5}$$

$$b(x) = \frac{d}{dx}B(x). \tag{6}$$

Regarding the structure and discipline of the service facility, a variety of additional quantities must be specified, such as the extent of storage capacity available to hold

waiting customers, the number of service stations available, the queueing discipline (FCFS, LCFS, and random order of service), etc.

In addition to interarrival and service times distributions, queueing systems may differ in the number of servers, the capacity of the queue (infinite or finite), and the service discipline. Some common service disciplines are the following.

—*FCFS* (*First-Come, First-Served*). A customer that finds the service center busy goes to the end of the queue.
—*LCFS* (*Last-Come, First-Served*). A customer that finds the service center busy proceeds immediately to the head of the queue. It will be served next, given that no further customers arrive.
—*RS* (*random service*). The customers in the queue are served in random order.
—*RR* (*round robin*). Every customer gets a time slice. If its service is not completed, it will re-enter the queue.
—*PR* (*Priority*). Every customer has a (static or dynamic) priority; the server always selects the customers with the highest priority. This scheme can use preemption or not.

The *Kendall notation* is used for a short characterization of queueing systems [Bolch et al. 2006]. A queueing system description looks as $A/B/m/K$—S, where A denotes the distribution of the customer interarrival time, B denotes the distribution of the service time, m denotes the number of servers, K denotes the maximum capacity of the queue in the finite case (if $K = \infty$, then this letter is omitted), and the optional S denotes the service discipline used. If S is omitted, the service discipline is always FCFS. For A, the following abbreviations are very common.

—*M* (*Markov property*). This denotes the exponential distribution with an average arrival rate of λ customers/time unit. In other words, the number of customers follows a Poisson distribution with the average of one customer per $1/\lambda$ time unit.
—*D* (*Deterministic*). The interarrival times are constant and have the same value.
—*G* (*General*). General distribution, not further specified. In most cases, at least the mean and the variance are known.

Similarly, B can be specified by these notations (*M, D, and G*) to describe the distribution of service time. For instance, the *M*/*G*/2/10-*RS* queueing system can be described as follows.

—The customer interarrival times are exponentially distributed (with specified average).
—The service time distribution is arbitrary (with specified average and variance).
—There are two servers in the system.
—The queue has room for at most ten customers.
—The customers in the queue are served in random order.

After specifying a queueing system, it is appropriate that we identify the measures of performance and effectiveness that we shall obtain by analysis. Basically, we are interested in the waiting time for a customer, the number of customers in the queue, the length of busy and idle periods of the server (i.e., the continuous interval during which the server is busy or idle), and the current work backlog (unfinished work) expressed in units of time. All these quantities are random variables, and thus we seek their complete probabilistic description, such as their pdf. However, in most applications, it is enough to calculate the first few moments (mean, variance, etc.). Also within the scope of queueing theory is the case where several servers are arranged in a network and customers move through the network to visit several servers.

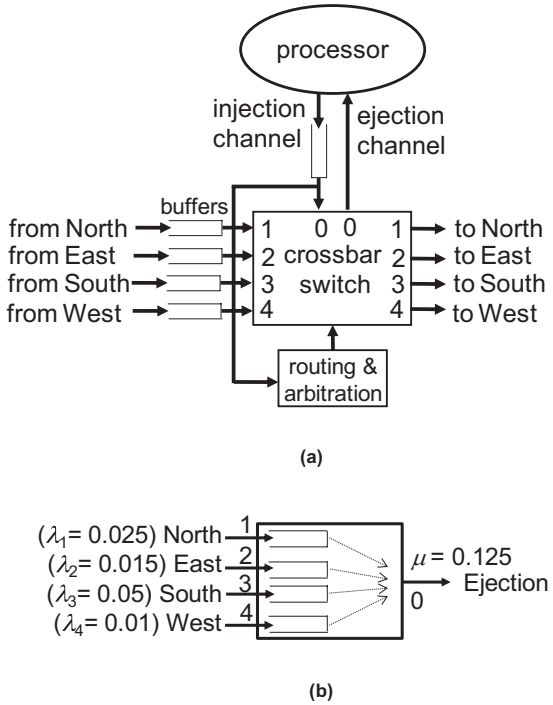


Fig. 3. (a) The structure of a router in a 2D mesh network; (b) queueing model of the ejection channel.

2.2. An Example

As an example, consider a packet-switched mesh network on which packet routing is carried out by a router at each node. Every node contains a processor and a router. Packets are injected into the network on crossbar input port 0 (injection channel) and leave on output port 0 (ejection channel), as shown in Figure 3(a). In the following, we utilize queueing theory to estimate the average waiting time for accessing the ejection channel. The following assumptions are made when developing the queueing model.

—The packet arrivals to the northern, eastern, southern, and western input channels are independent and follow Poisson processes with mean rates of $\lambda_1 = 0.025, \lambda_2 = 0.015, \lambda_3 = 0.05$, and $\lambda_4 = 0.01$ packets/cycle, respectively.
—An infinite FIFO (first in, first out) buffer is associated only with each input channel, for storing packets in transit.
—Messages are broken into some packets of fixed length. When a packet arrives on an ejection channel, it is accepted by the processor in eight network cycles. Therefore, we can model the ejection channel as a constant-rate server with service rate of $\mu = 1/8 = 0.125$ packets/cycle.
—We consider an ejection channel as a server in which the packets have preferential treatment based on priorities associated with them. We assume that the priority of a packet is an integer fixed at arrival time and that packets with priority index 1, 2, 3, and 4 come from northern, eastern, southern, and western input channels, respectively. We say one packet has higher priority than another if it belongs to a priority class with a lower index. For the service discipline, we assume that whenever a packet has traversed the ejection channel completely, the ejection channel is assigned next to that packet at the head of the highest-priority nonempty queue.

To calculate the average waiting time for ejection channels, we model the ejection channel as an *M*/*D*/1 priority queue, as shown in Figure 3(b). The average waiting time of random arrivals to the *i*th queue of *M*/*D*/1 system, $\bar{W}_i$, can be written [Bolch et al. 2006] as

$$\bar{W}_k = \frac{\frac{1}{2\mu}\sum_{i=1}^{4}\rho_i}{\left(1-\sum_{i=1}^{k-1}\rho_i\right)\left(1-\sum_{i=1}^{k}\rho_i\right)}, \tag{7}$$

where $\rho_i = \lambda_i/\mu$. In this example, $\rho_1 = 0.2$, $\rho_2 = 0.12$, $\rho_3 = 0.4$, and $\rho_4 = 0.08$. Thus, the waiting times can be computed as

$$\begin{aligned}
\bar{W}_1 &= \frac{3.2}{1-0.2} = 4.0 \text{ cycles};\\
\bar{W}_2 &= \frac{3.2}{(1-0.2)(1-0.32)} = 5.9 \text{ cycles};\\
\bar{W}_3 &= \frac{3.2}{(1-0.32)(1-0.72)} = 16.8 \text{ cycles};\\
\bar{W}_4 &= \frac{3.2}{(1-0.72)(1-0.8)} = 57.1 \text{ cycles}.
\end{aligned}$$

Using Little's theorem [Kleinrock 1975], the average number of packets in each input port can be computed as follows.

$$\begin{aligned}
\bar{N}_1 &= \lambda_1\bar{W}_1 = 0.10 \text{ packet};\\
\bar{N}_2 &= \lambda_2\bar{W}_2 = 0.09 \text{ packet};\\
\bar{N}_3 &= \lambda_3\bar{W}_3 = 0.84 \text{ packet};\\
\bar{N}_4 &= \lambda_4\bar{W}_4 = 0.57 \text{ packet}.
\end{aligned}$$

2.3. Applications in NoCs

Similar to other networks, traffic patterns play an important role in the performance of NoCs; consequently, traffic models are critically needed for effectively evaluating existing and new NoC designs. As a result, network traffic modeling is a first step towards understanding the design space of NoC architectures, protocols, and implementations. Soteriou et al. [2006] proposed an on-chip traffic model for homogeneous NoCs. The model is based on three statistical parameters: temporal burstiness, spatial hop distribution, and spatial injection distribution. The authors showed that their model captures the characteristics of NoC traffic accurately when compared to actual NoC application traces gathered from full system simulations of chip platforms. Varatkar and Marculescu [2004] found long-range dependent behavior in communications traffic between different parts of the MPEG-2 video decoding application. They presented an approach for analyzing such a traffic pattern based on self-similar processes. They showed that characterizing the degree of self-similarity via the Hurst parameter helps in finding the optimal buffer-length distribution. However, Scherrer et al. [2009] showed that long-range dependence is not an ubiquitous property of the traffic produced by on-chip processors running multimedia applications. Using a cycle-accurate simulator of a complete SoC, they showed that long-range dependence impact on the Network-on-Chip is highly correlated with the low-level communication protocol used.

Performance evaluation techniques for NoCs have been inherited from parallel and distributed processing research groups. Many of the previous analytical latency models

in off-chip networks have been formulated for a specific topology and traffic pattern [Kim and Das 1994; Kiasari et al. 2008a]. Queueing theory has been used to estimate average performance metrics, such as average packet latency, average throughput, average energy and power consumption, and average resource utilization. System designers utilize these metrics to make decisions for solving problems, such as module placement [Kiasari et al. 2008d], routing decision [Kiasari et al. 2010], buffer configuration [Hu et al. 2006], and link capacity [Guz et al. 2007].

Guan et al. [1993] proposed an analytical model for a general topology with an exponential packet length distribution. Their approach has high complexity for high-dimensional networks. Using queueing theory, Hu and Kleinrock [1997] presented a general analytical model for wormhole routing to estimate the average packet latency in interconnection networks. In order to provide fast performance estimates during the design cycle, Kim et al. [2005] developed a queueing theory-based model for quantifying the performance and energy behavior of on-chip networks. They assumed that packet arrivals at all input channels have Markov property. Hu et al. [2006] considered $M/M/1/K$ queueing models and solved a series of nonlinear equations to quickly analyze the current buffer size configuration and detect the performance bottlenecks in the router channels. This model was then used in buffer-sizing problems in packet-switched NoCs. More precisely, given the traffic characteristics of the target application and the total budget of the available buffering space, the proposed model automatically assigns the buffer depth for each input channel, in different routers across the chip, such that the average packet latency is minimized in the system. Based on $M/M/1$ queueing model, an analytical delay model for virtual channeled wormhole networks was proposed for link capacity allocation in NoC-based systems by Guz et al. [2007]. This assignment algorithm allocates network resources efficiently so that quality of service (QoS) and performance requirements are met.

Hur et al. [2008] presented a performance analysis of hard and soft on-chip networks for FPGAs. They applied Jackson's queueing model [Jackson 1957] to analyze the performance of a multiprocessor SoC. They further used Jackson's model to analyze circuit-switched NoCs and showed that the hardwired networks perform significantly better than conventional soft NoCs. A Markovian performance model for torus on-chip networks with deterministic routing and wormhole switching was proposed by Kiasari et al. [2008b]. The model was then used to estimate the power consumption of all routers. This model is restricted to Poisson arrival process and uniform traffic pattern. Kiasari et al. [2008c] modeled each channel in an NoC with a $G/G/1$-PR queueing model ($G/G/1$ queue with priority discipline) and estimated per-flow average packet latency. However, the modeling approach is limited to k-ary n-cube networks with single flit buffers and a dimension-order routing algorithm. This model was used to map the processing cores onto an SoC architecture such that the average communication delay is minimized [Kiasari et al. 2008d]. A case study of using an analytical method based on Markov chain stochastic processes for latency evaluation of an NoC arranged in a 2D mesh topology with a deterministic routing algorithm and uniform traffic pattern was presented by Foroutan et al. [2009].

Foroutan et al. [2010] proposed a generic analytical model for estimating communication latencies and link-buffer utilizations for wormhole-switched NoCs with a given application mapped on it. This work correctly models the resulting interdependencies between the routers. An analytical performance model for wormhole-switched NoCs has been proposed by Ogras et al. [2010]. Using an $M/G/1$ queueing model, the average number of packets at each buffer is computed. This model provides three performance metrics, namely average buffer utilization, average packet latency, and network throughput. Another analytical model for estimating the communication performance

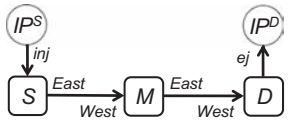


Fig. 4. A two-hop flow from IP^S (source) to IP^D (destination).

of wormhole-switched NoCs is presented by Cheng et al. [2011]. This model supports arbitrary network topology with virtual channels. To resolve the inherent dependency of successive links occupied by a packet, the authors use a routing path decomposition approach to generating a series of ordered link categories. Next, they used $M/M/1$ and $M/M/1/K$ queueing models to derive the transmission latency of network components. The analytical model proposed by Krimer et al. [2011] is inspired by industrial workflow modeling techniques. The authors introduced a packet-level static timing analysis for wormhole-switched NoCs with virtual channels. It relies on a reduced Markov chain to represent the network state, including the occupancy of all buffers. The model handles any topology, link capacities, and buffer sizes and provides per-flow delay analysis. Wang et al. [2011] proposed a performance analytical model using a semi-Markov process for estimating the average packet latency in NoCs. More precisely, a semi-Markov process is used to describe the behavior of each link in the network, and the header flit delay is calculated.

Kiasari et al. [2012] extended the proposed queueing model of Kiasari et al. [2008c] to support arbitrary topology, buffer size, and oblivious routing algorithm. This model is developed for wormhole switching, and it supports any kind of spatial and temporal traffic patterns. Spatial traffic pattern refers to the distribution of packet destinations, and temporal component of traffic is determined by the distribution of interarrival times of packets. It means that the model accepts both Poisson and non-Poisson arrival processes. The average packet latency (L) is used as the performance metric. The authors assume that the packet latency spans the instant when the packet is created, to the time when the packet is delivered to the destination node. They also assume that the packets are consumed immediately once they reach their destination nodes. In Figure 4, consider a flow which is generated in IP^S and reaches its destination (IP^D) after traversing routers R^S, R^M, and R^D. The latency of this packet ($L^{S\to D}$) consists of two parts: the latency of head flit ($L_h^{S\to D}$) and the latency of body flits (L_b). $L_h^{S\to D}$is the time since the packet is created in IP^S until the head flit reaches the IP^D, including the transfer times of a flit across the injection and ejection channels (t_{inj} and t_{ej}), routing decision delay for a packet (t_r), crossing time of a flit over the crossbar switch (t_s), transfer time of a flit across a wire between two adjacent routers (t_w), and queueing time spent at the source node and intermediate nodes ($W^N_{i\to j}$, the mean waiting time for a packet from input port i of node N to output port j of the same node). Having looked at Figure 4, we can infer that the latency of a head flit of a two-hops packet includes injection channel delay (t_{inj}), first router delay ($t_r + W^S_{inj\to East} + t_s$), internode wire delay (t_w), second router delay ($t_r + W^M_{West\to East} + t_s$), internode wire delay (t_w), third router delay ($t_r + W^D_{West\to ej} + t_s$), and ejection channel delay (t_{ej}). Therefore, it can be written as

$$\begin{aligned} L_h^{S\to D} &= t_{inj} + \left(t_r + W^S_{inj\to East} + t_s\right) \\ &\quad + t_w + \left(t_r + W^M_{West\to East} + t_s\right) \\ &\quad + t_w + \left(t_r + W^D_{West\to ej} + t_s\right) + t_{ej}. \end{aligned} \tag{8}$$

Once the head flit arrives at the destination, the body flits follow the header flit in a pipelined fashion. Therefore, the only unknown parameter for computing the latency is $W^N_{i\rightarrow j}$. This value was calculated using a priority queueing model. Later, in Section 6.1, we show how to use this analytical model to estimate the average packet latency.

Studies done by Varatkar and Marculescu [2004] have demonstrated that the traffic of some multimedia applications exhibits a long-range dependent behavior which has considerable impact on queueing performance. The assumption of the traditional Poisson arrival process is inherently unable to capture such a traffic pattern. Therefore, it is crucial to reexamine the performance properties of interconnection networks in the context of more realistic traffic models before practical implementations show their potential faults. Toward this end, Min and Ould-Khaoua [2004] proposed an analytical queueing model for wormhole-switched networks in the presence of self-similar traffic. This study reveals that the network suffers considerable performance degradation when subjected to self-similar traffic, stressing the great need for improving network performance to ensure efficient support for this type of traffic.

In queueing theory, generally, the average quantities in an equilibrium state are considered. Characterizing the transient behavior of queueing systems is known as a very difficult problem which has been addressed by either simplified analytical models or simulation [Odoni and Roth 1983; van As 1986; Bertsimas and Mourtzinou 1997; Yang and Liu 2010]. As an alternative approach, Bogdan and Marculescu [2007] proposed a statistical physics-inspired framework to analyze the traffic dynamics in NoCs and show how the nonstationary effects of the system workload can be effectively captured. The temperature of a physical system is replaced by the NoC packet injection rate, and the authors predicted that the buffer occupancy follows a power law distribution. Later, they addressed the buffer sizing problem under non-equilibrium conditions [Bogdan and Marculescu 2009]. The main idea in this model is that packets move from one node to another in a manner that is similar to particles moving in a Bose gas and migrating between various energy levels as a consequence of temperature variations. Bogdan and Marculescu [2010, 2011] also investigated the impact of nonstationary effects (as a function of packet injection rate) on buffer overflow probability and node-to-node latency exceedance probability.

In order to analyze a queueing system, it is necessary to know something about the laws governing the arrival pattern, the logic governing the behavior of the queue, and the characteristics of the service facility. Queueing theory is concerned with the mathematical analysis of such systems subject to demands whose occurrences and lengths can, in general, be specified only probabilistically.

3. NETWORK CALCULUS

3.1. Overview

Network calculus is a mathematical framework for deriving the worst-case bounds on maximum latency and backlog in a single node and a network of nodes. Therefore, it can be seen as a theory for analyzing performance guarantees in computer networks. Cruz [1991a, 1991b] pioneered the network calculus, and based on Cruz's foundation, Chang [2000] and Le Boudec and Thiran [2001] have further developed the network calculus theory and based it on *min-plus algebra*. The basic elements in this algebra are arrival curves as an abstraction of application traffic and service curves as an abstraction of network elements. Network calculus is similar to conventional system theory, in which a system consists of an input function, a transfer function, and an output function. The difference to conventional system theory is that min-plus algebra is used, where addition and multiplication are replaced by minimum and addition, respectively. As in conventional system theory, a key operation in network calculus is

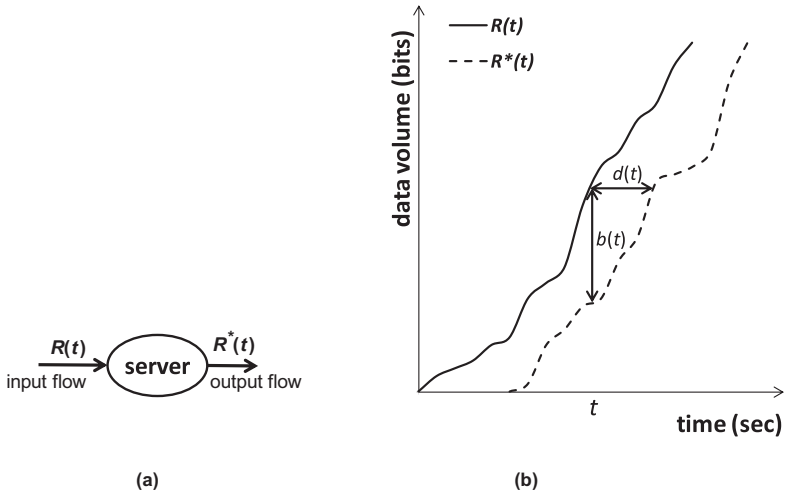


Fig. 5. Backlog and virtual delay of a flow at time t.

the *min-plus convolution*. The min-plus convolution of $f(t)$ and $g(t)$ is defined as

$$(f \otimes g)(t) = \inf_{0 \le s \le t} \{f(s) + g(t-s)\}. \tag{9}$$

The infimum (*inf*) is similar to the minimum. A minimum of a set is the smallest element of the set, and of course, is in the set. An infimum of a set is the greatest lower bound of the set and need not be in the set. The same applies for the maximum and supremum (*sup*).

In network calculus theory, cumulative functions $R(t)$ and $R^*(t)$ describe the *input function* and *output function*, respectively. They represent the number of bits (words or packets) seen on the input and output dataflow in time interval $[0, t]$. It is obvious that functions R and R^* are always monotonically increasing functions. System S receives input data and delivers the output data after a variable delay. System S might be, for example, a single buffer served at a constant rate, a complex communication node, or even a complete network.

The *backlog* is the number of bits that are held inside the system; if the system is a single buffer, it determines the queue length. In contrast, if the system is more complex, then the backlog is the number of bits in transit, assuming that we can observe input and output simultaneously. Therefore, for a lossless system, the *backlog* at time t is

$$b(t) = R(t) - R^*(t). \tag{10}$$

The virtual delay at time t is the delay that would be experienced by a bit arriving at time t if all bits received before it are served before it. Hence, the virtual delay at time t is

$$d(t) = \inf_{\tau \ge 0}\{R(t) \le R^*(t+\tau)\}. \tag{11}$$

In other words, $d(t)$ is the smallest value satisfying $R^*(t + d(t)) = R(t)$. As shown in Figure 5, the backlog and virtual delay are shown as the vertical and horizontal deviation between input and output functions, respectively. In network calculus theory, the input and transfer functions are referred to as *arrival curve* and *service curve*, respectively.

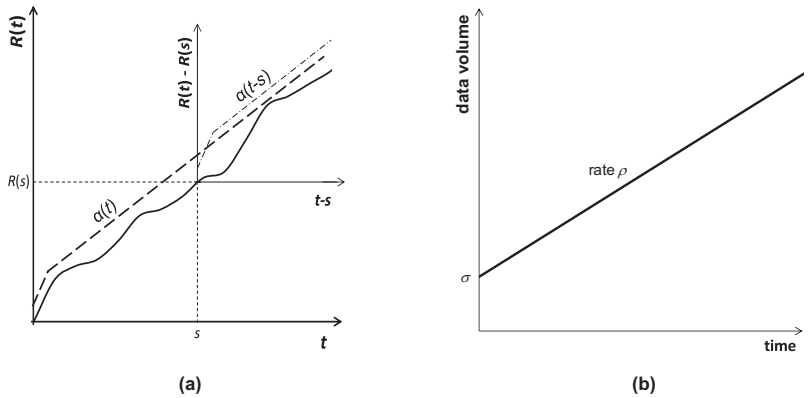


Fig. 6. (a) Input function $R(t)$ is constrained by an arrival curve $\alpha(t)$; (b) leaky bucket (affine) arrival curve.

3.1.1. Traffic Model. Assume that we want to provide guarantees to traffic flows. This requires some specific support in the network to limit the traffic sent by sources. This is done by using the concept of *arrival curve*, illustrated in Figure 6(a). Given an increasing function $\alpha(t)$ defined for $t \geq 0$, we say that an input flow $R(t)$ is constrained by $\alpha(t)$ if and only if for all $s \leq t$

$$R(t) - R(s) \leq a(t-s). \tag{12}$$

The min-plus representation of this equation is

$$R \leq R \otimes a. \tag{13}$$

We say that R has α as an arrival curve, or also that R is α-smooth.

A common arrival curve is a *leaky bucket* arrival curve (or affine arrival curve) defined by

$$\alpha(t) = \gamma_{\rho,\sigma} = \rho t + \sigma, t \geq 0, \tag{14}$$

where ρ is the rate of the flow (in units of data per time unit) and σ limits the burstiness of the flow (in units of data). Having such an arrival curve allows a source to send σ bits at once, but not more than ρ bits/second over the long run. The (σ, ρ) traffic characterization was initially proposed by Cruz [1991a], and the corresponding arrival curve is shown in Figure 6(b).

3.1.2. Network Elements Model. Service curve describes minimal service levels of network elements (router, channel, etc). It often abstracts a scheduling policy. Consider a system S and a flow through S with input and output functions R and R^*. We say that S offers to the flow a service curve β if and only if

$$\begin{aligned} &\text{—}\beta \text{ is an increasing function.} \\ &\text{—}\beta(0) = 0. \\ &\text{—}R^* \geq R \otimes \beta. \end{aligned} \tag{15}$$

In other words, for all t, there exists some $s \leq t$ such that

$$R^*(t) \geq R(s) + \beta(t-s). \tag{16}$$

Figure 7(a) shows a graphical representation of this condition. A well-defined service curve is latency-rate function $\beta_{R,T}$.

$$\beta_{R,T}(t) = R(t-T)^+ = \begin{cases} R(t-T), & t > T, \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

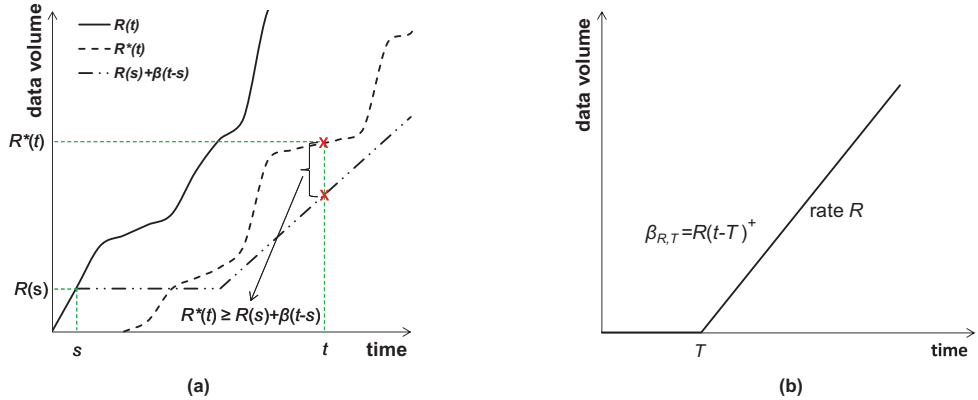


Fig. 7. (a) Definition of service curve; (b) a latency-rate service curve.

where R is the service rate and T the maximum response delay of the node [Stiliadis and Varma 1998]. Figure 7(b) shows such a service curve which is widely used to model the routers in a network.

3.1.3. Basic Bounds. Assume a flow, constrained by an arrival curve α, traverses a system that offers a service curve β.

—The backlog for all t satisfies

$$b(t) = R(t) - R^*(t) \leq \sup_{s \geq 0} \{\alpha(s) - \beta(s)\}. \tag{18}$$

—The virtual delay for all t satisfies

$$d(t) \leq \sup_{s \geq 0} \left\{ \inf_{\tau \geq 0} \{\alpha(s) \leq \beta(s+\tau)\} \right\}. \tag{19}$$

—The output flow is computed by the min-plus deconvolution operator ($\oslash$) and constrained by the curve

$$\alpha^* = \alpha \oslash \beta = \sup_{u \geq 0} \{\alpha(t+u) - \beta(u)\}. \tag{20}$$

For instance, in a system with a leaky bucket arrival curve and latency-rate service curve (shown in Figure 8), the maximum backlog, maximum delay, and output traffic characterization [Le Boudec and Thiran 2001] are

$$b_{max} = \sigma + \rho T, \tag{21}$$

$$d_{max} = T + \sigma/R, \tag{22}$$

$$\alpha^*(t) = \gamma_{\rho,\sigma+\rho T} = \rho t + \sigma + \rho T. \tag{23}$$

Using *superposition theorem*, *concatenation theorem*, and *leftover service theorem* [Jiang and Liu 2008], network calculus can be applied to a network of nodes.

Superposition. Consider the superposition of n flows R_i, $i = 1, \ldots, n$. If each flow R_i has an arrival curve α_i,the aggregate flow R has an arrival curve $\alpha = \sum_{i=1}^{n} \alpha_i$.

The superposition property implies that the aggregate of individual flows can be represented by a single aggregate flow. For instance, the aggregate flow of two flows

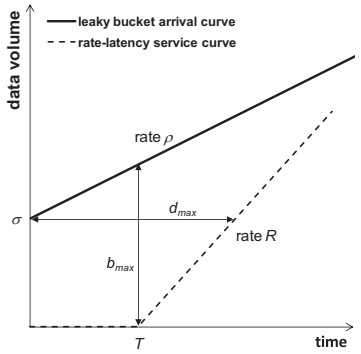


Fig. 8. Maximum backlog and delay in a system with a leaky bucket arrival curve and latency-rate service curve.

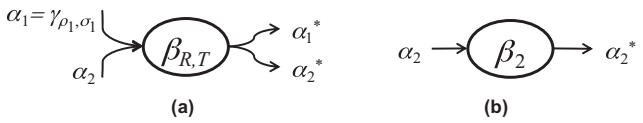


Fig. 9. (a) Server offers a service curve β to the aggregate of two flows; (b) the second flow receives leftover service curve $\beta_2 = \beta_{R-\rho_1, T+\frac{\rho_1 T+\sigma_1}{R-\rho_1}}$.

constrained by $\alpha_1 = \gamma_{\rho_1,\sigma_1}$ and $\alpha_2 = \gamma_{\rho_2,\sigma_2}$ is constrained by $\gamma_{\rho_1+\rho_2,\sigma_1+\sigma_2}$, because.

$$\alpha = \alpha_1 + \alpha_2 = \gamma_{\rho_1,\sigma_1} + \gamma_{\rho_2,\sigma_2} = (\rho_1 t + \sigma_1) + (\rho_2 t + \sigma_2) = \gamma_{\rho_1+\rho_2,\sigma_1+\sigma_2}.$$

Concatenation. Similar to traditional system theory, the concatenation of a series of servers in tandem with service curves β_i $(i = 1, \ldots, n)$ offers a service curve of $\beta = \otimes_{i=1}^{n} \beta_i = \beta_1 \otimes \beta_2 \otimes \ldots \otimes \beta_n$. As an example, consider two nodes each offering a latency-rate service curve β_{R_1,T_1} and β_{R_2,T_2}. A simple computation gives $\beta = \beta_{R_1,T_1} \otimes \beta_{R_2,T_2} = \beta_{\min(R_1,R_2),T_1+T_2}$.

Leftover Service. Consider a system offering a service curve β to the aggregate of flows R_1 and R_2. If R_1 has an arrival curve α_1, then $(\beta - \alpha_1)^+$ can be a service curve for flow R_2. For instance, assume a latency-rate server serves an aggregate of two flows as shown in Figure 9(a). If flow 1 is (σ_1, ρ_1) regulated flow, then the offered service to flow 2 is.

$$\begin{aligned}\beta_2 = (\beta - \alpha_1)^+ &= \big(R(t-T)^+ - (\rho_1 t + \sigma_1)\big)^+ = (R-\rho_1)\left(t - \left(T + \frac{\rho_1 T + \sigma_1}{R-\rho_1}\right)\right)^+ \\ &= \beta_{R-\rho_1, T+\frac{\rho_1 T+\sigma_1}{R-\rho_1}}.\end{aligned}$$

It means that the second flow is guaranteed a latency-rate service curve with parameters $R_2 = R - \rho_1$, and $T_2 = T + \frac{\rho_1 T+\sigma_1}{R-\rho_1}$, as shown in Figure 9(b).

Network calculus has been extremely successful when applied to ATM and IP networks with both differentiated and integrated services to achieve predictable performance [Le Boudec and Thiran 2001]. Recently, it has also been applied to wireless LAN [Kim and Hou 2009], sensor networks [Schmitt and Roedig 2005; She et al. 2009], and on-chip networks [Jafari et al. 2010]. Network calculus has been extended to a few directions. In the following, we briefly describe the real-time calculus and stochastic network calculus.

3.1.4. Real-Time Calculus. Chakraborty et al. [2003] proposed real-time calculus for modeling and analyzing heterogeneous systems in a compositional manner. It is a framework based on network calculus and relies on the modeling of timing properties of event streams and available resources with curves called arrival curves and service curves. In real-time calculus, an arrival curve is a function of relative time that constrains the number of events that can occur in an interval of time. For any sliding window of time of length Δ, the pair of arrival curves (α^l, α^u) gives the lower bound $\alpha^l(\Delta)$ and upper bound $\alpha^u(\Delta)$ on the number of events. Similarly, the processing capacity of a component is specified by a service curve (β^l, β^u). The number of events that may be processed in any time interval of size Δ is at least $\beta^l(\Delta)$ and at most $\beta^u(\Delta)$. In other words, arrival curve $\alpha = (\alpha^l, \alpha^u)$ and service curve $\beta = (\beta^l, \beta^u)$ are expressed in terms of numbers of events per time interval. As an alternative representation, Altisen et al. [2010] expressed arrival and service curves in terms of length of time interval. In this case, an arrival curve is represented by a pair of curves $\xi = (\xi^l, \xi^u)$. $\xi^l(k)$ and $\xi^u(k)$ respectively provide the lower and upper bounds on the length of the time interval during which any k consecutive events could arrive. Let t_i denote the arrival time of the i$^{\text{th}}$ event; we have $\xi^l(k) \leq t_{i+k} - t_i \leq \xi^u(k)$ for all $i \geq 0$ and $k \geq 0$. Also, the processing capacity of a component is specified by a service curve $\psi = (\psi^l, \psi^u)$. The length of time to process any k consecutive events for any potential stream is at least $\psi^l(k)$ and at most $\psi^u(k)$. Actually, ξ is a pseudoinverse of α, satisfying $\xi^u(k) = min_{\Delta \geq 0}\{\Delta | \alpha^l(\Delta) \geq k\}$ and $\xi^l(k) = max_{\Delta \geq 0}\{\Delta | \alpha^u(\Delta) \leq k\}$ (same for β and ψ). Also, the length of time to process any k consecutive events for any potential stream is at least $\psi^l(k)$ and at most $\psi^u(k)$.

Based on the results from network calculus, the maximum delay experienced by an event and the maximum number of backlogged events from the stream that are waiting to be processed can be given by the following inequalities.

$$delay \leq \sup_{t \geq 0} \left\{ \inf_{\tau \geq 0} \{\alpha^u(t) \leq \beta^l(t+\tau)\} \right\}; \tag{24}$$

$$backlog \leq \sup_{t \geq 0} \{\alpha^u(t) - \beta^l(t)\}. \tag{25}$$

Furthermore, real-time calculus gives exact bounds on the output stream of a component as a function of its input stream. This result can then be used as input for the next component. An event stream entering a processing or communication resource gets processed or transmitted, thereby generating an outgoing event stream which might enter another resource. As a result, the processing capability (such as the processor or bus bandwidth) of the resource, as specified by its upper and lower service curves, gets modified.

Given an event stream which is specified by its arrival curves $\alpha = (\alpha^l, \alpha^u)$ and a resource which processes this event stream and its processing capability being specified by its service curves $\beta = (\beta^l, \beta^u)$, let $\alpha' = (\alpha^{l'}, \alpha^{u'})$ denote the outgoing arrival curve of the (processed) event stream and $\beta' = (\beta^{l'}, \beta^{u'})$ denote the remaining service curves of the resource. By generalizing ideas from network calculus, these curves can be calculated as follows [Chakraborty et al. 2003].

$$\alpha^{l'}(\Delta) = min\left\{\beta^l(\Delta), inf_{0 \leq \mu \leq \Delta}\left\{sup_{\lambda \geq 0}\left\{\alpha^l(\mu+\lambda) - \beta^u(\lambda)\right\} + \beta^l(\Delta-\mu)\right\}\right\}; \tag{26}$$

$$\alpha^{u'}(\Delta) = min\left\{\beta^u(\Delta), sup_{\lambda \geq 0}\left\{inf_{0 \leq \mu \leq \lambda+\Delta}\left\{\alpha^u(\mu) + \beta^u(\lambda+\Delta-\mu)\right\} - \beta^l(\lambda)\right\}\right\}; \tag{27}$$

$$\beta^{l'}(\Delta) = sup_{0 \leq \lambda \leq \Delta}\left\{\beta^l(\lambda) - \alpha^u(\lambda)\right\}; \tag{28}$$

$$\beta^{u'}(\Delta) = max\left\{0, inf_{\lambda \geq \Delta}\left\{\beta^u(\lambda) - \alpha^l(\lambda)\right\}\right\}. \tag{29}$$

3.1.5. Stochastic Network Calculus. Stochastic network calculus [Jiang and Liu 2008] is the probabilistic version (deterministic) network calculus. Providing deterministic service often guarantees results in low resource utilization in the network. However, in some applications, such as multimedia applications, excess delay and loss of a small amount of data can be tolerated. For such applications, providing stochastic service guarantees can give better utilization of resources in the network without jeopardizing performance. Furthermore, in some networks, such as wireless networks, the service offered by a communication channel may vary randomly over time due to channel contention and impairment. Such networks can only provide stochastic services and guarantees [Jiang and Liu 2008]. Also, several stochastic versions of arrival curves have been proposed by extending the concept of arrival curve to the stochastic case based on the traffic amount property or virtual backlog property. In contrast to deterministic arrival curves, stochastic arrival curves envelop traffic tighter, but have higher implementation complexity. An arrival process R is said to be constrained by a stochastic arrival curve $a(t)$ with bounding function $f(x)$, if for all $0 \leq s \leq t$ and $x \geq 0$ there holds

$$P\left\{sup_{0\leq s\leq t}\left\{R(t) - R(s) - a(t-s)\right\} > x\right\} \leq f(x), \tag{30}$$

where notation $P\{Z\}$ means the occurrence probability of event Z [Jiang and Liu 2008]. Also, a system S is said to provide a stochastic service curve β with bounding function $g(x)$, if for all $t \geq 0$ and $x \geq 0$ there holds [Jiang and Liu 2008]

$$P\left\{sup_{0\leq s\leq t}\left\{R \otimes \beta(s) - R^*(s)\right\} > x\right\} \leq g(x) \tag{31}$$

Similar to network calculus, the probabilistic versions of backlog and delay bounds are computed based on stochastic arrival and service curves.

3.2. Applications in NoCs

Zhang [1995] surveyed several service disciplines proposed in the literature to provide per-connection end-to-end performance guarantees in packet-switching networks. Various issues and trade-offs in designing service disciplines for guaranteed performance service are discussed, and a general framework for studying and comparing these disciplines are presented. This work gives an excellent overview of guaranteed service in networks that can be applicable to NoCs.

Network calculus can be used to estimate the worst-case flow delays and backlogs in a given system. Qian et al. [2009c] investigated per-flow flit and packet worst-case delay bounds in on-chip wormhole networks. The authors first proposed analysis models for flow control, link, and buffer sharing, and then based on these analysis models, they obtained an open-ended service analysis model capturing the combined effect of flow control, link, and buffer sharing. With the service analysis model, they computed leftover service curves for individual flows and then derived their flit and packet delay bounds.

Lu et al. [2009] defined a regulation spectrum for lossless flow regulation and used it to reduce delay and backlog bounds in SoC architectures. Based on the regulation spectrum, Jafari et al. [2010] then formulated optimization problems for minimizing total buffers and buffer variations under QoS constraints. The regulation analysis was performed for best-effort networks. Bakhouya et al. [2011] presented a methodology to analyze and evaluate on-chip interconnects in terms of performance and cost metrics, such as latency, energy consumption, and area requirements. The 2D mesh, spidergon, and WK-recursive topologies were compared using a given traffic pattern. The authors showed that WK-recursive outperforms mesh and spidergon in all considered metrics. Lu [2011] used network calculus to analyze and determine the delay and buffer bounds for TDM virtual circuits crossing synchronous clock domains.

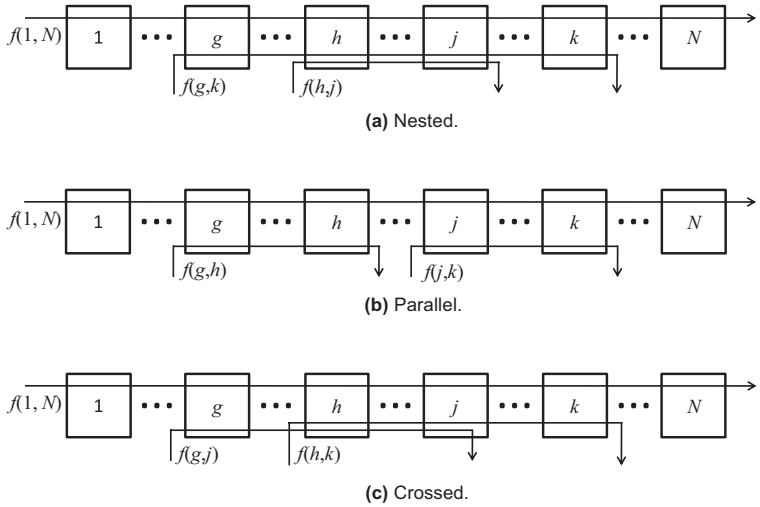


Fig. 10. Three basic contention patterns for a tagged flow [Qian et al. 2010a] © IEEE 2010.

Qian et al. [2009a] applied network calculus to NoCs in order to analyze delay and backlog bounds for self-similar traffic. The authors first showed that self-similar traffic cannot be constrained by any deterministic arrival curve. Then they proved that self-similar traffic can be constrained by leaky bucket arrival curves if an additional parameter, excess probability, is used to capture its burstiness exceeding the arrival envelope. Qian et al. [2010a] derived the worst-case delay bound for an individual flow on packet-switched best-effort NoCs. To derive the leftover service curve for the flows, the authors first constructed a *contention tree* [Lu et al. 2005] for each flow, which captures its contention with other interfering flows along its routing path, and then scanned the tree. A tagged flow directly contends with interfering flows. Also, interfering flows may contend with each other and then contend with the tagged flow again. This indirect contention may, in turn, influence the performance of the tagged flow. To decompose a complex contention scenario, they identified three primitive contention patterns. Figure 10 shows a tagged flow, $f(1,N)$, traverses a tandem of N routers from source to destination, and is multiplexed with contention flows. The contention scenarios the tagged flow may experience can be classified into three patterns: *nested*, *parallel*, and *crossed*. The authors analyzed the three scenarios and derived their basic analytical models with focus on the derivation of the service curve the tandem provides.

For nested, parallel, and crossed contention flows, the service curves of tandem $(1,N)$ for $f(1,N)$ are calculated as in Eqs. (32), (33), and (34), respectively. Interested readers can find more details in Qian et al. [2010a].

$$\beta_{(1,N)} = \left(\otimes_{i=1}^{g-1}\beta_i\right) \otimes \beta_{(1,N)}^{\{g\to k\},eq} \otimes \left(\otimes_{i=k+1}^{N}\beta_i\right); \tag{32}$$

$$\beta_{(1,N)} = \left(\otimes_{i=1}^{g-1}\beta_i\right) \otimes \beta_{(1,N)}^{\{g\to h\},eq} \otimes \left(\otimes_{i=h+1}^{j-1}\beta_i\right) \otimes \beta_{(1,N)}^{\{j\to k\},eq} \otimes \left(\otimes_{i=k+1}^{N}\beta_i\right); \tag{33}$$

$$\beta_{(1,N)} = \left(\otimes_{i=1}^{g-1}\beta_i\right) \otimes \beta_{(1,N)}^{\{g\to h-1\},eq} \otimes \beta_{(1,N)}^{\{h\to k\},eq} \otimes \left(\otimes_{i=k+1}^{N}\beta_i\right). \tag{34}$$

After obtaining the tandem service curve, the authors derived closed-form formulas to calculate the delay bound and output arrival curve based on Eqs. (19) and (20), respectively. The authors showed that the simulated delays are totally constrained by the calculated delay bounds and that the bounds are all tight. In this work, the authors

have assumed big enough buffers in routers. Bounded buffers and virtual channels were considered in Qian et al. [2009b] and Qian et al. [2010b], respectively.

Based on real-time calculus, Hamann et al. [2004] presented a framework for design space exploration and system optimization for heterogeneous SoCs and distributed systems using SymTA/S, a software tool for formal performance analysis. SymTA/S takes the hierarchical structure of the design space of heterogeneous SoCs and distributed systems into account, allowing the designer to control the exploration process. The authors showed that optimization potential through traffic shaping in complex SoCs and distributed systems is very high. Therefore, the central aspect in their proposed framework is traffic shaping. Although traffic shaping is a promising approach for implementing real time systems, it is not suitable for non-real-time and best-effort systems, since it makes the system non-work-conserving. SymTA/S allows designers to control the exploration process and provides them with insights on system-level performance dependencies. Based on this knowledge, designers can identify interesting design subspaces, worthy of being searched in-depth or even completely.

Soft real-time applications, such as a video decoder, may miss some deadlines without much of a detriment to their perceived performance. In these instances, Nelson et al. [2010] proposed a conservative simulation approach as an alternative to formal modeling for soft real-time applications. The authors introduced a hybrid simulation method which enables performance guarantees on a per-trace basis without any modeling effort. Furthermore, they evaluate an implementation of the described technique and compare it with an actual MPSoC instance implemented on an FPGA.

Network calculus emerged as a new theory for the analysis of performance bounds in network-based systems. In contrast to queueing theory, network calculus deals with worst-case analysis instead of average-case analysis. Hence, it has been a promising formalism for quality of service analysis. With network calculus, we are able to derive the worst-case bounds on maximum latency, backlog, and minimum throughput.

4. SCHEDULABILITY ANALYSIS

4.1. Overview

Schedulability analysis is a mathematical formalism for investigating the timing properties in real-time systems. It was originally proposed for analyzing the computation systems [Liu and Layland 1973; Leung and Whitehead 1982; Lehoczky et al. 1989] and then applied to communication platforms, such as multicomputers [Li and Mutka 1994] and NoCs [Shi and Burns 2008]. Usually in schedulability analysis, tasks are modeled with periodic and sporadic models. Periodic and sporadic tasks are released repeatedly. A periodic task is released at regular intervals, and a sporadic task is released at arbitrary times, but with a specified minimum time interval between releases. Given a set of periodic and sporadic tasks, their worst-case execution time, and a scheduling policy, schedulability analysis determines whether it is possible to schedule these tasks such that deadline misses never occur. The earliest results in real-time scheduling and schedulability analysis have been obtained under restrictive assumptions about the task set and the underlying architecture: the task set is composed of a fixed number of independent tasks mapped on a single processor—the tasks are periodically released, each with a fixed period, the deadlines equal the periods, and the task execution times are fixed. Later works were done under more relaxed assumptions, such as multiprocessor systems, data dependency relationships among the tasks, deadlines less than or equal to the periods, and sporadic tasks.

Usually real-time systems are equipped with a *schedulability test* [Wu et al. 2010], which determines whether each of the admitted tasks can meet its deadline. A new task will not be admitted unless it passes the schedulability test. The schedulability

test can be either *direct* or *indirect*. In a direct schedulability test, the worst-case response time of the tasks is calculated, and a task set is schedulable if and only if the worst-case response time of each task is less than or equal to its deadline. This type of test is accurate, but the computing cost in calculating the response times is very high. Audsley et al. [1993] proposed an iterative formula to compute the worst-case response time of a periodic task set. The complexity of this test is pseudo-polynomial, thus it may be unsuited for online admission control, especially in those real-time applications consisting of large task sets. Sjodin and Hansson [1998] proposed a few methods for reducing the number of iterations in computing task response times. However, the worst-case complexity of their test is still pseudo-polynomial. Indirect schedulability tests do not compute the delays, but test another performance factor of the system in order to determine the task schedulability. The *utilization-based* test is the most common indirect schedulability test which tests system resource utilization to determine the task schedulability. A new task can be admitted only if the utilization is lower than a pre-derived bound. For utilization-based schedulability test, a task set is schedulable when the utilization of the task set is lower than a pre-derived bound.

In the seminal work of Liu and Layland [1973], the problem of multiprogram scheduling on a single processor is studied. They derived a utilization bound for rate monotonic (RM) scheduling policy in which a task with a shorter period is given a higher priority than a task with a longer period. They considered sets of periodic tasks on a uniprocessor system under the assumptions that all tasks start simultaneously at time $t = 0$, deadlines are equal to periods, and tasks are independent. Under such assumptions, a set of n periodic tasks is schedulable by an RM algorithm if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1), \tag{35}$$

where C_i is the worst-case execution time (WCET) of task i, T_i is the period of task i, and n is the number of tasks. The left-hand side of the inequality represents the maximum utilization of the system, and the right-hand side represents the utilization bound, a quantity which decreases monotonically from 0.83 when $n = 2$ to $\ln 2 \approx 0.69$ as $n \to +\infty$.

In the same paper, they analyzed the set of tasks in the case that are dynamically scheduled by a runtime scheduler according to a dynamic assignment of priorities to tasks. The assignment is made according to the earlier deadline first (EDF) algorithm (i.e., the closer the task deadline, the higher the task priority). Task preemption is allowed. Under these assumptions, they proved that a task set is schedulable if and only if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1. \tag{36}$$

Consequently, it can meet all the deadlines of all periodic tasks up to full processor utilization. They also proved that in a uniprocessor system, RM and EDF are the optimal static and dynamic priority assignment algorithms, respectively. In other word, if a task set is not schedulable by RM (EDF), then it cannot be scheduled by any other static (dynamic) priority assignment. As an example, consider a processor with three tasks: their WCETs and periods are shown in Table I.

The maximum utilization of the processor is $1/5 + 3/9 + 2/10 = 0.73$, and the utilization bound for the three tasks is $U = 3(2^{1/3} - 1) = 0.78$, Since $0.73 < 0.78$, the system is surely schedulable by the RM algorithm.

In spite of the dominance of the EDF over the RM, the RM algorithm is more common in practical real-time systems, because it is easier to implement [Sha et al. 1986]. The

Table I. Time Properties of Tasks in a Real-Time System

task	WCET (C_i)	period (T_i)
task 1	1	5
task 2	3	9
task 3	2	10

typical motivations that are usually given in favor of RM state that RM introduces less runtime overhead, is easier to analyze, is more predictable in overload conditions, and causes less jitter in task execution. However, Buttazzo [2005] compared RM against EDF under several aspects, using theoretical results and simulation experiments to show that many common beliefs are either false or only restricted to specific situations.

Based on Liu and Layland's result, any periodic task set of any size will be able to meet all deadlines all of the time if the rate monotonic algorithm is used and the total utilization is not greater than 69%. It is worth mentioning that this condition is sufficient and not necessary. In practical systems, the RM algorithm can often successfully schedule task sets having total utilization higher than 69%. Based on stochastic analysis, Lehoczky et al. [1989] performed an average-case study and showed that for randomly generated task sets consisting of a large number of tasks whose periods are drawn from a uniform distribution, 88% is a good approximation to the threshold of schedulability for the RM algorithm. This implies that due to better resource utilization, the average case is substantially better than the worst case. Exact schedulability tests for RM yielding to necessary and sufficient conditions have been independently derived by Joseph and Pandya [1986], Lehoczky et al. [1989], Audsley et al. [1993], and Manabe and Aoyagi [1995].

Leung and Whitehead [1982] considered the case of deadlines smaller than periods and proved that the deadline monotonic priority assignment is optimal. Also, arbitrary deadline assignment schemes have been studied [Lehoczky et al. 1989; Lehoczky 1990; Peng and Shin 1993]. Xuan et al. [2000] and Abdelzaher et al. [2004] derived some utilization bounds for nonperiodic systems. Pop et al. [2000] proposed solutions to the schedulability analysis of hard real-time systems with control and data dependencies.

Oh and Bakker [1998], Liu [2000], and Bini et al. [2003] proposed approaches for deriving polynomial time tests with better acceptance ratios. For instance, the hyperbolic bound proposed in Bini et al. [2003] improves the acceptance ratio by a factor of $\sqrt{2}$ for large n, compared with the Liu and Layland test. According to the hyperbolic bound method, a set of periodic tasks is schedulable by RM if

$$\prod_{i=1}^{n}(\frac{C_i}{T_i} + 1) \leq 2. \tag{37}$$

The authors also extended this test in the case of resource constraints and aperiodic servers. Bini and Buttazzo [2004] derived a schedulability test for periodic task sets under an arbitrary fixed priority assignment which can be tuned through a parameter to balance complexity versus acceptance ratios, so that it can be used online to better exploit the processor based on the available computational power.

Oh and Son [1995] studied the problem of allocating a set of periodic tasks on a multiprocessor system such that tasks are scheduled to meet their deadlines on individual processors by the RM scheduling algorithm. Utilization bounds of RM in multiprocessor systems are also derived [Oh and Bakker 1998; Andersson and Jonsson 2000; Andersson et al. 2001; Funk et al. 2001; Baker 2003]. Davis and Burns [2011] surveyed hard real-time scheduling algorithms and schedulability analysis techniques for homogeneous multiprocessor systems. The survey provides a taxonomy of the different

scheduling methods and considers the various performance metrics that can be used for comparison purposes.

4.2. Application in NoCs

The SoC communication platform needs to provide different levels of service for various application components on the same network. Real-time communication has very stringent requirements—the correctness relies not only on the communication result but also the completion time bound. A data packet received by a destination too late could be useless. The worst-case acceptable time metric is defined to be the deadline of the packet. A set of real-time traffic flows over the network are termed *schedulable* if all the packets belonging to these traffic flows meet their deadlines under any arrival order of the packet set. In such a system, schedulability analysis deals with investigation of the schedulability of flows in the network. This formalism uses an iterative approach to estimate the maximum end-to-end latency of flows in a network-based system.

To support real-time communication in interconnection networks, several flow control mechanisms have been proposed to explore the priority-based packet scheduling mechanism in the literature [Li and Mutka 1994; Song et al. 1997; Balakrishnan and Ozguner 1998]. Li and Mutka [1994] proposed a flow control mechanism in which there is the same number of virtual channels as the number of priority levels, and a packet can request only a virtual channel which is numbered lower than or equal to its priority. Song et al. [1997] proposed *throttle and preempt* flow control to avoid the priority inversion problem of traditional blocking flow control in wormhole routers. Priority inversion is referred to as a situation where a higher-priority packet must wait for the transferring of a lower-priority packet. Throttle and preempt flow control prohibits low priority packets from using input buffers beyond their allowed limit, so that high-priority packets can always preempt the low-priority packets to use the channels, if necessary. Hence, this flow control does not cause priority inversion. However, the upper bound of network latency for each packet in the network is not delivered by this method.

A few works address the packet delivery guarantee problem in communication platforms and propose schedulability analysis methods to solve it. Since a link in the network is shared among several flows, it is too complex to use the utilization-based tests for determining the packet schedulability. Hence, researchers had to use direct schedulability tests and proposed a few methods for calculating the worst-case delay of packets. Kandlur et al. [1994], Sathaye and Strosnider [1994], and Li and Mutka [1996] addressed scheduling of real-time communication on direct networks. Kandlur et al. [1994] analyzed interprocessor communication for real-time systems with a direct network using store-and-forward switching. They presented a method for guaranteeing the maximum end-to-end delivery time for packets, and a schedulability test to ensure that real-time packets meet their deadlines. Several flow control methods for real-time wormhole-routed networks were proposed by Li and Mutka [1996]. These methods increase the likelihood of real-time packets meeting their deadlines, but have no guarantees on the feasibility of packets. For this reason, they are more suited to soft-deadline systems than hard-deadline systems. Hary and Ozguner [1997] presented FT1, an offline feasibility test for real-time wormhole-routed packets. This test works for any static priority assignment method. Passing FT1 is a sufficient but not a necessary condition for feasibility. All the links used to form a route for a flow are lumped as one shared resource (like a bus structure). Since they just considered the direct competitions and ignored indirect competition, their result was optimistic. Balakrishnan and Ozguner [1998] utilized the same model proposed by Hary and Ozguner [1997] and considered the indirect competitions. However, since direct and indirect contentions are considered the same, their result is pessimistic. They showed

that the computation complexity of the proposed schedulability analysis algorithm is $O(n^2)$, where n is the number of flows in the system. Kim et al. [1998] used a blocking dependency graph to express the contentions a flow may meet and derived the packet delivery upper bound. Lu et al. [2005] formulated a contention tree to take into account the direct and indirect contentions and captures concurrent use of links.

Shi and Burns [2008] proposed an offline schedulability analysis approach to discuss a real-time on-chip communication with wormhole switching and fixed priority scheduling. The authors proved that the general problem of determining the exact schedulability of real-time traffic flows over the on-chip network is NP-hard. However, they gave a determinant upper bound on the schedulability of real-time traffic flows by evaluating diverse inter-relationships among the traffic flows. They proposed a method to predict the packet network latency based on direct and indirect contention from higher priority traffic flows. Although wormhole switching with fixed priority preemption is a possible solution for real-time on-chip communication, the hardware implementation cost is expensive. Shi and Burns [2009] proposed a solution by utilizing a priority share policy to reduce the resource overhead while still achieving the hard real-time service guarantees. However, the blocking introduced by priority share policy complicates the analysis process. To address this problem, Shi and Burns [2010] proposed a per-priority basis analysis scheme which computes the total time window at each priority level instead of at each traffic flow. By checking the release instance of each flow at the corresponding priority window, they determined schedulability efficiently. Building on this static analysis, for a given set of tasks and network topology, the authors further proposed a task mapping and priority assignment algorithm in such a way that the hard time bounds are met with reduced hardware overhead.

The focus of schedulability analysis is on real-time systems, and it determines if a real-time system can meet its deadline or not. Furthermore, schedulability analysis tries to assign priority to tasks so that each task meets its deadline.

5. DATAFLOW ANALYSIS

5.1. Overview

A dataflow graph is a model-of-computation (MoC), where a number of concurrent processes communicate with each other via unbounded FIFO channels [Lee and Parks 1995]. Writing to these channels is nonblocking, while reading from these channels is blocking [Jantsch and Sander 2005]. A dataflow program is a directed graph consisting of nodes (actors) that represent communication and arcs that represent ordered sequences (streams) of data units (tokens), as illustrated in Figure 11(a). Circles represent nodes, arrows represent streams, and the dot represents a token. Dataflow graphs can be hierarchical since a node can represent a dataflow graph. The execution of a dataflow graph is a sequence of firings. During each firing, an actor consumes input tokens and produces output tokens. The number of tokens consumed and produced may vary for each firing and is defined in the firing rules of a dataflow actor. An important property of dataflow graphs is that an actor firing only depends on the availability of data. This implies that dataflow graphs are untimed, meaning that nothing is specified about points in time at which firings occur. Dataflow graphs have been shown to be very valuable in digital signal processing applications (e.g., audio and video applications) for concurrent implementation on parallel hardware. When running multiple actors on a single resource, a sequence of firings, also called a schedule, is required. For general dataflow models, it cannot be decided whether such a schedule exists because it depends on the input data.

Depending on how the consumption, production, and firing rules are specified, there exists a variety of different dataflow MoCs. They differ in their *expressiveness and suc-*

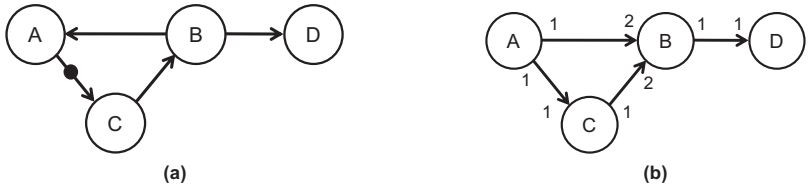


Fig. 11. (a) Dataflow network; (b) a synchronous dataflow (SDF) network.

cinctness, *analyzability,* and *implementation efficiency* [Stuijk et al. 2011]. The expressiveness and succinctness of an MoC indicates how well it can explain characteristics of the system and how compact it is. The analyzability is determined by the availability of analysis and synthesis algorithms and the runtime needed for an algorithm on a graph with a given number of nodes. The implementation efficiency of an MoC is influenced by the complexity of the runtime scheduling problem. We shall now briefly describe the most important dataflow types and then compare them with regard to expressiveness, analyzability, and implementation efficiency.

Synchronous dataflow (SDF) model is currently the most popular and widely studied dataflow model for streaming applications [Bekooij et al. 2005]. As shown in Figure 11(b), SDF puts further restrictions on the general dataflow model, since a process consumes and produces a fixed number of tokens for each firing [Lee and Messerschmitt 1987a]. With this restriction, it can be tested efficiently if a finite static schedule exists. If one exists, it can be effectively computed. The numbers on the arcs show how many tokens are produced and consumed during each firing. There exists an algorithm to construct a static periodic schedule for SDF models [Lee and Messerschmitt 1987b]. A possible schedule for the given SDF network is {A, A, C, C, B, D}. This allows for determining a static firing sequence which returns the SDF graph into its initial state. Such a firing sequence can be repeated in a loop to statically schedule an SDF graph operating on a stream of data. Much of the existing work on SDF scheduling focuses on optimizing static and dynamic schedules for parallel execution, required sizes of buffers, and end-to-end throughput. There exist many analysis algorithms for SDFs which have polynomial complexity [Stuijk et al. 2011]. Hence, it is possible to derive efficient implementations based on SDF.

Since in SDF, the data rates over different channels are not the same, it is also called a multirate dataflow model [Horstmannshoff et al. 1997]. A single-rate or homogeneous SDF (HSDF) graph is a restricted form of SDF model in which the consumption and production on each edge is a single token [Rumbaugh 1977; Dennis 1980; Lee and Messerschmitt 1987a]. A token is fireable if there is at least one token on all its incoming edges. For any HSDF, a static schedule can be easily constructed by compile-time scheduling tools. Parhi [1989] described an algorithm that transforms any SDF graph into an HSDF graph.

One problem with the SDF model is that for algorithms with variation of the data rate, the model uses more memory than the application actually needs. For example, consider the graph in Figure 12. Here, implementing the up-sample actor as an SDF actor requires a large memory to hold all of the output tokens from a single firing. This problem has been addressed by extending the SDF model to support *cyclo-static* actors, in which rate conversion actors are implemented more efficiently to execute in multiple phases [Lauwereins et al. 1994].

In the cyclo-static dataflow (CSDF) model [Lauwereins et al. 1994; Bilsen et al. 1996] the number of consumed and produced tokens by an actor varies cyclically. There are fixed numbers of phases in a cycle, and each actor produces or consumes a fixed number of tokens in each phase, but different phases may have different behavior. As shown in

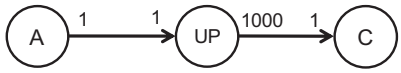


Fig. 12. An SDF graph with a large sample rate change. C's input requires excessive memory [BUCK 1994] © IEEE 1994.

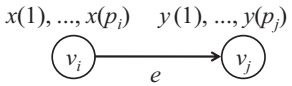


Fig. 13. Cyclo-static dataflow (adapted from [Bilsen et al. 1996] © IEEE 1996).

Figure 13, the production of actor v_i on edge e is represented as a sequence of constant integers $[x_i(1), x_i(2), \ldots, x_i(p_i)]$. The nth time that actor v_i is executed, it produces $x(1+(n\text{-}1) \bmod p_i)$ tokens on edge e. The consumption of vertex v_j is analogous. The firing rule of a cyclo-static actor v_j is evaluated as "true" for its nth firing if and only if all input FIFOs contain at least $y_i(1+(n\text{-}1) \bmod p_j)$ tokens. Lauwereins et al. [1994] showed that although the CSDF is more compact than the HSDF, it is as expressive as HSDF. Since the data rate of each channel is not fixed, analysis and scheduling of CSDF are more complex compared to SDF.

Due to some nonsynchronous and data-dependent behavior, some streaming applications cannot be expressed by SDF and CSDF [Buck 1994]. This problem can be addressed by extending the SDF model to permit some actors with data-dependent behavior. A further generalization of the SDF model is Boolean dataflow (BDF) [Buck 1993], where the numbers of consumed and produced tokens depends on the value of a token read from a dedicated control input. Since the token productions and consumptions depend on data values during runtime, a BDF network is not completely statically schedulable. However, extending the SDF model to support some dynamic actors (SWITCH and SELECT actors) while preserving static scheduling as much as possible has been studied [Lee 1991; Buck and Lee 1993]. By using SWITCH and SELECT actors, we can build conditional constructs like if-then-else and do-while loops. As shown in Figures 14(a) and 14(b), the SWITCH actor gets a control token and then copies a token from the input to the appropriate output, determined by the Boolean value of the control token. Figures 14(c) and 14(d) show that the SELECT actor gets a control token and then copies a token from the appropriate input, determined by the Boolean value of the control token, to the output. These actors are not SDF compliant, because the number of produced/consumed tokens is not fixed and depends on an input Boolean control.

The dynamic dataflow (DDF) model [Lee and Parks 1995] is a Boolean dataflow model with one additional variation: the control actors mentioned in the BDF model are able to read multiple token value, and the data actors can be fired conditionally based on the control actor read. Because of the incomplete knowledge at compile time, BDF and DDF MoCs need a runtime scheduling mechanism to determine when an actor becomes executable. Moreover, it is not always possible to predict whether a schedule with bounded buffer lengths can be constructed. Consequently, runtime scheduling and deadlock detection mechanisms are required to implement these MoCs. This makes their implementation less efficient compared to SDF and CSDF. To overcome this problem, several dataflow MoCs have been proposed in related literatures, such as Parameterized Synchronous Dataflow (PSDF) [Bhattacharya and Bhattacharyya 2001], Scenario-Aware Dataflow (SADF) [Theelen et al. 2006], Variable Rate Dataflow (VRDF) [Wiggers et al. 2008], and Variable Phased Dataflow (VPDF) [Wiggers et al.

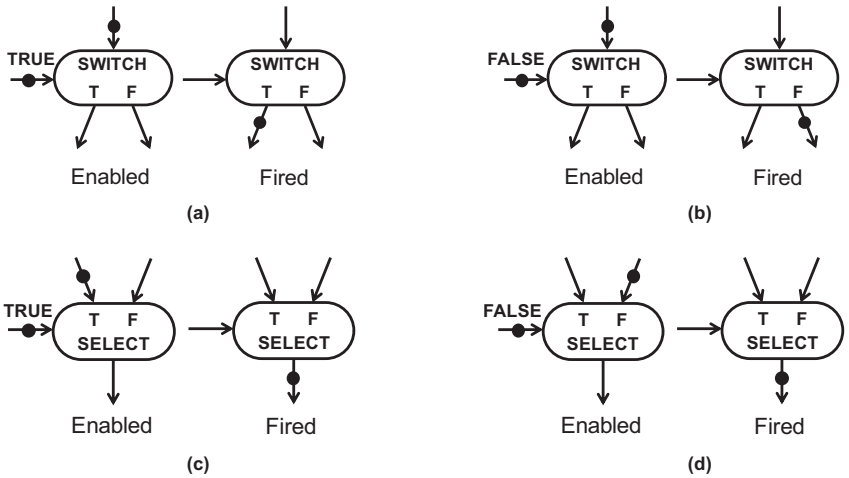


Fig. 14. The behavior of SWITCH and SELECT actors for different inputs (derived from [Buck 1994] © IEEE 1994).

2011]. These MoCs provide a trade-off between analyzability and implementation efficiency. Consequently, they can express some dynamism while allowing design-time analysis and low overhead implementation.

Bhattacharya and Bhattacharyya [2001] proposed a parameterized dataflow framework to improve the expressive power of dataflow MoCs. The parameterized dataflow framework is compatible with many of the existing data flow models, including SDF and CSDF. As an application of the parameterized modeling framework, formal semantics for parameterized SDF (PSDF) is developed in the same paper. In the PSDF MoC, channel rates are allowed to be parameterized rather than constant. Therefore, parameterized schedules and buffer sizes can be computed. Although PSDF can model data-dependent and dynamic DSP systems, options to express dynamism are limited. Variable Rate Dataflow (VRDF) is proposed to model the data-dependent communication behavior. In VRDF, data rates on channels can vary arbitrarily within a specified range. Variable Phased Dataflow (VPDF) is a generalization of both VRDF and of CSDF MoCs, where the number of repetitions of CSDF phases can vary in some finite interval. Also, the presented algorithm for computing buffer capacities under throughput constraint is a generalization of the algorithms presented Wiggers et al. [2007b, 2008] for CSDF and VRDF, respectively. Existing analyses of VRDF and VPDF are limited to computing buffer capacities that satisfy a throughput constraint. In the Scenario-Aware Dataflow (SADF) MoC, the dynamic behavior of an application is viewed as a collection of different scenarios (behaviors) [Theelen et al. 2006; Stuijk et al. 2011]. Each scenario is static and predictable in performance and resource usage. An SDF MoC models the behavior of each scenario. Since the SDF model of different scenarios may differ in all aspects, it is possible to exploit the dynamic behavior of applications to derive an implementation with limited runtime overhead.

As shown in Figure 15, Stuijk et al. [2011] compared dataflow MoCs based on the previously mentioned aspects of expressiveness and succinctness, analyzability, and implementation efficiency. Dataflow models are ordered in terms of their ability to capture dynamic behavior in a compact way in the expressiveness and succinctness axis. An overall conclusion is that expressiveness is typically traded off against analyzability and implementation efficiency.

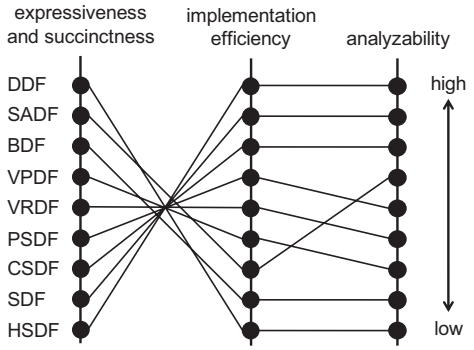


Fig. 15. Comparison of dataflow MoCs (adapted from [Stuijk et al. 2011] © IEEE 2011).

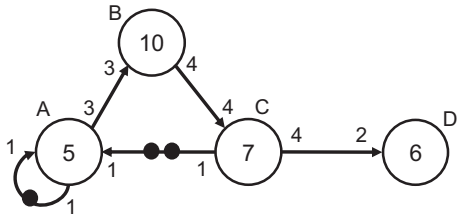
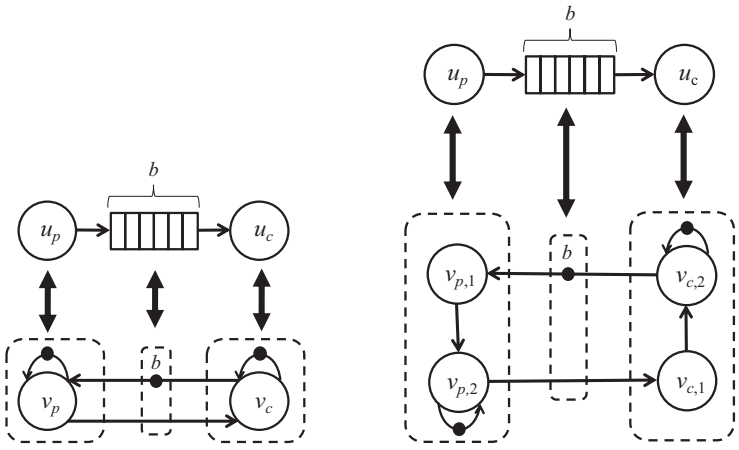


Fig. 16. An SDF graph with execution time [Kumar et al. 2008].

5.2. Applications in NoCs

The classical dataflow models are untimed. To address the timing properties of a system, a worst-case execution time can be associated with each actor [Sriram and Bhattacharyya 2009]. This extension allows us to assess the timing behavior of the NoC-based system, such as throughput and latency. A worst-case execution time is added to each actor, as shown in Figure 16. The specified number of tokens is consumed and produced within the execution time of the actor. A self-edge of an actor is used to model that the previous execution must be finished before the next execution can start. Scheduling policies can be modeled indirectly by transforming the worst-case execution time to the worst-case response time [Bekooij et al. 2005].

Throughput is an important performance indicator in streaming applications. It has been well studied in the literature on dataflow models [Dasdan and Gupta 1998; Dasdan 2004; Ghamarian et al. 2006]. All these studies focused on analysis of HSDFs and are applicable to SDFs only through a conversion to HSDF [Lee and Messerschmitt 1987a; Sriram and Bhattacharyya 2009]. Maximum cycle mean (MCM) analysis is then used to determine throughput. To determine the MCM, the maximum of the cycle means of all simple cycles in the HSDF graph needs to be determined, where the cycle mean (CM) of a cycle c is the sum of the response times of the actors on c divided by the number of initial tokens on the cycle c. The maximal attainable throughput of the graph relates to 1/MCM. Latency is another prominent performance metric. However, a little research has been done on latency. Sriram and Bhattacharyya [2009] studied the latency of HSDFs. Although it is possible to compute the latency for an SDF through conversion to an HSDF, the conversion may lead to an exponential increase in the number of nodes in the graph which makes it prohibitively expensive in predicting performance metrics [Stuijk et al. 2006]. Moreira and Bekooij [2007] presented a closed-form expression for the latency of HSDF graphs. The authors provide useful bounds on maximum latency for jobs with periodic, sporadic, and bursty sources, as well as a



(a) One actor. **(b)** Two actors.

Fig. 17. A task model with [Wiggers et al. 2007a].

technique to check latency requirements. Ghamarian et al. [2007] proposed a latency minimization technique that works directly on SDFs. This technique computes the minimal achievable latency for an SDF and provides an execution scheme that gives the minimal latency.

Bekooij et al. [2004] proposed an NoC-based multiprocessor architecture and an HSDF model of the jobs which enables reasoning about the timing behavior of the system. The NoC provides virtual point-to-point connections with a guaranteed throughput and maximal latency. The authors modeled every task by one actor with a self edge, as depicted in Figure 17(a). Wiggers et al. [2007a] have shown that latency-rate servers [Stiliadis and Varma 1998] can be included in a dataflow model by two actors, as shown in Figure 17(b). One actor models the rate, and the other models the latency.

Bekooij et al. [2005] used SDF models to derive the end-to-end temporal behavior of jobs in a real-time embedded multiprocessor system. Hansson et al. [2009] and Hansson and Goossens [2010] showed how to construct a CSDF model that conservatively models an NoC connection. Then they used the proposed dataflow model for dimensioning the buffer size in network interfaces to guarantee the system performance and showed that buffer sizes are determined with a runtime comparable to analytical methods and results comparable to exhaustive simulation. Wiggers et al. [2007b] proposed an algorithm that determines close-to-minimal buffer capacities for CSDF graphs such that the throughput requirement and constraints on maximum buffer capacities are satisfied. Also, they showed that a CSDF model can lead to reduced resource requirements compared to an SDF model.

6. NUMERICAL EXAMPLES

In this section, we consider a simple application mapped on an NoC and show how to estimate the performance metrics by using surveyed mathematical formalisms. Throughout these analyses, we assume the same topology and routing but a different flow control mechanism, since applications of formalisms differ starkly in purpose. As an example, schedulability analysis is usually used to determine the worst-case delay bound in systems with hard real-time constraints, so we assume the preemptive flow control. On the other hand, network calculus studies more general systems, so we assume the nonpreemptive flow control. Figure 18 shows the task graph and also communication

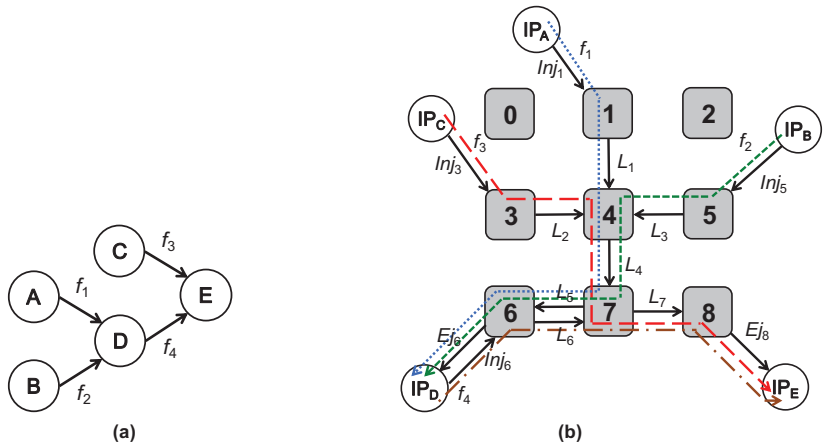


Fig. 18. (a) Task graph of an application mapped on an (b) NoC platform.

Table II. Description of Traffic Flows

flow	priority	source	destination	packet length (m_i)	packet generation rate (λ_i)	route
f_1	high	1	6	8	0.02	$Inj_1, L_1, L_4, L_5, Ej_6$
f_2	low	5	6	16	0.01	$Inj_5, L_3, L_4, L_5, Ej_6$
f_3	medium	3	8	12	0.02	$Inj_3, L_2, L_4, L_7, Ej_8$
f_4	low	6	8	8	0.03	Inj_6, L_6, L_7, Ej_8

infrastructure for the on-chip network. Links and routers are organized in a 3×3 mesh structure, as shown in Figure 18(b). The delay of links and routers are assumed one cycle and two cycles, respectively. Tasks, executing on different intellectual property (IP) modules communicate with each other by transmitting packets through the NoC.

Table II shows attributes of the traffic flows, including flow priority, source and destination of the flow, packet length in flits, and average packet generation rate in packet/cycle/IP, as well as route of the flow in the network. For instance, f_1 has the highest priority in the system and starts in IP_A, passing through injection channel 1, links 1, 4, 5, and ejection channel 6 before terminating in IP_D. All packets of this flow have the same length as the eight flits, and the average packet generation rate is 0.02 packet/cycle. In other words, on average, every 50 cycles, a packet is generated in IP_A.

6.1. Queueing Theory

In this section, the queueing theory-based analytical model proposed by Kiasari et al. [2012] is used to estimate the average latency of flows in Figure 18(b). We described this model briefly in Section 2.3. We assume that the flow control mechanism is wormhole switching and that there is one flit buffer per input channel. Channels are allocated per packet. It means that the channel is released when the whole packet has passed through the channel. Also, we assume that nodes generate packets independently of each other following a Poisson process.

The basic packet latency, d_i, happens when there is no traffic contention. It consists of two parts: the latency of head flit and the latency of body flits. Latency of head flit is determined by routing distance and router and wire delay. Once the head flit arrives at the destination, the body flits follow the header flit in a pipelined fashion. Hence, the body flit latency is a function of packet size and wire delay. For instance, according to Figure 18(b), head flit of f_1 passes through four routers and five links. Therefore,

the head flit latency is $4t_{router} + 5t_{wire} = 13$ cycles, and the body flit latency equals $(m_1 - 1)t_{wire} = 7$. As a result, the basic packet latency of f_1 is

$$d_1 = 4t_{router} + 5t_{wire} + (m_1 - 1)t_{wire} = 8 + 5 + 7 = 20 \text{ cycles}.$$

The average packet latency of f_1, D_1, is the time since the packet is created in IP_A until the last flit reaches the IP_D, including the queueing time spent at the source node ($\bar{W}_{Inj_1 \to L_1}$) and intermediate nodes ($\bar{W}_{L_1 \to L_4}$). In Figure 18(b), D_1 can be computed as

$$\bar{D}_1 = d_1 + \bar{W}_{Inj_1 \to L_1} + \bar{W}_{L_1 \to L_4} + \bar{W}_{L_4 \to L_5} + \bar{W}_{L_5 \to Ej_6}.$$

Note that $\bar{W}_{L_4 \to L_5}$ and $\bar{W}_{L_5 \to Ej_6}$ are equal to zero. The input buffer of L_4 and L_5 only have space for one head flit. Hence, if the head flit holds the input buffer of L_4, it can access channel L_5 without any waiting time. Therefore,

$$\bar{D}_1 = d_1 + \bar{W}_{Inj_1 \to L_1} + \bar{W}_{L_1 \to L_4}.$$

To estimate the $\bar{W}_{i \to j}$, the first moment (average) and second moment of channels service time should be computed. The second moment of a random variable X is defined as the average of $X^2 (\overline{X^2} = \sum_{i=1}^{k} (X_i)^2 / k)$. Determination of the channel service time moments starts at the ejection channels and works in the reverse order of routing towards to the source of the packet. It means that to compute $\bar{D}_1$, we should compute the service time of Ej_6, L_5, L_4, and L_1 ($\bar{s}_{Ej_6}, \bar{s}_{L_5}, \bar{s}_{L_4}, \bar{s}_{L_1}$, respectively). Since the delay of all channels is considered one cycle, an ejection channel offers a service time of m_i cycles to a packet of length m_i flits. According to Figure 18(b) and Table II, Ej_6 serves flows 1 and 2 with lengths of 8 and 16 flits and rates of 0.02 and 0.01 packet/cycle, respectively. Therefore, the average service time of ejection channel 6 is

$$\bar{s}_{Ej_6} = \frac{0.02}{0.03} \times 8 + \frac{0.01}{0.03} \times 16 = 10.67.$$

The waiting time for a channel closer to the destination (ejection channel) can be thought of as adding to the service time of channels farther from the destination. In other words,

$$\begin{aligned} \bar{s}_{L_5} &= \bar{s}_{Ej_6} + \bar{W}_{L_5 \to Ej_6}; \\ \bar{s}_{L_4} &= \frac{3}{5}(\bar{s}_{L_5} + \bar{W}_{L_4 \to L_5}) + \frac{2}{5}(\bar{s}_{L_7} + \bar{W}_{L_4 \to L_7}); \\ \bar{s}_{L_1} &= \bar{s}_{L_4} + \bar{W}_{L_1 \to L_4}. \end{aligned}$$

As stated before, $\bar{W}_{L_5 \to Ej_6} = 0$. Hence, $\bar{s}_{L_5} = \bar{s}_{Ej_6} = 10.67$. To compute the $\bar{s}_{L_4}$, we have to compute $\bar{W}_{L_4 \to L_7}$ in advance. The first and second moments of service time of L_7 can be given by

$$\begin{aligned} \bar{s}_{L_7} &= \bar{s}_{Ej_8} = \frac{2}{5} m_3 + \frac{3}{5} m_4 = 9.6; \\ \overline{s^2_{L_7}} &= \frac{2}{5}(m_3)^2 + \frac{3}{5}(m_4)^2 = 96. \end{aligned}$$

After computing the moments of service time, the service rate and squared coefficient of variation (SCV) of the service time of L_7 are computed.

$$\begin{aligned} \mu_{L_7} &= 1/\bar{s}_{L_7} = 0.1042; \\ C^2_{s_{L_7}} &= \overline{s^2_{L_7}}/(\bar{s}_{L_7})^2 - 1. = 0.0417. \end{aligned}$$

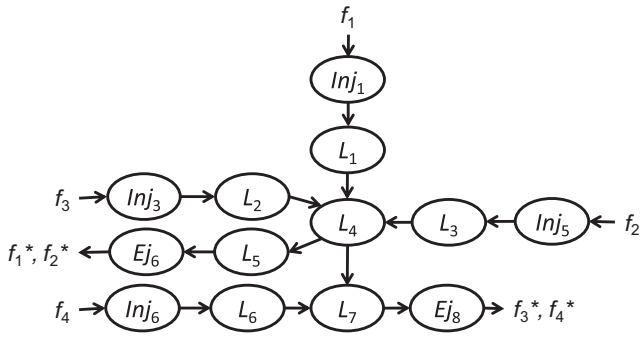


Fig. 19. Network calculus model of the system in Figure 18(b).

Now we are able to compute the waiting time for channel L_7. Flows f_3 and f_4 compete to access L_7, while f_3 has higher priority than f_4.

$$\bar{W}_{L_4 \to L_7} = \frac{1}{2}\left(C_A^2 + C_{s_{L_7}}^2\right)\frac{\lambda_{L_7}}{\mu_{L_7}^2} = 2.4.$$

C_A^2 is the SCV of the arrival process, and for the Poisson process, it equals 1. Similarly, for L_4, we can write the following.

$$\begin{aligned}
\bar{s}_{L_4} &= \frac{3}{5}(\bar{s}_{L_5} + \bar{W}_{L_4 \to L_5}) + \frac{2}{5}(\bar{s}_{L_7} + \bar{W}_{L_4 \to L_7}) = 11.2;\\
\overline{s_{L_4}^2} &= \frac{6}{8}(\bar{s}_{L_5} + \bar{W}_{L_4 \to L_5})^2 + \frac{2}{8}(\bar{s}_{L_7} + \bar{W}_{L_4 \to L_7})^2 = 125.9;\\
\mu_{L_4} &= 1/\bar{s}_{L_4} = 0.0893;\\
C_{s_{L_4}}^2 &= \overline{s_{L_4}^2}/(\bar{s}_{L_4})^2 - 1 = 0.0037.
\end{aligned}$$

Waiting time for L_4 is given by

$$\bar{W}_{L_1 \to L_4} = \frac{1}{2}\left(C_A^2 + C_{s_{L_4}}^2\right)\frac{\lambda_{L_4}}{\mu_{L_4}^2} = 3.1.$$

If we repeat the computation for L_1, $\bar{W}_{Inj_1 \to L_1}$ can be computed as

$$\bar{W}_{Inj_1 \to L_1} = \frac{1}{2}\left(C_A^2 + C_{s_{L_1}}^2\right)\frac{\lambda_{L_1}}{\mu_{L_1}(\mu_{L_1} - \lambda_1)} = 2.9.$$

Finally, we can write

$$\bar{D}_1 = d_1 + \bar{W}_{Inj_1 \to L_1} + \bar{W}_{L_1 \to L_4} = 26 \text{ cycles}.$$

Following the same approach, the average packet latency for other flows can be computed.

6.2. Network Calculus

In this section, we show how to apply the network calculus formalism for estimating the worst-case latency in the NoC described in Figure 18 and Table II. Figure 19 shows the system model in network calculus.

We consider the virtual cut-through switching with a nonpreemptive priority scheduling policy. It is also assumed that four leaky bucket controllers constrain the packet injection process in the system. In other words, f_i is a (σ_i, ρ_i) regulated flow

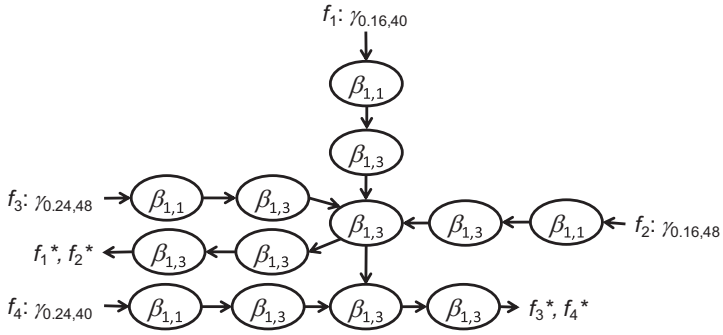


Fig. 20. System model based on the leaky bucket arrival curves and latency-rate servers.

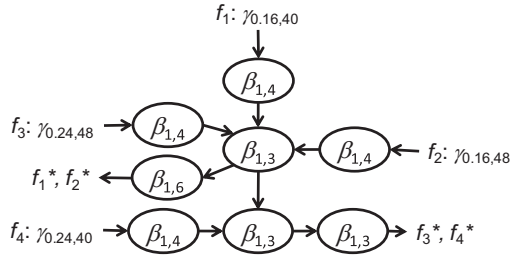


Fig. 21. Simplified system model.

which is constrained by arrival curve $\gamma_{\rho_i,\sigma_i} = \rho_i t + \sigma_i$. According to Table II, average flit injection rates are calculated as follows.

$$\begin{aligned}
\rho_1 &= 0.02 \times 8 = 0.16 \text{ flit/cycle};\\
\rho_2 &= 0.01 \times 16 = 0.16 \text{ flit/cycle};\\
\rho_3 &= 0.02 \times 12 = 0.24 \text{ flit/cycle};\\
\rho_4 &= 0.03 \times 8 = 0.24 \text{ flit/cycle}.
\end{aligned}$$

Let us further assume that the burstiness values are as follows.

$$\begin{aligned}
\sigma_1 &= 5 \text{ packets} = 40 \text{ flits};\\
\sigma_2 &= 3 \text{ packets} = 48 \text{ flits};\\
\sigma_3 &= 4 \text{ packets} = 48 \text{ flits};\\
\sigma_4 &= 5 \text{ packets} = 40 \text{ flits}.
\end{aligned}$$

Channels and routers are modeled with latency-rate servers, $\beta_{R,T}(t) = R(t - T)^+$. Average service rates are considered one flit/cycle, and wire and router delay are supposed to be one cycle and two cycles, respectively. Therefore, we can model the system, as shown in Figure 20.

Applying the concatenation theory, the model shown in Figure 20 can be simplified to Figure 21. As we mentioned in Section 3.1.3, the concatenation of two latency-rate servers results in a new latency-rate server.

$$\beta_{R_1,T_1} \otimes \beta_{R_2,T_2} = \beta_{\min(R_1,R_2),T_1+T_2}.$$

Figure 22 shows how to compute the leftover service curve for f_1 step by step. The node in the center of Figure 22(a) guarantees the service curve $\beta_{1,3} = t - 3$ to the aggregate of the three flows, where f_1 has the highest priority. Then f_1 is guaranteed a service

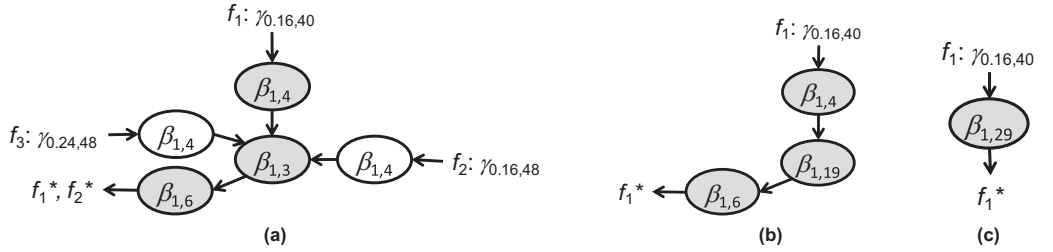


Fig. 22. Leftover service curve for flow f_1.

curve $\beta_{1,3} = (t-3) - 16$, because the maximum packet size for the lower-priority flows is 16 [Le Boudec and Thiran 2001]. Applying the concatenation theory on Figure 22(b) results in Figure 22(c).

By using the delay bound formula of Eq. (22), we can write

$$D_1 \leq T + \sigma_1/R = 29 + 40/1 = 69 \text{ cycles.}$$

The worst-case delay of other flows can be computed by the same approach.

6.3. Schedulability Analysis

In Section 4.2, we reviewed FT1 proposed by Hary and Ozguner [1997], a feasibility test for real-time wormhole-routed systems. In this section, we describe it in more detail as a sample of a schedulability analysis approach. Consider again the system described in Figure 18 and Table II. We assume that packets are injected periodically in the network. The length of time between releases of successive packets of f_i is a constant, which is called the period T_i for this flow. Using the packet generation rates in Table II, the period of each flow can be easily computed.

$$\begin{aligned} T_1 &= 1/0.02 = 50 \text{ cycles;} \\ T_2 &= 1/0.01 = 100 \text{ cycles;} \\ T_3 &= 1/0.02 = 50 \text{ cycles;} \\ T_4 &= 1/0.03 = 33 \text{ cycles.} \end{aligned}$$

It is also assumed that a router's architecture supports preemptive priority scheduling and that there are as many virtual channels per link as flows per link. Therefore, a packet cannot be blocked due to its inability to access a virtual channel. Let S_i be the set of higher-priority flows that share at least one link with f_i.

$$\begin{aligned} S_1 &= \emptyset; \\ S_2 &= \{f_1, f_3\}; \\ S_3 &= \{f_1\}; \\ S_4 &= \{f_3\}; \end{aligned}$$

A packet from f_i can only be blocked from accessing a link by higher-priority packets that share a link with f_i (i.e., any packets from $f_j \in S_i$). f_i may be blocked by more than one instance of each $f_j \in S_i$, since flows are periodic. The maximum end-to-end latency of f_i is the sum of the blocking time and d_i. d_i is the basic packet latency, we showed how to calculate it in Section 6.1.

$$\begin{aligned} d_1 &= 4t_{router} + 5t_{wire} + (m_1 - 1)t_{wire} = 8 + 5 + 7 = 20 \text{ cycles;} \\ d_2 &= 4t_{router} + 5t_{wire} + (m_2 - 1)t_{wire} = 8 + 5 + 15 = 28 \text{ cycles;} \end{aligned}$$

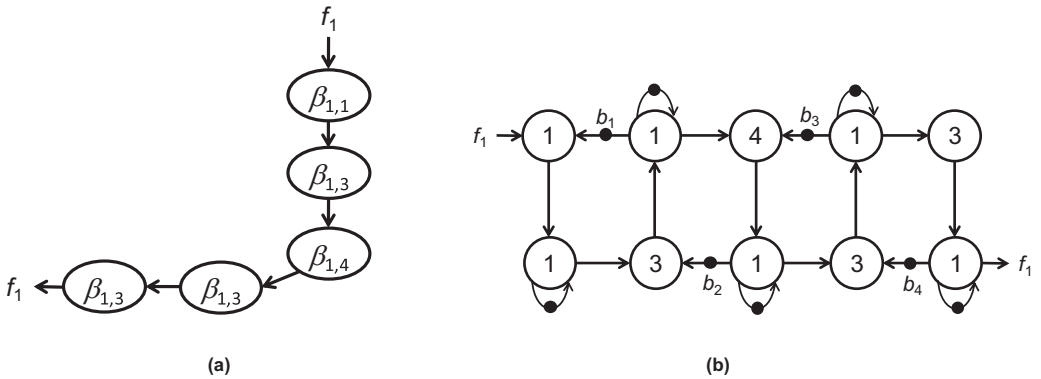


Fig. 23. (a) Latency-rate model and (b) dataflow model of the network for flow f_1.

$$d_3 = 4t_{router} + 5t_{wire} + (m_3 - 1)t_{wire} = 8 + 5 + 11 = 24 \text{ cycles};$$
$$d_4 = 3t_{router} + 4t_{wire} + (m_4 - 1)t_{wire} = 6 + 4 + 7 = 17 \text{ cycles}.$$

f_1 does not suffer any contention and receives the worst-case network latency equal to its basic latency. Therefore, the worst-case delay of a packet from f_1 is 20 cycles.

The worst-case response time of other flows, f_i, at time t, $R_i(t)$, is given as

$$R_i(t) = d_i + \sum_j d_j \lceil t/T_j \rceil,\ f_j \in S_i, \tag{38}$$

where $\lceil t/T_j \rceil$ is the maximum number of instances of higher-priority packets f_j that can occur up to time t. An iterative approach is used to solve Eq. (38), and the first iteration begins at $t = 0$. The value of t used for each iteration is the latency of the previous iteration. Eq. (38) converges when the latency of the current iteration is equal to the latency of the previous iteration. For instance, consider flow f_3, which shares at least one link with higher-priority flow f_1. The worst-case latency for flow f_3 is given by

$$R_3(t) = d_3 + d_1 \lceil t/T_1 \rceil = 24 + 20 \lceil t/50 \rceil.$$
$$t = 0\colon R_3(t) = 24;$$
$$t = 24\colon R_3(t) = 24 + 20 \lceil 20/50 \rceil = 44;$$
$$t = 44\colon R_3(t) = 24 + 20 \lceil 44/50 \rceil = 44.$$

The worst-case latency of f_3 converges at $t = 44$ cycles. Therefore, based on the FT1 test, if the deadline of f_3 is greater than 44, f_3 is schedulable, otherwise not.

6.4. Dataflow Analysis

We consider again the application and architecture in Figure 18. In this section, we assume that all packets are single-flit and that they arrive strictly periodically. We use the proposed approach by Wiggers et al. [2007a] to find the minimum required buffer for f_1, if IP_A injects a packet to the network every two cycles ($throughput(f_1) = 0.5$). Similar to the network model in Section 6.2, we can use the latency-rate servers to model the network elements which serve f_1. The result is shown in Figure 23(a). Figure 23(b) shows the HSDF model of the network for f_1.

In the dataflow model shown in Figure 23(b), we need to guarantee that the throughput of the graph equals the throughput of flow f_1. Since $throughput(f_1) = 0.5$, it is required that the MCM of the graph to be maximally 2, as we described in Section 5.2.

In other words,

$$MCM = max\left\{\frac{1}{1}, \frac{1+1+1+3}{b_1}, \frac{1+3+1+4}{b_2}, \frac{1+4+1+3}{b_3}, \frac{1+3+1+3}{b_4}\right\} \leq 2,$$

which results in

$$\frac{1+1+1+3}{b_1} \leq 2;$$
$$\frac{1+3+1+4}{b_2} \leq 2;$$
$$\frac{1+4+1+3}{b_3} \leq 2;$$
$$\frac{1+3+1+3}{b_4} \leq 2.$$

Therefore, $b_1 = 3$ packets, $b_2 = 5$ packets, $b_3 = 5$ packets, and $b_4 = 4$ packets are the minimum number of buffers to guarantee $throughput(f_1) = 0.5$.

7. BRIDGING DIFFERENT FORMALISMS

With the advances of technology, SoCs' characteristics and requirements are changing. Therefore, there has been a demand for theories dealing with heterogeneous architectures and applications encountered in such systems. The two front-runner models for performance analysis and performance guarantees of SoCs are average-case and worst-case analytical models. An open research issue is a unified analytical model for performance. To overcome the weaknesses of individual formalisms, research should try to combine the components of these formalisms. A few attempts have been made to link these formalisms together.

Based on network calculus, Schmitt [2003] derived bounds on delay and backlog per traffic class in nonpreemptive priority queueing systems. There are known results for the average behavior of such a queueing system from queueing theory. By use of numerical investigations, worst-case bounds are compared to those average-case analysis results in order to give a feel as to how conservative the worst-case bounds are. Pandit et al. [2004] analyzed the impact of network calculus bounds on queueing theory results. More precisely, they studied the impact of traffic shaping and service curve enforcement on a single M/M/1 queue. They did not analyze the system analytically, and the study was performed through simulation. The queue length distribution was compared with the original M/M/1 case, and the authors showed how the probability mass of the higher buffer states (longer queues) of the M/M/1 queue distributes over the lower buffer states (shorter queues).

Another attempt to link queueing theory and network calculus was presented by Jiang [2009]. Based on the two network calculus principles and the min-plus and the max-plus convolutions, the author derived delay bounds for the single node case and showed that they are consistent with similar bounds derived based on Lindley's equation [Lindley 1952] for G/G/1 queues. Besides attempts to link network calculus and queueing theory, there is an effort to bridge a gap between network calculus and dataflow analysis formalisms. A relationship between concepts from the network calculus and dataflow domains was described by Wiggers et al. [2007a], where it is shown that a latency-rate server, which is a concept from network calculus, can be included in the HSDF model. Figure 24 models a task u_x with one input FIFO and one output FIFO that executes on a latency-rate server with latency T_x and allocated

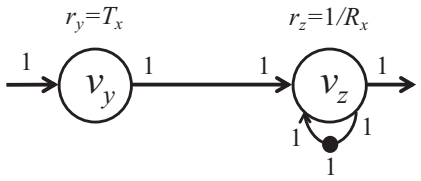


Fig. 24. A dataflow component that models a latency-rate server (derived from [Wiggers et al. 2007a]).

rate R_x. r_y and r_z are the response time of actors y and z, respectively. The resulting dataflow model provides guarantees on the temporal behavior of the implementation.

8. CONCLUSION

To summarize the discussion, we compare the presented formalisms based on the event model, node mode, and analysis output of the formalisms.

8.1. Event Model

The event model refers to the data packet representation. In queueing theory, the event model is the probability distribution of the interarrival time of packets. The interarrival times of different flows are independent and identically distributed random variables. In network calculus, an upper bound for the number of packets, called the arrival curve, models the events. An arrival curve is associated to each flow, and they are assumed to be independent. In schedulability analysis, the events are modeled with periodic and sporadic models in which a flow is represented by its minimum interarrival times. Like in queueing theory and network calculus, there is no dependency between events. Dataflow analysis models events with tokens which are produced and consumed by nodes. The production of new tokens (events) in the output ports depends on the availability of tokens in the input ports. Hence, dataflow analysis is the only formalism that captures the dependency between the events.

8.2. Node Model

Nodes are modeled based on their service time. In queueing theory, service time is specified probabilistically. The node model is the probability distribution of the router service time. In network calculus, nodes are modeled using the notion of a service curve, which is a function characterizing the minimum number of bits a node must transmit in any given time interval. In scheduling analysis, a node is modeled based on its worst-case delay and the scheduling policy. The worst-case node delay and scheduling policy represent a node in dataflow analysis. Note that scheduling policy is modeled explicitly in scheduling theory and data flow models, but implicitly in network calculus and queueing theory as more abstract node models.

Transformation of one node model to another is not always possible without loss of information or accuracy. Since queueing theory deals with average-case analysis and the three other formalisms deal with worst-case analysis, transformations between them results in information loss. Many common distributions in queueing theory (e.g., exponential distribution) assign a nonzero probability to any positive number, even regions far from the mean value. Therefore, usually it is not possible to find a worst-case bound for the service time of a node in queueing theory. Also, it is not possible to estimate the shape of the service time distribution from the worst-case service time. Node models in the other formalisms (network calculus, schedulability analysis, and dataflow analysis) can be transformed to each other without information loss. For instance, it is easy to find the worst-case node delay in dataflow analysis from a service

Table III. Input Model, Node Model, and Output of the Mathematical Formalisms

Formalism	Event model	Node model	Analysis Result
Queueing Theory	The probability distribution of the interarrival time of packets	Probability distribution of the node service time	Average packet latency, average throughput, average energy and power consumption, and average resource utilization
Network Calculus	An upper bound for the number of packet (arrival curve)	A representation of worst-case service time of the node (service curve)	Worst-case latency and backlog
Schedulability Analysis	Minimum interarrival times of periodic or sporadic packets	Worst-case node delay and scheduling policy	Worst-case latency
Dataflow Analysis	Tokens which are produced and consumed by nodes	Worst-case node delay and scheduling policy	Throughput, buffer sizing, worst-case latency

Table IV. Advantages and Disadvantages of the Formalisms

Formalism	Feature	Weakness
Queueing Theory	• Abstract model • Average-case analysis	• Hard to derive accurate models • Cannot represent flow dependencies
Network Calculus	• Abstract model • Worst-case analysis	• Hard to derive accurate models • Cannot represent flow dependencies
Schedulability Analysis	• Easy to set up event and node models	• Cannot represent flow dependencies • Limited accuracy
Dataflow Analysis	• Can express flow dependencies and flow control	• Must be used with restricted models such as SDF and CSDF

curve in network calculus, or designers can estimate the shape of the service curve from the worst-case node delay.

8.3. Analysis Results

The four considered formalisms lead to different kinds of analysis results. Since queueing theory deals with probability models, it can compute average-case performance metrics, such as average packet latency, average throughput, average energy and power consumption [Kim et al. 2005; Kiasari et al. 2008b], and average resource utilization. Network calculus computes the worst-case packet latency and maximum backlog in the system, and schedulability analysis estimates the worst-case latency of a flow to determine if it is schedulable or not. Finally, dataflow analysis determines the worst-case latency and throughput of a given system. Table III summarizes the input model, node model, and output of the studied formalisms.

8.4. Summary

Finally, we list the features and weaknesses of each formalism, which are summarized in Table IV.

All formalisms can find a closed-form relationship between system parameters and system performance metrics. However, in the case of queueing theory and network calculus, it is difficult to derive mathematical models of a given network, because they use complicated event models and node models. On the other hand, since schedulability analysis uses simpler event models, the performance model is easily extracted with less accuracy. A common problem of all models except dataflow is that they cannot

capture well dependencies between data flows. The dataflow model is the only formalism that can model the dependent flows in a system accurately. The general trade-off between abstraction and accuracy can also be observed in the comparison between these four formalisms. Queueing theory, network calculus, and schedulability analysis can be considered more abstract than dataflow. As a consequence, details such as flow control, back-pressure, and data dependencies are more difficult to capture in a natural way. Dataflow can easily model these details, but for an efficient and precise analysis, restricted models such as SDF and CSDF have to be used.

8.5. Outlook

Since each of the reviewed formalisms has different advantages and difficulties, and since they also partially differ in purpose, none of them can easily replace all others. There are definitely point problems for each formalism that are worthy further studies, but research on integrated approaches to the problems of system performance analysis is most urgent. Although each formalism can be extended in various directions, these extensions typically run into problems of complex mathematics, or they are perceived to be unnatural and cumbersome. Therefore, we believe that comprehensive frameworks that combine two or more formalisms would be most desirable. For instance, queueing theory and network calculus could be combined to offer both worst-case and average-case analysis. The result could be combined with dataflow analysis to naturally model event dependencies and lead a bridge to simulation. However, significant work exploring and understanding the relations between these models and the possible and useful transformations between them is required.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their valuable comments and suggestions.

REFERENCES

Abdelzaher, T. F., Sharma, V., and Lu, C. 2004. A utilization bound for aperiodic tasks and priority driven scheduling. *IEEE Trans. Comput. 53,* 3, 334–350.

Andersson, B. and Jonsson, J. 2000. Fixed-priority preemptive multiprocessor scheduling: To partition or not to partition. In *Proceedings of the 7th International Conference on Real-Time Systems and Applications (RTCSA'00)*. IEEE Computer Society, 337–346.

Andersson, B., Baruah, S., and Jonsson, J. 2001. Static-priority scheduling on multiprocessors. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01)*. IEEE Computer Society, 193–202.

Audsley, N. C., Burns, A., Tindell, K., and Wellings, A. 1993. Applying new scheduling theory to static priority preemptive scheduling. *Softw. Eng. J. 8,* 5, 284–292.

Altisen, K., Liu, Y., and Moy, M. 2010. Performance evaluation of components using a granularity-based interface between real-time calculus and timed automata. In *Proceedings of the 8th Workshop on Quantitative Aspects of Programming Languages (QAPL'10).* 16–23.

Baker, T. P. 2003. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS'03)*. IEEE Computer Society, 120–129.

Bakhouya, M., Suboh, S., Gaber, J., El-Ghazawi, T. A., and Niar, S. 2011. Performance evaluation and design tradeoffs of on-chip interconnect architectures. *Simul. Model. Practice Theory 19,* 6, 1496–1505.

Balakrishnan, S. and Ozguner, F. 1998. A priority-driven flow control mechanism for real-time traffic in multiprocessor networks. *IEEE Trans. Parallel Distrib. Syst. 9,* 7, 664–678.

Bekooij, M. J. G., Moriera, O., Poplavko, P., Mesman, B., Pastrnak, M., and Meerbergen, J. 2004. Predictable embedded multiprocessor system design. In *Proceedings of the International Workshop on Software and Compilers for Embedded Systems*, Lecture Notes in Computer Science, vol. 3199, Springer, 77–91.

Bekooij, M. J. G., Hoes, R., Moreira, O., Poplavko, P., Pastrnak, M., Mesman, B., Mol, J. D., Stuijk, S., Gheorghita V., and van Meerbergen J. 2005. Dataflow analysis for real-time embedded multiprocessor system design. In *Dynamic and Robust Streaming in and between Connected Consumer-Elecronic Devices*, Chapter 15. Kluwer Academic Publishers, Norwell, MA.

Bertsimas, D. and Mourtzinou, G. 1997. Transient laws of non-stationary queueing systems and their applications. *Queu. Syst. 25*, 1–4, 115–155.

Bhattacharya, B. and Bhattacharyya, S. S. 2001. Parameterized dataflow modeling for DSP systems. *IEEE Trans. Signal Process. 49,* 10, 2408–2421.

Bilsen, G., Engels, M., Lauwereins, R., and Peperstraete, J. 1996. Cyclo-static dataflow. *IEEE Trans. Signal Process. 44,* 2, 397–408.

Bini, E., Buttazzo, G. C., and Buttazzo, G. M. 2003. Rate monotonic scheduling: The hyperbolic bound. *IEEE Trans. Comp. 52,* 7, 933–942.

Bini, E. and Buttazzo, G. C. 2004. Schedulability analysis of periodic fixed priority systems. *IEEE Trans. Comput. 53,* 11, 1462–1473.

Bogdan, P. and Marculescu, R. 2007. Quantum-like effects in network-on-chip buffers behavior. In *Proceedings of the 44th Annual Design Automation Conference (DAC'07)*. ACM Press, 266–267.

Bogdan, P. and Marculescu, R. 2009. Statistical physics approaches for network-on-chip traffic characterization. In *Proceedings of the 7th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'09).* ACM Press, 461–470.

Bogdan, P. and Marculescu, R. 2010. Workload characterization and its impact on multicore platform design. In *Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'10)*. ACM Press, 231–240.

Bogdan, P. and Marculescu, R. 2011. Non-stationary traffic analysis and its implications on multicore platform design. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst. 30,* 4, 508–519.

Bolch, G., Greiner, S., de Meer, H., and Trivedi, K. S. 2006. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. 2nd Ed. John Wiley & Sons, Hoboken, NJ.

Buck, J. T. 1993. Scheduling dynamic dataflow graphs with bounded memory using the token flow model. Ph.D. dissertation, Dept. of EECS, UC Berkeley, Berkeley, CA.

Buck, J. T. and Lee, E. A. 1993. Scheduling dynamic dataflow graphs with bounded memory using the token flow model. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing: Plenary, Special, Audio, Underwater Acoustics, VLSI, Neural Networks (ICASSP'93)*, vol. I. IEEE Computer Society, 429–432.

Buck, J. T. 1994. A dynamic dataflow model suitable for efficient mixed hardware and software implementations of DSP applications. In *Proceedings of the 3rd International Workshop on Hardware/Software Co-design (CODES'94).* IEEE Computer Society, 165–172.

Buttazzo, G. C. 2005. Rate monotonic vs. EDF: Judgement day. *Real Time Syst. 29*, 1, 5–26.

Chakraborty, S., Kunzli, S., and Thiele, L. 2003. A general framework for analysing system properties in platform-based embedded system designs. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'03).* IEEE Computer Society, 10190–10195.

Chang, C.-S. 2000. *Performance Guarantees in Communication Networks*. Springer-Verlag, London, U.K.

Cheng, A.-L., Pan, Y., Yan, X.-L., and Huan, R.-H. 2011. A general communication performance evaluation model based on routing path decomposition. *J. Zhejiang Univ. - Sci. C (Comput. & Electron.) 12,* 7, 561–573.

Cruz, R. L. 1991a. A calculus for network delay, part I: Network elements in isolation. *IEEE Trans. Inf. Theory 37,* 1, 114–131.

Cruz, R. L. 1991b. A calculus for network delay, part II: Network analysis. *IEEE Trans. Inf. Theory 37,* 1, 132–141.

Dasdan, A. 2004. Experimental analysis of the fastest optimum cycle ratio and mean algorithms. *ACM Trans. Des. Autom. Electron. Syst. 9,* 4, 385–418.

Dasdan, A. and Gupta, R. 1998. Faster maximum and minimum mean cycle algorithms for system performance analysis. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst. 17,* 10, 889–899.

Davis, R. I. and Burns, A. 2011. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv. 43,* 4, article 35.

Dennis, J. B. 1980. Data flow supercomputers. *Computer 13,* 11, 48–56.

Foroutan, S., Thonnart, Y., Hersemeule, R., and Jerraya, A. 2009. Analytical computation of packet latency in a 2D-mesh NoC. In *Proceedings of the Joint IEEE North-East Workshop on Circuits and Systems and TAISA Conference*. 1–4.

Foroutan, S., Thonnart, Y., Hersemeule, R., and Jerraya, A. 2010. An analytical method for evaluating network-on-chip performance. In *Proceedings of the 13th Conference on Design, Automation and Test in Europe (DATE'10).* 1629–1632.

FUNK, S., GOOSSENS, J., AND BARUAH, S. 2001. On-line scheduling on uniform multiprocessors. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01)*. IEEE Computer Society, 183–192.

GHAMARIAN, A. H., GEILEN, M. C. W., STUIJK, S., BASTEN, T. THEELEN, B. D., MOUSAVI, M. R., MOONEN, A. J. M., AND BEKOOIJ, M. J. G. 2006. Throughput analysis of synchronous data flow graphs. In *Proceedings of the 6th International Conference on Application of Concurrency to System Design (ACSD'06)*. IEEE Computer Society, 25–36.

GHAMARIAN, A. H., STUIJK, S., BASTEN, T., GEILEN, M. C. W., AND THEELEN, B. D. 2007. Latency minimization for synchronous data flow graphs. In *Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD'07)*. IEEE Computer Society, 189–196.

GUAN, W., TSAI, W., AND BLOUGH, D. 1993. An analytical model for wormhole routing in multicomputer interconnection networks. In *Proceedings of the International Parallel Processing Symposium*. 650–654.

GUZ, Z., WALTER, I., BOLOTIN, E., CIDON, I., GINOSAR, R., AND KOLODNY, A. 2007. Network delays and link capacities in application-specific wormhole NoCs. *J. VLSI Design*, article 90941.

HAMANN, A., JERSAK, M., RICHTER, K., AND ERNST, R. 2004. Design space exploration and system optimization with SymTA/S - symbolic timing analysis for systems. In *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS'04)*. IEEE Computer Society, 469–478.

HANSSON, A., WIGGERS, M., MOONEN, A., GOOSSENS, K., AND BEKOOIJ, M. J. G. 2009. Enabling application-level performance guarantees in network-based systems on chip by applying dataflow analysis. *IET Comput. Digital Tech. 3,* 5, 398–412.

HANSSON, A. AND GOOSSENS, K. 2010. *On-chip Interconnect with Aelite: Composable and Predictable Systems*, Springer.

HARY, S. L. AND OZGUNER, F. 1997. Feasibility test for real-time communication using wormhole routing. *IEE Proc. Comput. Digital Tech. 144,* 5, 273–278.

HORSTMANNSHOFF, J., GROTKER, T., AND MEYR, H. 1997. Mapping multirate dataflow to complex RT level hardware models. In *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'97)*. 283–292.

HU, P.-C. AND KLEINROCK, L. 1997. An analytical model for wormhole routing with finite size input buffers. In *Proceedings of the 15th International Teletraffic Congress*. 549–560.

HU, J., OGRAS, U. Y., AND MARCULESCU, R. 2006. System-level buffer allocation for application-specific networks-on-chip router design. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst. 25,* 12, 2919–2933.

HUR, J. Y., GOOSSENS, K., AND MHAMDI, L. 2008. Performance analysis of soft and hard single-hop and multi-hop circuit-switched interconnects for FPGAs. In *Proceedings of the IFIP International Conference on Very Large Scale Integration*. 224–229.

JACKSON, J. R. 1957. Networks of waiting lines. *Oper. Res. 5,* 518–521.

JAFARI, F., LU, Z., JANTSCH, A., AND YAGHMAEE, M. H. 2010. Buffer optimization in network-on-chip through flow regulation. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst. 29,* 12, 1973–1986.

JANTSCH, A., AND SANDER, I. 2005. Models of computation and languages for embedded system design. *IEE Proc. Comput. Digital Tech. 152,* 2, 114–129.

JIANG, Y. AND LIU, Y. 2008. *Stochastic Network Calculus*, Springer-Verlag, Berlin.

JIANG, Y. 2009. Network calculus and queueing theory: Two sides of one coin. In *Proceedings of the 4th International Conference on Performance Evaluation Methodologies and Tools*. Article 37.

JOSEPH, M. AND PANDYA, P. 1986. Finding response times in a real-time system. *Brit. Comput. Soc. Comp. J. 29,* 5, 390–395.

KANDLUR, D. D., SHIN, K. G., AND FERRARI, D. 1994. Real-time communication in multihop networks. *IEEE Trans. Parallel Distrib. Syst. 5,* 10, 1044–1056.

KIASARI, A. E., SARBAZI-AZAD, H., AND OULD-KHAOUA, M. 2008a. An accurate mathematical performance model of adaptive routing in the star graph. *Future Gen. Comput. Syst. 24,* 6, 461–474.

KIASARI, A. E., RAHMATI, D., SARBAZI-AZAD, H., AND HESSABI, S. 2008b. A Markovian performance model for networks-on-chip. In *Proceedings of the 16th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP'08)*. 157–164.

KIASARI, A. E., SARBAZI-AZAD, H., AND HESSABI, S. 2008c. Caspian: A tunable performance model for multi-core systems. In *Proceedings of the 14th European Conference on Parallel and Distributed Computing (Euro-Par'08)*, Lecture Notes in Computer Science, vol. 5168, Springer, 100–109.

KIASARI, A. E., HESSABI, S., AND SARBAZI-AZAD, H. 2008d. PERMAP: A performance-aware mapping for application-specific SoCs. In *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors (ASAP'08)*. IEEE Computer Society, 73–78.

KIASARI, A. E., JANTSCH, A., AND LU, Z. 2010. A framework for designing congestion-aware deterministic routing. In *Proceedings of the 3rd International Workshop on Network-on-Chip Architectures (NoCArc'10)*. ACM Press, 45–50.

KIASARI, A. E., LU, Z., AND JANTSCH, A. 2012. An analytical latency model for networks-on-chip. *IEEE Trans. Very Large Scale Integ. (VLSI) Syst.* Doi: 10.1109/TVLSI.2011.2178620. To appear.

KIM, J. AND DAS, C. R. 1994. Hypercube communication delay with wormhole routing. *IEEE Trans. Comput. 43,* 7, 806–814.

KIM, B., KIM, J., HONG, S. J., AND LEE, S. 1998. A real-time communication method for wormhole switching networks. In *Proceedings of the International Conference on Parallel Processing (ICPP'98)*. IEEE Computer Society, 527–534.

KIM, J., PARK, D., NICOPOULOS, C., VIJAYKRISHNAN, N., AND DAS, C. R. 2005. Design and analysis of an NoC architecture from performance, reliability and energy perspective. In *Proceedings of the ACM Symposium on Architecture for Networking and Communications Systems (ANCS'05)*. ACM Press, 173–182.

KIM, H. AND HOU, J. C. 2009. Enabling network calculus-based simulation for TCP congestion control. *Comput. Netw. 53,* 11–24.

KLEINROCK, L. 1975. *Queueing Systems, vol. 1.*, John Wiley, Hoboken, NJ.

KRIMER, E., KESLASSY, I., KOLODNY, A., WALTER, I., AND EREZ, M. 2011. Static timing analysis for modeling QoS in networks-on-chip. *J. Parallel Distrib. Comput. 71,* 5, 687–699.

KUMAR, A., MESMAN, B., THEELEN, B. D., CORPORAAL, H., AND HA, Y. 2008. Analyzing composability of applications on MPSoC platforms. *J. Syst. Architec. Embed. Syst. Design 54,* 3–4, 369–383.

LAUWEREINS, R., WAUTERS, P., ADE, M., AND PEPERSTRAETE, J. A. 1994. Geometric parallelism and cyclo-static data flow in GRAPE-II. In *Proceedings of the International Workshop on Rapid System Prototyping*. 90–107.

LE BOUDEC J.-Y. AND THIRAN, P. 2001. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, Berlin.

LEE, E. A. AND MESSERSCHMITT, D. G. 1987a. Synchronous data flow. *Proc. IEEE 75,* 9, 1235–1245.

LEE, E. A. AND MESSERSCHMITT, D. G. 1987b. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput. 36,* 1, 24–35.

LEE, E. A. 1991. Consistency in dataflow graphs. *IEEE Trans. Parallel Distrib. Syst. 2,* 2, 223–235.

LEE, E. A. AND PARKS, T. M. 1995. Dataflow process networks. *Proc. IEEE 83,* 5, 773–799.

LEHOCZKY, J. P., SHA, L., AND DING, Y. 1989. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the IEEE Real-Time Systems Symposium*. 166–171.

LEHOCZKY, J. P. 1990. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the IEEE Real Time Systems Symposium*. 201–209.

LEUNG, J. Y. T. AND WHITEHEAD, J. 1982. On the complexity of fixed priority scheduling of periodic, real-time tasks. *Perf. Eval. 2,* 4, 237–250.

LI, J.-P. AND MUTKA, M. W. 1994. Priority based real-time communication for large scale wormhole networks. In *Proceedings of the 8th International Symposium on Parallel Processing*. IEEE Computer Society, 433–438.

LI, J.-P. AND MUTKA, M. W. 1996. Real-time virtual channel flow control. *J. Parallel Distrib. Comput. 32,* 1, 49–65.

LINDLEY, D. V. 1952. The theory of queues with a single server. *Proc. Cambridge Philoso. Soci. 48,* 2, 277–289.

LIU C. L. AND LAYLAND, J. W. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM 20,* 1, 46–61.

LIU, J. W. S. 2000. *Real-Time Systems* 1st Ed. Prentice Hall, Upper Saddle River, NJ.

LU, Z., JANTSCH, A., AND SANDER, I. 2005. Feasibility analysis of messages for on-chip networks using wormhole routing. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC'05)*. ACM Press, 960–964.

LU, Z., MILLBERG, M., JANTSCH, A., BRUCE, A., VAN DER WOLF, P., AND HENRIKSSON, T. 2009. Flow regulation for on-chip communication. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE'09)*. 578–581.

LU, Z. 2011. Cross clock-domain TDM virtual circuits for networks on chips. In *Proceedings of the 5th ACM/IEEE International Symposium on Networks-on-Chip (NoCS'11)*. ACM Press, 209–216.

MANABE, Y. AND AOYAGI, S. 1995. A feasibility decision algorithm for rate monotonic scheduling of periodic real-time tasks. In *Proceedings of the Real-Time Technology and Applications Symposium (RTAS'95)*. IEEE Computer Society, 212–218.

MIN, G. AND OULD-KHAOUA, M. 2004. A performance model for wormhole-switched interconnection networks under self-similar traffic. *IEEE Trans. Comput. 53,* 5, 601–613.

MOREIRA, O. M. AND BEKOOIJ, M. J. G. 2007. Self-timed scheduling analysis for real-time applications. *EURASIP J. Adv. Signal Proces.* Article Id: 083710.

NELSON, A., HANSSON, A., CORPORAAL, H., AND GOOSSENS, K. 2010. Conservative application-level performance analysis through simulation of MPSoCs. In *Proceedings of the 8th IEEE Workshop on Embedded Systems for Real-Time Multimedia.* 51–60.

ODONI, A. AND ROTH, E. 1983. An empirical investigation of the transient behavior of stationary queueing systems. *Oper. Res. 31,* 432–455.

OGRAS, U. Y., BOGDAN, P., AND MARCULESCU, R. 2010. An analytical approach for network-on-chip performance analysis. *IEEE Trans. Comp.-Aided Des. Integ. Cir. Syst. 29,* 12, 2001–2013.

OH, D. AND BAKKER, T. P. 1998. Utilization bounds for n-processor rate monotone scheduling with static processor assignment. *Real Time Syst. J. 15,* 1, 183-193.

OH, Y. AND SON, S. H. 1995. Allocating fixed-priority periodic tasks on multiprocessor systems. *Real-Time Syst. 9,* 3, 207–239.

PANDIT, K., SCHMITT, J., AND STEINMETZ, R. 2004. Network calculus meets queueing theory - a simulation based approach to bounded queues. In *Proceedings of the IEEE International Workshop on Quality of Service*. 114–120.

PARHI, K. K. 1989. Algorithm transformation techniques for concurrent processors. *Proc. IEEE 77,* 12, 1879–1895.

PENG, D.-T. AND SHIN, K. G. 1993. A new performance measure for scheduling independent real-time tasks. *J. Parallel Distrib. Comput. 19,* 1, 11–26.

POP, P., ELES, P., AND PENG, Z. 2000. Schedulability analysis for systems with data and control dependencies. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems (RTS'00).* IEEE Computer Society, 201–208.

QIAN, Y., LU, Z., AND DOU, W. 2009a. Applying network calculus for performance analysis of self-similar traffic in on-chip networks. In *Proceedings of the 7th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'09).* ACM Press, 453–460.

QIAN, Y., LU, Z., AND DOU, W. 2009b. Analysis of worst-case delay bounds for best-effort communication in wormhole networks on chip. In *Proceedings of the 3rd ACM/IEEE International Symposium on Networks-on-Chip (NOCS'09).* IEEE Computer Society, 44–53.

QIAN, Y., LU, Z., AND DOU, W. 2009c. Worst case flit and packet delay bounds in wormhole networks on chip. *IEICE Trans. Fundamentals Electron. Commun. Comput. Sci. Special Section on VLSI Design and CAD Algorithms E92-A,* 12, 3211–3220.

QIAN, Y., LU, Z., AND DOU, W. 2010a. Analysis of worst-case delay bounds for on-chip packet-switching networks. *IEEE Trans. Comp.-Aided Des. Integ. Cir. Sys. 29,* 5, 802–815.

QIAN, Y., LU, Z., AND DOU, W. 2010b. QoS scheduling for NoCs: Strict priority queueing versus weighted round robin. In *Proceedings of the 28th International Conference on Computer Design (ICCD'10).* 52–59.

RUMBAUGH, J. 1977. A data flow multiprocessor. *IEEE Trans. Comput. 26,* 2, 138–146.

SATHAYE, S. S. AND STROSNIDER, J. K. 1994. A real-time scheduling framework for packet-switched networks. In *Proceedings of the IEEE International Conference on Distributed Computing*. IEEE Computer Society, 182–191.

SCHERRER, A., FRABOULET, A., AND RISSET, T. 2009. Long-range dependence and on-chip processor traffic. *Microprocess. Microsyst. 33,* 1, 72–80.

SCHMITT, J. B. AND ROEDIG, U. 2005. Sensor network calculus: A framework for worst case analysis. In *Distributed Computing in Sensor Systems,* Lecture Notes in Computer Science, vol. 3560, Springer, 141–154.

SCHMITT, J. 2003. On average and worst case behaviour in non-preemptive priority queueing. In *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems*. 197–204.

SHA, L., LEHOCZKY, J. P., AND RAJKUMAR, R. 1986. Solutions for some practical problems in prioritized preemptive scheduling. In *Proceedings of the IEEE Real-Time Systems Symposium.*181–191.

SHE, H., LU, Z., JANTSCH, A., ZHOU, D., AND ZHENG, L. 2009. Analytical evaluation of retransmission schemes in wireless sensor networks. In *Proceedings of the 69th IEEE Vehicular Technology Conference (VTC'09)*.

SHI, Z. AND BURNS, A. 2008. Real-time communication analysis for on-chip networks with wormhole switching. In *Proceedings of the 2nd ACM/IEEE International Symposium on Networks-on-Chip (NOCS'08).* IEEE Computer Society, 161–170.

SHI, Z. AND BURNS, A. 2009. Real-time communication analysis with a priority share policy in on-chip networks. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE Computer Society, 3–12.

SHI, Z. AND BURNS, A. 2010. Schedulability analysis and task mapping for real-time on-chip communication. *Real-Time Syst. 46,* 3, 360–385.

SJODIN, M. AND HANSSON, H. 1998. Improved response-time analysis calculations. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'98)*. IEEE Computer Society, 399–409.

SONG, H., KWON, B., AND YOON, H. 1997. Throttle and preempt: A new flow control for real-time communications in wormhole networks. In *Proceedings of the International Conference on Parallel Processing (ICPP'97)*. IEEE Computer Society, 198–202.

SOTERIOU, V., WANG, H., AND PEH, L.-S. 2006. A statistical traffic model for on-chip interconnection networks. In *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation (MASCOTS'06)*. IEEE Computer Society, 104–116.

SRIRAM, S., AND BHATTACHARYYA, S. S. 2009. *Embedded Multiprocessors: Scheduling and Synchronization* 2nd Ed. CRC Press, Boca Raton, FL.

STILIADIS, D. AND VARMA, A. 1998. Latency-rate servers: A general model for analysis of traffic scheduling algorithms. *IEEE/ACM Trans. Netw. 6,* 5, 611–624.

STUIJK, S., GEILEN, M. C. W., AND BASTEN, T. 2006. Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. In *Proceedings of the 43rd Annual Design Automation Conference (DAC'06)*. ACM Press, 899–904.

STUIJK, S., GEILEN, M. C. W., THEELEN, B. D., AND BASTEN, T. 2011. Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications. In *Proceedings of the International Conference on Embedded Computer Systems*. 404–411.

THEELEN, B. D., GEILEN, M. C. W., BASTEN, T., VOETEN, J. P. M., GHEORGHITA, S. V., AND STUIJK, S. 2006. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *Proceedings of the International Conference on Formal Methods and Models for Co-Design*. 185–194.

VAN AS, H. R. 1986. Transient analysis of Markovian queueing systems and its application to congestion-control modeling. *IEEE J. Select. Areas Commun. 4,* 6, 891–904.

VARATKAR, G. V. AND MARCULESCU, R. 2004. On-chip traffic modeling and synthesis for MPEG-2 video applications. *IEEE Trans. Very Large Scale Integr. Syst. 12,* 1, 108–119.

WANG, J., LI, Y., AND PENG, Q. 2011. A novel analytical model for network-on-chip using semi-Markov process. *Adv. Elect. Comput. Eng. 11,* 1, 111–118.

WIGGERS, M. H., BEKOOIJ, M. J. G., AND SMIT, G. J. M. 2007a. Modeling run-time arbitration by latency-rate servers in dataflow graphs. In *Proceedings of the International Workshop on Software and Compilers for Embedded Systems*. 11–22.

WIGGERS, M. H., BEKOOIJ, M. J. G., AND SMIT, G. J. M. 2007b. Efficient computation of buffer capacities for cyclo-static dataflow graphs. In *Proceedings of the 44th Annual Design Automation Conference (DAC'07)*. ACM Press, 658–663.

WIGGERS, M. H., BEKOOIJ, M. J. G., AND SMIT, G. J. M. 2008. Buffer capacity computation for throughput constrained streaming applications with data-dependent inter-task communication. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'08)*. IEEE Computer Society, 183–194.

WIGGERS, M. H., BEKOOIJ, M. J. G., AND SMIT, G. J. M. 2011. Buffer capacity computation for throughput-constrained modal task graphs. *ACM Trans. Embed. Comput. Syst. 10,* 2, article 17.

WU, J., LIU, J.-C., AND ZHAO, W. 2010. A general framework for parameterized schedulability bound analysis of real-time systems. *IEEE Trans. Comput. 59,* 6, 776–783.

XUAN, D., BETTATI, R., CHEN, J., ZHAO, W., AND LI, C. 2000. Utilization-based admission control for real-time applications. In *Proceedings of the International Conference on Parallel Processing (ICPP'00)*. IEEE Computer Society, 251–262.

YANG, F. AND LIU, J. 2010. Transient analysis of general queueing systems via simulation-based transfer function modeling, In *Proceedings of the Winter Simulation Conference (WSC)*. IEEE Computer Society, 1110–1122.

ZHANG, H. 1995. Service disciplines for guaranteed performance service in packet-switching networks. *Proc. IEEE 83,* 10, 1374–1396.

Received April 2011; revised August 2011, January 2012; accepted April 2012