Evaluating Dynamic Task Mapping in Network Processor Runtime Systems

Xin Huang, *Student Member, IEEE*, Tilman Wolf, *Senior Member, IEEE*

***Abstract*— Modern network processor systems require the ability to adapt their processing capabilities at runtime to changes in network traffic. Traditionally, network processor applications have been optimized for a single static workload scenario, but recently several approaches for runtime adaptation have been proposed. Comparing these approaches and developing novel runtime support algorithms is difficult due to the multicore system-on-a-chip nature of network processors. In this paper, we present a model for network processors that can aid in evaluating different runtime support systems. The model considers workload characteristics of applications and network traffic using a queuing network abstraction. The accuracy of this analytical approach to modeling runtime systems is validated through simulation. We illustrate the effectiveness of our model by comparing the performance of two existing workload adaptation algorithms.**

***Index Terms*— Multiprocessor system, workload partitioning and mapping, runtime management, performance evaluation**

I. Introduction

NETWORK processors (NPs) provide programmable packet processing capabilities on modern routers. This ability to modify and customize the data path in these systems provides a vehicle for network services that go beyond simple packet forwarding. Routers can be enabled to provide QoS routing [32], advanced firewalling and intrusion detection [18], SSL termination [15], and numerous other functions that are more suitable for implementation inside the network than on end-systems. Current research on next-generation network architectures [6] proposes to further expand processing capabilities on routers and make packet processing services a first-class networking function. These trends illustrate the importance of network processors as enabling platforms.

Current network processors are implemented as system-on-a-chip multiprocessors. The use of several to dozens of simple, parallel RISC processors provides the computational power to handle Gigabit per second data rates. The simplicity and repetitiveness of network processing tasks ensures that a high level of parallelism can be achieved. One of the main challenges in using network processors is to program them in a way that fully explores the capabilities of the system. The tight interactions between processors, co-processors, on-chip and off-chip memory, and other shared components cause a lot of runtime issues that need to be considered by a programmer. Based on analytical models and simulation, workloads can be optimized for traffic scenarios by statically allocating sufficient resources.

The main problem with this approach is that network traffic changes dynamically. With changing proportions of different types of packets, processing requirements on the network processor change, too. In our results, we show that using a static allocation of processing tasks for a realistic packet trace with four types of applications achieves only 20%–60% utilization. This becomes worse if more diverse types of processing are considered as it can occur on more advanced routers (i.e., tens of different packet processing options [8]). Thus, it is necessary to consider dynamic adaptation of processor allocations during runtime.

The concept of runtime support for network processors is not new [16], [30]. As has been noted in this related work, there are considerable differences between runtime systems for network processors and conventional operating systems for workstations and server. In particular, the numerous parallel processor cores with limited instruction store make the dynamic adaptation problem complex. A runtime system needs to be able to allocate tasks to processing resources in such a way that current traffic patterns can be processed efficiently. Due to the inherent cost of reprogramming processing components, a particular configuration needs to be maintained for a certain amount of time. Thus, a given allocation needs to be somewhat "predictive" of a short window of future traffic.

Several approaches that attempt to solve the problem of allocating processing tasks to system resources have been published [16], [23], [30]. Unfortunately, they all make a variety of assumptions on the underlying system structure that complicate the comparison of these solutions. This is a major problem as fair comparison is important for enabling progress in this area of research.

In this paper, we propose a novel methodology to systematically evaluate runtime systems for network processors. Our contributions are:

- **Definition of Dynamic Workloads**. We develop a model for describing workloads for network processors in scenarios where network traffic causes changes in processing requirements. We provide examples from realistic applications and network traces and provide a mechanism for generating synthetic workloads that can be used in benchmarks.
- **Queuing Model for Analytical Evaluation of Runtime Systems**. We present a model that can determine the performance of different runtime support systems for network processors. We show how it can be used to determine a range of system performance metrics including throughput, processor utilization, and average number of packets in system.
- **Comparison of Existing Mapping Algorithms**. We illustrate the generality of our model by comparing two existing runtime support approaches that have been published previously by two different research groups.

The remainder of this paper is organized as follows. Section II discusses related work. The overall methodology is described in Section III. The main components of this methodology are discussed in the following sections: Section IV covers the workload model, Section V covers the system model, and Section VI discusses the analytical performance evaluation. In section VII, the results of the comparison of specific runtime systems using analytical evaluation are shown. Moreover, the analytical results

are compared to simulation results for validation. Section VIII summarizes and concludes this paper.

II. Related Work

Several network processors are commercially available and have been used in research and development: Intel IXP [14], Hifn PowerNP [1], EZchip NP-1 [9], AMCC np7510 [2]. These systems have been used for a broad range of networking applications, ranging from general programmable router component [26] to overlay networks [11] and content-based switching [31].

Programming of network processors is challenging due to the complex interactions between the multiple processor cores, memory, and other shared components. Different programming abstractions have been proposed by Goglin et al. [12] and Shah et al. [24] for static workload scenarios. Dynamic scenarios have been considered by Memik et al. [20] and Teja [27]. While both of these approaches provide a thin network processing operating system to simplify programming, neither provides the ability to quickly adapt multiple applications during runtime. Other scheduling algorithms have been proposed by Franklin and Datar [10]. Plishker et al. [21] have proposed mapping based on their domain specific language for network processors. Kokku et al. [16] have proposed an algorithm for adapting allocations of entire applications to processor cores to reduce power consumption. We will use this algorithm as one of the evaluation scenarios in Section VII. Wolf et al. [30] have proposed runtime support that also considers the partitioning of applications across multiple processor cores. This will be another approach that is evaluated in the Section VII.

In order to evaluate the performance of a given workload scenario on a network processor system, a number of different models have been proposed. These have typically been applied to design space evaluation by Thiele et al. [28], Crowley and Baer [7], Gries et al. [13], and Wolf and Franklin [29]. We use a less detailed approach to evaluating system performance and base our results on simulation results as proposed by Ramaswamy and Wolf [22].

Lu and Wang [19] have proposed queuing networks for modeling network processor systems. Their work addresses the performance evaluation of a single static network processing application. While their model is more detailed and considers memory accesses, it does not address the issue of dynamic workload scenarios and runtime adaptation, which is the main contribution of our work.

III. Evaluation Methodology

Our overall methodology for evaluating runtime systems for network processors is outlined in Figure 1. It can be divided into three main components:

- **Dynamic Workload Characterization**. This aspect of the methodology consists of two components, a description of the individual processing steps (“applications”) and a description of the variation in processing that depends on changes in network traffic.
- **Runtime Support System**. This component of the methodology handles the allocation of processing tasks to system resources as specified by different runtime systems. Typically, this process consists of a general system specification (e.g., number and capabilities of processors) and an algorithm that determines which application is assigned to which processor (“mapping algorithm”). The result of this process is a dynamic task allocation.
- **Performance Evaluation**. The task allocation obtained from the runtime support system needs to be evaluated for comparison between different runtime support systems and for comparison to the optimal performance. In principle, there are three methods for evaluation: analysis, simulation, and measurement. In this paper, we focus on analysis and simulation. The other method is conceptually possible with our methodology, but is not further explored in this paper (and thus shown with dashed lines).

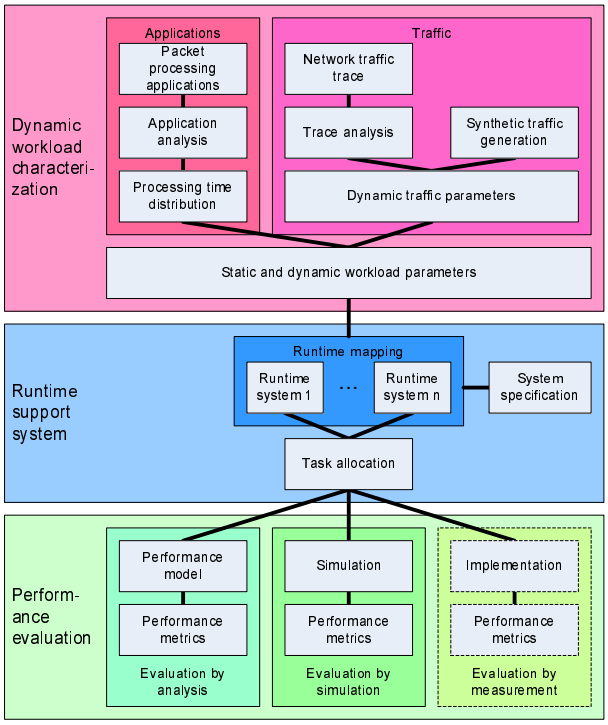


Fig. 1. Components of Evaluation Methodology and Their Interactions.

The main challenge in developing such a methodology lies in the differences in assumptions on workloads and system specifications between different existing runtime systems. We therefore focus on the following principles in our methodology:

- **Simplicity**. We do not want to make more assumptions than necessary. There are numerous details and special cases that could be considered in the evaluation process. In the current state of research on runtime support for network processors, a lot of fundamental questions about task allocation, dynamic remapping, etc. are not yet fully explored. We believe it more important to address these high-order issues before refining the methodology to consider less important special cases at the cost of generality.
- **Supports All Types of Evaluation**. As shown in Figure 1, runtime systems can be evaluated via analysis, simulation, and measurement. While we only pursue analytical evaluation and simulation in this paper, we have designed the methodology in such a way that it can also be used in the context of measurement. For example, system performance data such as throughput, average queue length, and average packet delay could be measured on a prototype system.

However, to explore fundamental design choices, analysis and simulation are more suitable.

- **Supports Hypothetical Scenarios**. Currently, there is no benchmark for NP runtime system. It is therefore important that a thorough evaluation methodology consider a broad range of scenarios, both existing (e.g., realistic network traffic) and hypothetical (e.g., extreme scenarios that may occur in the future).

In the following sections, we present the evaluation methodology that considers the above goals. First, we present a model to describe the workload and the system. Second, we show how to apply a queuing network model to estimate the performance of a particular configuration under different traffic scenarios. Then, we show results from a comparison of two specific runtime support systems that have been published previously by two different research groups. Finally, we use OPNET simulation to validate the result of the analytical queuing network models and explore the tradeoffs of these two different evaluation methods.

IV. Workload Model

The workload of a network processor system can be divided into two categories. First, there is processing workload. In most NP systems these components are not one single monolithic piece of software, but a collection of smaller "applications" (e.g., packet classification, IP forwarding, intrusion detection, etc.). Second, there is network traffic that exercises the network processor system. Depending on the constitution of traffic, different applications are used more or less. Thus, representative traffic is an essential aspect of defining the workload.

A. Applications

The combination of all applications present in the NP system is represented by a workload graph $W = (T, R)$ with vertices T, representing tasks, and edges R, representing transition probability. Each task $t \in T$ corresponds to a set of processing instructions. We do not make any assumption about the granularity of these tasks – they may be as complex as entire network processing applications or as small as individual RISC instructions. The transition probability r_{ij} between two tasks t_i and t_j gives the probability that task t_j follows task t_i in the execution of a particular packet.

While R determines the mix of the processing workload, it does not specify the packet rate that is injected into the system. We use Λ to represent that rate measured in packets per second.

B. Traffic

To introduce dynamic changes to the workload, we permit the edge matrix R and the system packet arrival rate Λ to change over time. We use the superscript t (i.e., R^t, Λ^t) to denote the state of R and Λ during the interval $[t, t + \Delta)$, where Δ is the duration of the interval. Note that the set of tasks, T, does not change over time. However, selected tasks can be activated and deactivated by adjusting R^t, which controls the amount of traffic that is sent to a given task.

C. Synthetic Traffic

It is important to obtain realistic values for the traffic characterization R^t as it defines how frequently different applications are used and how they are "chained together" when packets require multiple processing steps. As we show below, this information can be extracted from packet traces collected for network measurements. But as we have argued above, it is also important to generate scenarios that are different from today's network traffic and illustrate requirements of next-generation networks. That leads to the need for generating "synthetic" traffic scenarios.

TABLE I
Traffic Scenarios for Workload Characterization.

Scenarios	a	b	c	m	c_X
Static	any	0	0	n/a	0
Low variation	any	0	small	large	small
High variation	any	0	large	small	large
Transition	a_1 large	$b_1<0$	any	any	any
	a_2 small	$b_2>0$	any	any	any

How to generate network traffic that has realistic properties has been explored extensively in the network simulation community. Our goals are somewhat simple since we do not need to generate network traffic with all levels of detail (e.g., correct source-destination pairs, IP addresses, and flow sizes as in [17], [25]). Instead, we just need to generate scenarios that determine realistic values for R^t (i.e., the proportions of packets that traverse different paths in the graph) and Λ^t (i.e., the packet rate injected into the system).

We base the generation of a synthetic traffic trace on Holt-Winter forecasting as applied to networking in [5]. Holt-Winter forecasting decomposes a time series into three time-variant components: a baseline $a(t)$, a linear trend $b(t)$, and a seasonal trend $c(t)$. The prediction of the next element $\hat{x}(t+1)$ is

$$\hat{x}(t+1) = a(t) + b(t) + c(t+1-m), \tag{1}$$

where m is the period of the seasonal cycle. The parameters of the estimation are computed using exponential smoothing:

$$a(t) = \alpha(x(t) - c(t-m)) + (1-\alpha)(a(t-1) + b(t-1)) \tag{2}$$

$$b(t) = \beta(a(t) - a(t-1)) + (1-\beta)b(t-1) \tag{3}$$

$$c(t) = \gamma(x(t) - a(t)) + (1-\gamma)c(t-m) \tag{4}$$

where α, β, and γ are smoothing parameters for baseline, trend, and seasonal components ($0<\alpha$, β, $\gamma<1$). Note that $x(t)$ can represent number of packets or number of bytes depending on what is more suitable. In this paper, we assume $x(t)$ represents the number of packets in interval $[t, t+1)$. Also, we assume $x(t) \geq 0$ without explicit notation.

By applying the concepts of Holt-Winter forecasting for generation, we can obtain a traffic sequence:

$$x(t) = a + b \cdot t + c \cdot S(t \bmod m) + N(\sigma), \tag{5}$$

where a, b, c, and m are static Holt-Winter parameters for baseline, the linear trend, and amplitude and period of the seasonal trend, respectively. Since Holt-Winter forecasting does not specify the "shape" of the seasonal trend, we introduce $S : [0, m) \rightarrow [-1, 1]$, which allows the specification of any type of function. The shape function should be normalized to produce values in the range $[-1, 1]$ with zero mean to allow the specification of the amplitude by parameter c. Randomness in the traffic is achieved by the "noise" function N, which has a mean of zero and is

parameterized by the standard deviation σ. The distribution of N can be chosen arbitrarily and scaled appropriately with parameter σ.

We can use n generations functions $x_1(t) \ldots x_n(t)$ to describe n different classes of traffic, each with its own set of parameters $(a_i, b_i, c_i, m_i, S_i, \sigma_i)$. The total traffic Λ^t is simply the sum of all classes: $\Lambda^t = \sum_{i=1}^{n} x_i(t)$. Given that S and N have zero mean, the expected rate at a given point in time t can be estimated by the baseline and linear parameters: $E[\![x(t)]\!] = (\sum_{i=1}^{n} a_i) + (\sum_{i=1}^{n} b_i) \cdot t$.

The set of parameters $(a_i, b_i, c_i, m_i, S_i, \sigma_i)$ for each class of traffic gives us a large number of possible configurations. To ease the generation of traffic scenarios, we can make the following simplifications:

- **Identical Shape Functions**. While we allow different periods m_i for seasonal trends, we may simplify that all traffic classes follow the same seasonal shape $S = S_1 = \ldots = S_n$ (e.g., sinusoidal or step function).
- **Noise Proportional to Baseline**. The parameter for the standard deviation σ_i determines how much randomness is superimposed on the traffic. In many cases, we expect that traffic classes with a small baseline have a variation that is also small (i.e., proportional to the baseline). Thus, we can use a single parameter c_X, the coefficient of variation, to determine $\sigma_i = c_X \cdot a_i$. Note that c_X should not be confused with c_i. In cases where there is a significant linear component (i.e., large b_i), it may be more suitable to choose σ_i to be time-variant: $\sigma_i(t) = c_X \cdot (a_i + b_i \cdot t)$.

When generating workloads, we can create a number of different scenarios that are commonly observed in networking. The parameters for these scenarios are summarized in Table I. We assume that the simplifications discussed above with a sinusoidal shape function S and the use of the coefficient of variation c_X to compute σ_i. The static scenario only requires parameters for the baseline a_i (the period can take any value since $c_i = 0$). For low variation, a small seasonal parameter with a long period and a small coefficient of variation is chosen. Accordingly, high variation is achieved with a large seasonal parameter with a short period and a large coefficient of variation.

D. Example Workload

In this section, we present an example of a workload with different real and synthetic traffic scenarios. The workload graph for one traffic scenario is shown in Figure 2. There are five different packet processing tasks: packet classification, IP forwarding, intrusion detection, IPsec encryption, and IPsec decryption. The black arrows indicate the edges of the W with values that we have obtained from the first second of the measured traffic shown in Figure 3(e).

The colored arrows in Figure 2 show the paths that different packets take (i.e., the set of applications that are traversed when different packets get handled by the network processor). In our example, we consider four paths that correspond to the following four types of traffic:

- **Path 1**: packet classification and forwarding is the default handling of packet.
- **Path 2**: packet classification, intrusion detection, and forwarding is a scenario where incoming packets on an edge router are scanned for malware.

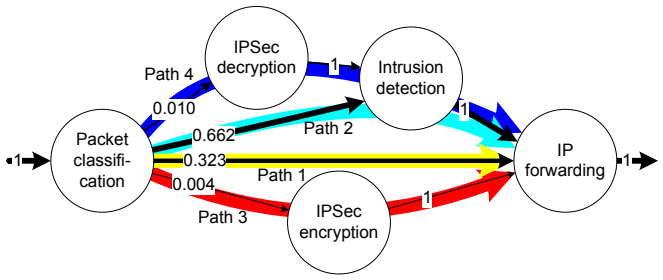
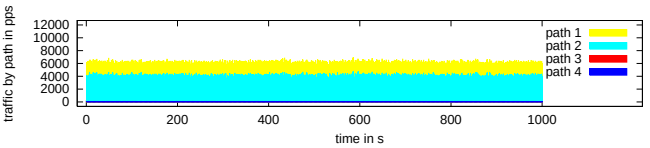
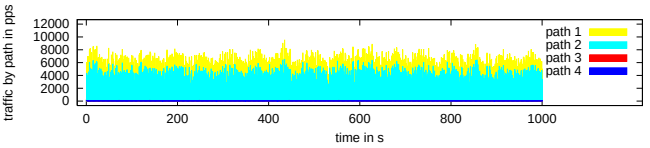


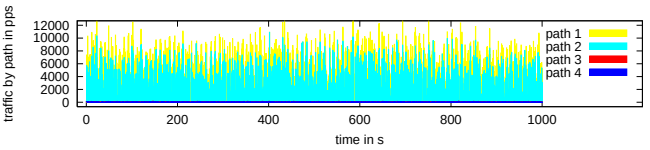
Fig. 2. Example Workload Graph with 4 Paths for Different Packet Classes.



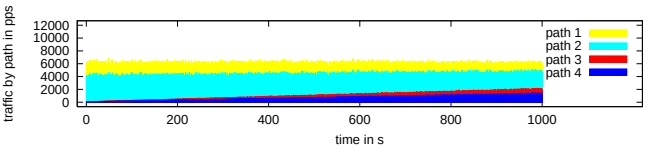
(a) Static (b_i=0, c_i=0, c_X=0.05)



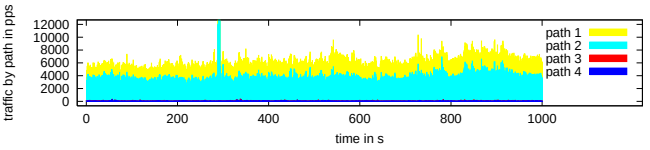
(b) Low Variation (b_i=0, c_i=.1a_i, m_i=40..70, c_X=0.2)



(c) High Variation (b_i=0, c_i=.5a_i, m=8..11, c_X=0.5)



(d) Transition (b_i=−1.33..1.33, c_i=0, c_X=0.05)



(e) Measured LAN Traffic

Fig. 3. Packets per Second for each Path for Different Traffic Scenarios. Packet counts are shown as cumulative for 1000 seconds. The baseline parameters a_i are chosen to match that of the measured LAN traffic in Figure 3(e).

- **Path 3**: packet classification, IPsec encryption, and forwarding is a scenario where some outgoing packets are tunneled via a VPN.
- **Path 4**: packet classification, IPsec decryption, intrusion detection, and forwarding is scenario where incoming VPN packets are decrypted and scanned for malware.

The weight of the edges in the workload graph change as network traffic changes. Different traffic scenarios are shown in Figure 3 as stacked bar graphs. Packets that traverse the network processor on different paths are shown with the same colors as the paths in Figure 2. Figures 3(a)–(d) show instances of the four synthetically generated workloads discussed in Table I.

Figures 3(e) shows the traffic of a LAN trace. All traffic scenarios are adjusted to have the same long-term average packet rate.

V. Runtime System Model

It is very difficult to create an accurate model for a network processor system. Many network processors contain specialized components to accelerate common functions (e.g., hash computation, crypto-processing), I/O components to access network interfaces and memory, and different types of internal interconnects for communication between processors other system units. As we have discussed above, our work aims at obtaining a general understanding of tradeoffs between different runtime systems. It is therefore inherently necessary to generalize the network processor system model to a level that allows a comparison of different implementations.

A. Network Processor Model

In this paper, we represent the network processor system as a queuing network [4]. Queuing networks are queuing systems that consist of processing elements ("servers") and queues and allow feedback loops and different classes of traffic. Queuing networks exhibit the following properties that make them particularly suitable for obtaining the performance evaluation results that we are interested in:

- **Generality**. A queuing network can represent a wide range of system configurations as we show in Section VII.
- **Suitable for Analytical Evaluation**. Very few system abstractions lend themselves for easy analytical evaluation. Queuing networks have been used extensively for analytical evaluation and are easy to simulate.
- **Representativeness**. Many network processor systems actually implement their processing tasks in form of processing steps that are interconnected with queues. For example, on the Intel IXP family of processors, scratch memory is used to queue packets between processing tasks.

We use the following notation: The network processor system is modeled by a graph $S = (P, Q)$ with vertices P ("processors") and edges Q ("queues"). The size of the queue that connects processors p_i and p_j is given by q_{ij}. If $q_{ij} = 0$, no queue exists between the processors. In our queuing network model, we can support different classes of packets (e.g., packets that traverse different paths in Figure 2). If packets from these different classes are queued in independent queues, Q can be replicated for each of the n classes: Q^c, where $c \in \{1, \ldots, n\}$.

In addition to the structure of the queuing network, it is necessary to specify the "service time" that a packet experiences when being handled by a processor. Therefore, we define the execution time of a task t_i on a processor p_j as $D(t_i, p_j)$ ("delay"). Since processing is data dependent and thus can vary, we assume D to be a random variable with cumulative distribution function of $F_{D(t_i,p_j)}(x) = P[\![D \leq x]\!]$ and probability density function $f_{D(t_i,p_j)}(x) = dF_{D(t_i,p_j)}(x)/dx$.

This delay model is a very simple approximation of the processing cost on a real system, but can be derived easily [22]. Several aspects of real NP systems are not considered (e.g., memory accesses and multithreading). Some of these aspects can be approximated in the model. For example, $f_{D(t_i,p_j)}(x)$ can be adjusted for different memory configurations and load levels. Integrating existing NP performance models [28] [29] or simulation-based approaches [12] with our queuing model can also be considered in the future.

B. Runtime Mapping

The main purpose of a runtime support system is to determine what part of the workload W^t is to be handled by what processing resources of the system S. We refer to this allocation process as "mapping." The details of how and when this mapping process is performed is the main characteristic of a runtime support system and therefore the focus of our study. It should be noted that there are several second-order issues (e.g., placement of data in memories) that also need to be handled during runtime. However, these are usually strongly influenced by the decision where to place the processing tasks. We therefore focus on the mapping of tasks to processors.

The mapping function M places tasks T onto processing elements P. This relationship is dependent on the workload at time t and thus:

$$M^t : T^t \rightarrow P. \tag{6}$$

An allocation of tasks to processing elements causes packets to be sent between processors. The parameter $\lambda_{i,j}^{c,t}$ specifies the rate of class c ($1 \leq c \leq n$) packets sent into queue $Q_{i,j}$ at time t. We can determine $\lambda_{i,j}^{c,t}$ based on the rate of traffic into the system Λ^t, workload W^t, and mapping M^t.

C. Example Runtime Systems

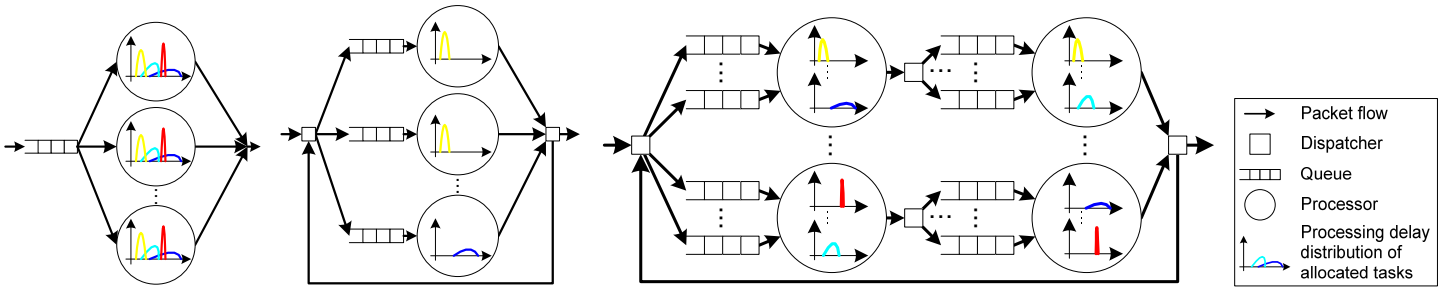
To illustrate the system model and runtime mapping process, we introduce three different mapping algorithms. These will be evaluated and compared in Section VII. The first algorithm is the ideal baseline case, which is not practical to implement. The second and third algorithm are runtime support systems that have been published by two different research groups. These systems are illustrated in Figure 4 (processors are represented as circles showing the processing delay distributions of allocated tasks).

1) Runtime System I: Ideal Allocation: This system assumes that all processors in the system can process all packets completely. The assumption that all applications are mapped to all processors is unrealistic for a real network processor system due to limitations in instruction store. However, it provides a baseline for the best possible performance that could be achieved in terms of packet delay, load balancing, and maximum data rate that can be sustained by the system. Actual runtime systems will perform less efficiently since task allocation may not match the exact current workload.

The ideal system is illustrated in Figure 4(a) In the case of the ideal system, all processor can process all types of packets. Thus, all packets are sent to the same queue, processed by any available processor, and then sent out of the system.

2) Runtime System II: Full Processor Allocation: This system is based on the runtime support system proposed by Kokku et al. [16]. The main idea of this approach is to allocate entire tasks to subsets of processors. During runtime, the number of processors allocated to each task is adapted. The decision to adjust the allocation is based on the queue length of packets waiting to be processed. The algorithm also attempts to use as few processors as possible to reduce the power consumption of the network processor.

The queuing network that corresponds to this runtime system is illustrated in Figure 4(b). We assume a processor allocation



(a) Ideal Scenario (b) Full Processor Allocation (c) Partitioned Application Allocation

Fig. 4. Instantiation of System Model for Different Runtime Support Systems.

granularity where each processor can only be assigned to process one type of task. The feedback from back to front is necessary to allow packets to traverse multiple processing steps as illustrated in Figure 2. Dispatcher nodes are used to direct packet flows accordingly.

3) Runtime System III: Partitioned Application Allocation: This system is based on the task allocation algorithm proposed by Wolf et al. [30]. In this scenario, tasks can be partitioned across multiple processors. The system is configured as a synchronous pipeline where packets move across stages in synchronous fashion. The mapping algorithm attempts to balance processing allocations across all processor to reduce overhead. The mapping is adjusted at regular intervals to adapt to changes in traffic.

This queuing system is shown in Figure 4(c). Each processor may have portions of different processing tasks installed and thus needs multiple queues for different classes of packets. Processing delay distributions in the same color and shape indicate that they are part of the same processing task. The feedback is necessary for packets that traverse multiple processing steps.

VI. Analytical Evaluation

In the previous section, we have shown how to represent a runtime system as a queuing network where the workload W^t and mapping M^t get adapted over time. As shown in Figure 1, there are different approaches to evaluation this system model. In this section, we present the basis of our analytical evaluation, which is used to derive the results shown in Section VII.

A. M/M/m–FCFS Model

We use the result of Baskett, Chandy, Muntz, and Palacios [3] (BCMP) to analyze the queuing network in our model. These BCMP networks may include several job classes, different queuing strategies, and service times with general distributions. The networks can be open, closed, or mixed. The assumptions for BCMP networks lead to four node types (using Kendall's notation): Type-1: –/M/m–FCFS, Type-2: –/G/1–PS, Type-3: –/G/∞ (IS), Type-4: –/G/1–LCFS PR. The arrival process is assumed to be one or multiple overlapping Poisson arrival streams.

The differences between these four node types lie in the queuing discipline and service time distribution. The queuing disciplines are First-Come-First-Serve (FCFS) for Type-1, Processor Sharing (PS) for Type-2, Infinite Server (IS) for Type-3, and Preemptive Resume (PR) for Type-4. The service time distribution for Type-1 is an exponential distribution and a general distribution for the other types. We have chosen Type-1 nodes for our analysis since network processors typically use FCFS queues between processing resources. An exponentially distributed service time for this node type is a reasonable assumption for analytical modeling.

For determining the equilibrium state probabilities, it has been shown that BCMP networks have product-form solution [3]. The steady-state probabilities π for states s have the following form:

$$\pi(s_1, \ldots, s_N) = \frac{1}{G(K)} d(s) \prod_{i=1}^{N} n_i(s_i), \tag{7}$$

where K is the number of jobs, $G(K)$ is a normalization constant, $d(s)$ is the product of arrival rates, and $n_i(s_i)$ is a function that depends on the type of nodes used in the BCMP queuing network. For the special case of an open BCMP network where arrival and processing rates are load independent, we can simplify the steady state probabilities:

$$\pi(k_1, \ldots, k_N) = \prod_{i=1}^{N} \pi_i(k_i), \tag{8}$$

where $\pi_i(k_i) = (1-\rho_i)\rho_i^{k_i}$, $k_i = \sum_{r=1}^{C} k_{ir}$, $\rho_i = \sum_{r=1}^{C} \rho_{ir}$, $\rho_{ir} = \Lambda_r \frac{e_{ir}}{\mu_i}$ (Type-1, $m_i = 1$), and $\overline{K}_{ir} = \frac{\rho_{ir}}{1-\rho_i}$. Details on these parameters can be found in [3] and [4] (with slightly different notation).

B. Performance Metrics

In our analysis, we focus on the following performance metrics, which are key indicators of the performance of a runtime system:

- **Processor Utilization** ρ. This metric is defined as the fraction of time that the processor is busy. Processor utilization indicates the efficiency at which the system operates.
- **Packets in System** K. This metric indicates how much the queues and processors are utilized in the system. A large value indicates that many packets are queued and that packets experience a large delay when traversing the system.

To obtain these metrics, from the above equations, we adapt the BCMP queuing network as follows.

For the Ideal Allocation system, we assume all the processing resources are identical and can process all types of traffic. Thus, the queuing network reduces to an M/G/m single server model, which we can easily evaluate.

For the Full Processor Allocation system and the Partitioned Application Allocation, we use the BCMP network introduced above with Type-1 nodes. In the Full Processor Allocation scenario, each processor can process one type of task with a given processing rate and deviation. In the Partitioned Application

Allocation system, we partition tasks into subtasks and allocate them among different processors. For simplicity, we assume that the service time distribution of each subtask is exponentially distributed (based on the number of instructions executed). Analyzing a system with subtasks from different tasks on the same processor is not easily possible with BCMP Type-1 network as they require all packet classes use the same service time distribution on a node. Therefore, we replicate each processor multiple times and reallocate subtasks such that only consecutive subtasks from a single task share a replicated processor. To obtain correct performance results, the parameter of the exponential distribution is adjusted such that the mean matches the delay that would have been encountered on the original system without replicas. This process allows us to analyze the performance of the system using the BCMP queuing network.

VII. Evaluation

Using the analytical queuing models described above, we first evaluate the performance of the two runtime systems and the ideal case described in Section V. Then, we compare and validate the analytical results against simulation results.

A. Setup

We use the following setup to evaluate the runtime systems:

- **System**: 16 processing engines, similar to an Intel IXP2800 network processor. The processing performance of each processor is assumed to be 100MIPS (corresponding to a 600MHz processor with CPI=6 due to memory access delay and other hazards). The queue lengths are assumed to be infinite to see the impact of queuing when observing the K metric.
- **Workload**: Applications are chosen as shown in Figure 2 and traffic scenarios are chosen according to Figure 3. The length of each trace scenario is 3026 seconds. To scale the overall data rate for some experiments, packet rates are scaled accordingly.
- **Other Assumptions**: The partitioned applications for Runtime System III are split into 7–15 subtasks.

B. Analytical Results

The results presented here are a representative subset of all experiments that we have performed. First, we illustrate the importance of runtime systems by exploring the performance of a static allocation mechanism. Then we show the behavior of runtime systems over time. Finally, we compare the performance of different runtime systems as data rates increase towards the maximum that a system can sustain.

1) Static Mapping: Figure 5 shows the processing demands that are encountered in the measured traffic trace. The packet rate for each application is multiplied by the average number of instructions that are executed, which is directly proportional to processing time. The left y-axis shows the demand for instructions over time as a stacked bar graph.

Assuming a static allocation (i.e., there are enough processing resources allocated to handle the worst case demand for each application), processing resources capable of processing 125.1 million instructions per second are necessary. Since this allocation assumes the worst case, most of the time the system is underutilized (as shown in Figure 5). A utilization of 100% may not even be reached if worst case demands of different applications do not coincide.

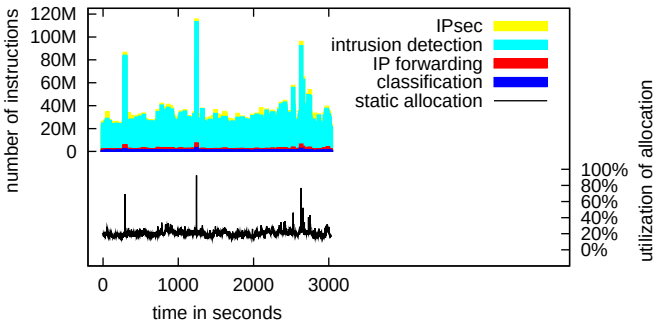


Fig. 5. Effectiveness of Static Allocation.

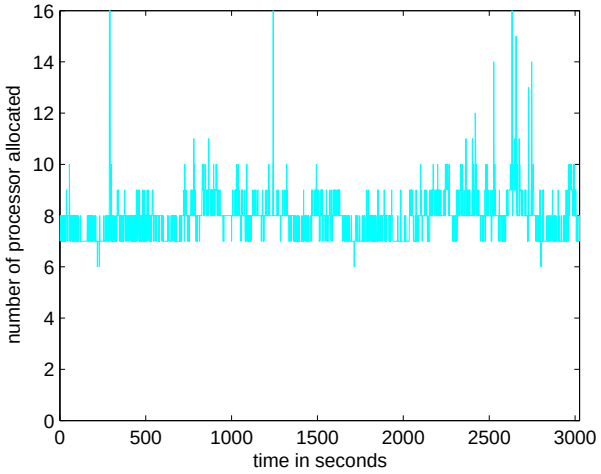


Fig. 6. Processors Allocated over Time in Full Processor Allocation System.

These results show that it is indeed important to adapt processing configurations during runtime to make full use of the available hardware (or to achieve comparable data rates with fewer processors). Especially with an increasing number of different services and applications on network processors this adaptation will become more important.

2) Performance over Time: To illustrate the behavior of a runtime system, it is easiest to consider the case of the Full Processor Allocation system. Figure 6 shows the total number of processors that are allocated over time. Note that this is different from the utilization that is achieved for a given allocation. Depending on the processing demand (see Figure 5), the allocation spikes during high-demand periods. As the demand decreases, processors are not utilized anymore. The allocation among different packet classes also changes accordingly (not shown). For the Ideal system and the Partitioned Application system all processors are in use at all times, but are only partially utilized.

This difference in utilization can be seen in Figure 7, where average utilization of the processors is shown over time. In the Partitioned Application Allocation case, applications are reallocated at fixed intervals (as proposed in [30]) of 1 second and the granularity at which traffic demands are estimated is 1%. As the figure shows, the Full Processor Allocation shows a higher utilization on the processors that are allocated to processing. The Ideal system achieves the lowest utilization because all processors

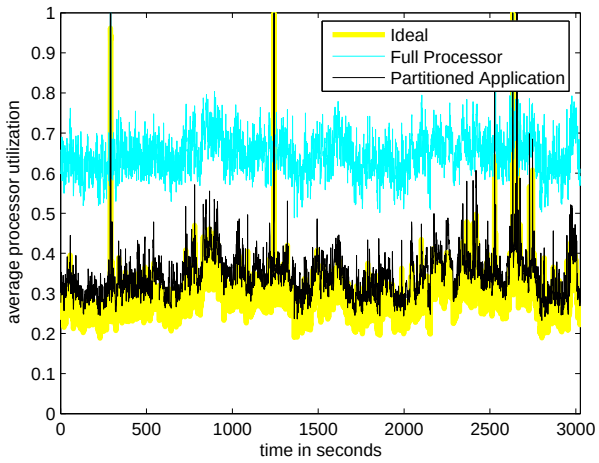


Fig. 7. Processor Utilization over Time.

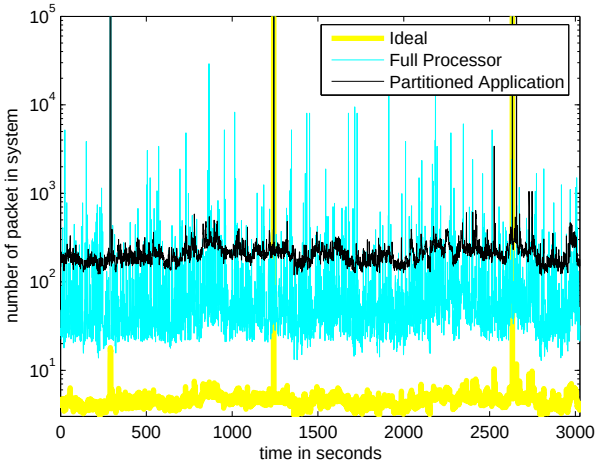


Fig. 8. Packets in System over Time.

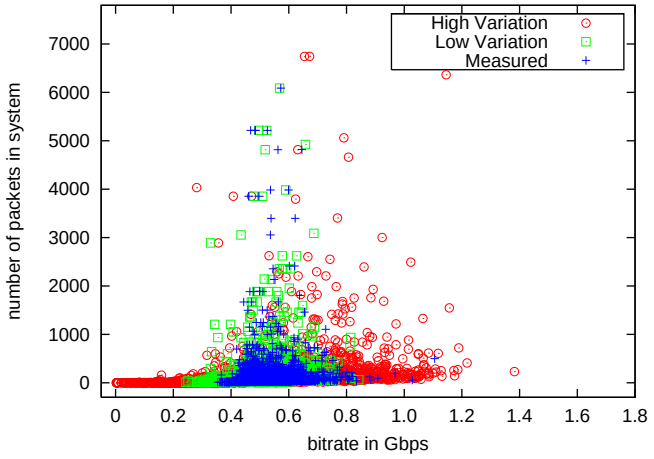


Fig. 9. Packets in System for Different Data Rates Observed in Different Traffic Scenarios.

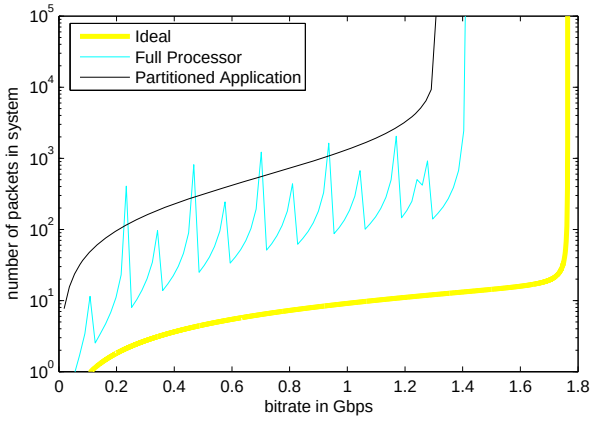


Fig. 10. Packets in System over Different Data Rates.

are utilized and the allocation allows all processor to process any type of packet. The Partitioned Application Allocation cannot equal the Ideal case due to overhead of fragmented subtasks.

The effects of the runtime system can also be seen when exploring the number of packets in the system as shown in Figure 8. The higher utilization in Full Processor Allocation causes packets to queue up. Runtime adaptation is triggered when the queue length exceeds a threshold. The average number of packets in the system for Full Processor Allocation increases and decreases with this threshold value. Although Partitioned Application Allocation has a lower processor utilization than the Full Processor Allocation, the potentially unbalanced allocation of subtasks and the associated synchronization overhead increases the number of packets that are present in the system. However, the partitioning of applications into subtasks allows for a finer granularity of processor resource allocation, which in turn leads to less variation in the number of packets in the system. In Figure 8, the standard deviation of number of packets in the system for Full Processor Allocation is 1048.6 compared to 85.4 for Partitioned Application Allocation and 0.9 for Ideal Allocation. Less variation in the number of packets in the system is generally preferable as it leads to less variation in the processing delay and thus less jitter in the network.

3) Impact of Different Traffic: The characteristics of the traffic that is processed by a system has a big impact on the overall queuing behavior. Figure 9 shows a scatter plot of bit rates at 1-second interval of traffic and the observed number of packets in the system for Full Processor Allocation. Different colors indicate data points from different traffic scenarios. The High Variation scenario generates traffic with a wide range of data rates. The adaptation cannot happen infinitely fast, and thus the system may have to queue packets. With higher variation, more packets need to be queued on average than for the Low Variation scenario. The measured scenario is also shown as reference.

4) Performance for Different Data Rates: Finally, we explore the behavior of all three runtime systems for increasing data rates. Figure 10 shows the average number of packets in the system for different data rates. The Ideal system can process packets with negligible delay up to the maximum possible data rate that can be sustained for the given system configuration and workload.

The Full Processor Allocation system shows longer queues for some data rates. This occurs when a given allocation is just barely sufficient to process the offered load. When the load increases a bit, an additional processor is allocated and the number of packets in the system drops to nearly zero.

The Partitioned Applications system shows an increasing num-

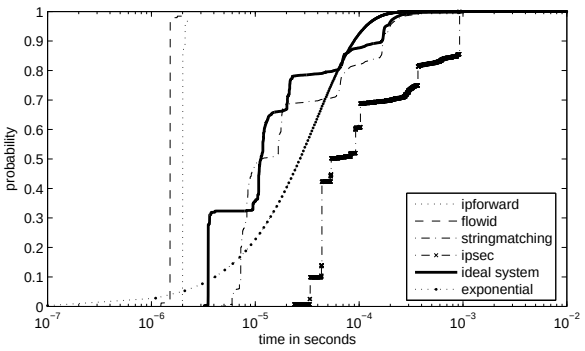


Fig. 11. Service Time CDF of All Applications, the Ideal System, and an Exponentially Distributed Random Variable.

ber of packets even for low data rates. It does not show the effects of reallocation as the Full Processor Allocation system since all processors are in use at all times. However, due to the partitioning of applications, finding a good mapping is difficult. As a result, the system is fully utilized at a lower data rate than the other two scenarios.

The point at which the queue length increases to infinity is important as it marks the maximum data rate that a particular system can sustain. For the scenario shown in Figure 10, Full Processor Allocation achieves 1.41Gbps / 1.77Gbps $= 79.6\%$ and Partitioned Application Allocation achieves 1.33Gbps / 1.77Gbps $= 75.1\%$. Of course, these numbers change for different workloads and system configurations.

C. Simulation Results

To validate the analytical models, we present performance results from simulations for each type of runtime system. We also explore the difference and tradeoffs of these two evaluation methods.

1) OPNET Simulation Setup: We use the OPNET simulator to model our queuing networks with three components: (1) Sources that generate packets of constant size (535 bytes – approximately the average packet size in the Internet) using the OPNET $simple\_source$ process model. The packet interarrival time can be chosen to be exponentially distributed (for memoryless source model) or taken from a trace file (to simulate realistic network traffic). (2) Sinks that absorb packets and collect statistics using the OPNET *sink* model. (3) A Queueing Network that implements the queuing models shown in Figure 4 based on the OPNET acb_fifo_ms processor model (each processor is assumed to process 100MIPS). The service time of each processor can be set to be constant, exponentially distributed (for memoryless service model), or taken from a random variable generation process specified through a probability mass function (for service model based on real processing time measurements). The default queue lengths are simulated to be infinite (within the limits of OPNET).

When simulating each runtime system, we can choose from several combinations of arrival and service distributions:

- M/M/m Simulation: Uses exponentially distributed packet interarrival times and processing times. This configuration matches closely to queuing models with memoryless arrival and service distributions.
- M/G/m Simulation: Uses exponentially distributed packet interarrival times and realistic processing times. The distribution of packet processing time is obtained by analyzing packets processing traces using PacketBench [22]. The cumulative distribution function for all applications of our workload (see Figure 3(e)) is shown in Figure 11.
- G/G/m Simulation: Uses realistic distributions of packet interarrival times and packet processing times. The interarrival times are obtained from a network traffic trace. The packet processing times are obtained as described for the M/G/m simulation above.

For each runtime system, we simulate the configurations that are feasible. The M/M/m simulation matches most closely to our analytical evaluation while the G/G/m simulation reflects a more realistic system configuration.

2) System Performance for Different Data Rates: In Figure 12, we show the simulation results for each of the three runtime systems. Each subfigure shows the analytical results (some of which are shown in Figure 10) and simulation results for one runtime system.

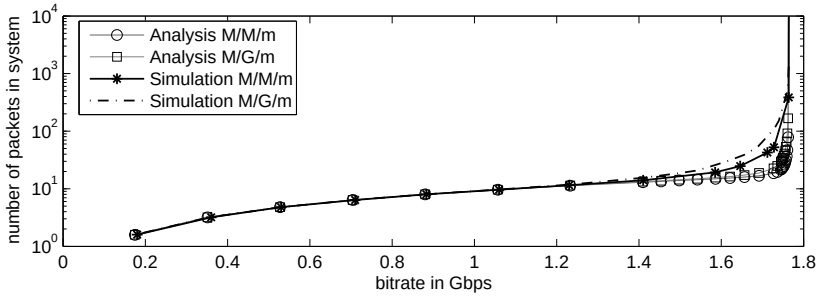
For the Ideal system (Figure 12(a)), we can compare the M/M/m and M/G/m models. As expected, the analytical results on number of packets in the system for both models are practically identical to those obtained from simulation except for very high data rates (and therefore high processor utilization). A difference of up to a factor of 2 can be observed when comparing the results from M/G/m and M/M/m for high bit rates in both analysis and simulation. This difference is due to the general distribution considering the coefficient of variation in addition to the service time mean. This leads to a higher number of packets in the system as the service time distribution shown in Figure 11 has a higher variance than the exponential distribution used in the M/M/m model.

A similar trend can be observed for the Full Processor Allocation system in Figure 12(b). Here, analysis can only provide results for M/M/m. The simulation results for M/M/m and M/G/m track the analytical results closely. Again, M/G/m provides higher (and more accurate) results for the number of packets in the system because of the underlying service time distribution.

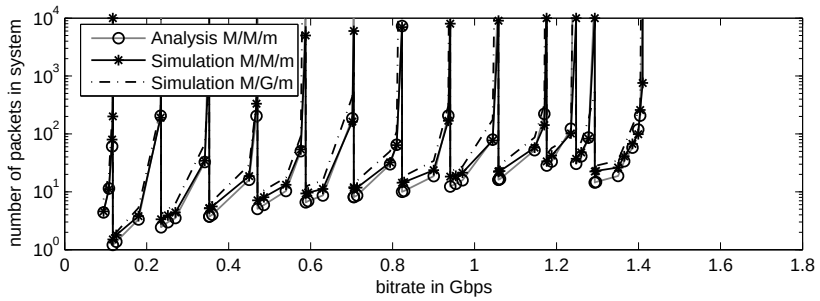
The Partitioned Application Allocation system results are shown in Figure 12(c). The simulation results are based on M/M/m (as are the analytical results) and on M/D/m. The M/D/m model is a special case of the M/G/m model with deterministic processing times. This is applicable in this case since in a partitioned application, instruction basic blocks are always executed with constant processing time. This determinism is reflected in the lower number of packets in the system.

We observe that for all three systems, analysis and simulation yield very similar results. The only considerable difference occur for high bit rates and for high processor utilization in the Full Processor Allocation scenario. Nevertheless, analysis can be used to get a good estimate of number of packets in the system.

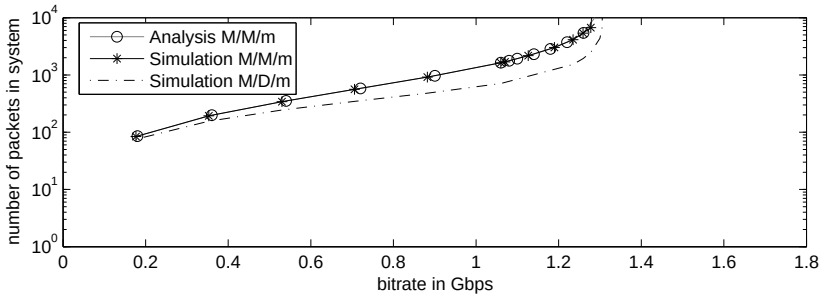
3) System Performance Over Time: The above simulation results do not consider a G/G/m model with realistic network traffic. In this section we explore simulation results that are based on a real traffic trace (scaled to the appropriate bitrate) and consider the "memory effect" between reallocations of processing resources (i.e., packets that are queued but not yet processed remain in the system). For the Full Processor Allocation system, we set the high/low threshold that triggers reallocation to 10/0 (Simulation 1) and 20/15 for tasks with high processing requirements and 5/3 for tasks with lower processing requirements (Simulation 2). The



(a) Ideal System



(b) Full Processor Allocation System



(c) Partitioned Application Allocation System

Fig. 12. Packets in System over Different Data Rates.

queue length is limited in this simulation.

The simulation results of packets in system over time using our sample traffic are shown in Figure 13. The simulation results closely track the behavior of the analytical results.

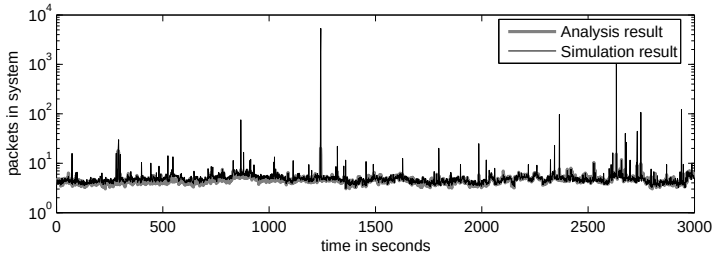
For the Ideal Scenario, the results are practically identical and show that the general arrival time distribution does not yield different results. For Full Processor Allocation, simulation shows a lower number of packets in the system than analysis due to the settings of high/low queue length thresholds, which trigger the allocation of more or less processing resources. For the Partitioned Application Allocation, the simulation provides a slightly lower number of packets in the system due to the use of a deterministic service time distribution in the simulation.

Overall, the simulation results show that our analytical models match closely the operational behavior of network processor runtime systems, even when considering realistic packet arrival and processing time distributions. This indicates that it is possible to use analytical performance analysis when exploring different runtime system design, which is much easier to do than implementing complex simulation setups. It is also possible to use analytical performance modeling within a runtime system to evaluate the current state of the system and make short-term performance predictions.

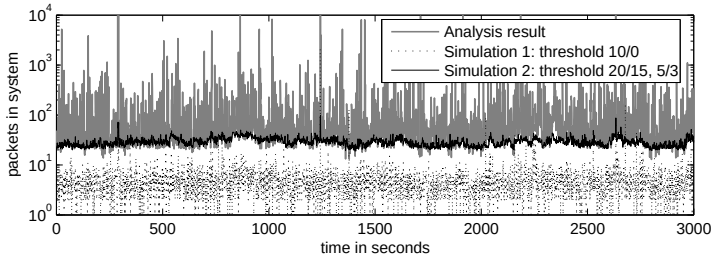
D. Implications for Runtime System Design

The above results yield several observations and conclusions with regards to runtime system design for network processors:

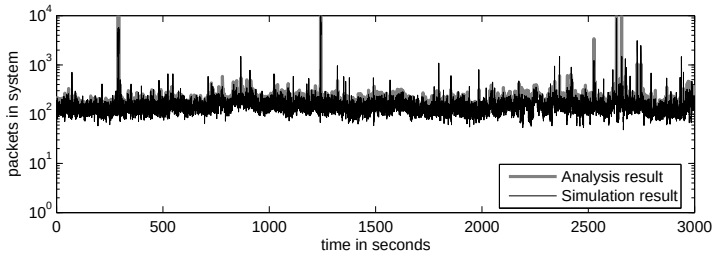
- Static allocations are not effective under varying traffic conditions. Figure 5 shows that only 20–60% of utilization would be achieved in such a scenario.
- Runtime systems which allocate only some of the processors to conserve power cause longer packet delays due to longer packet queues (see Figure 8). It is also possible to encounter longer delays when the system is not fully utilized (see spikes in Figure 8). It is therefore important to design the trigger mechanism for allocating additional processors to be sensitive enough to avoid that these effects become problems.
- High variation in traffic causes longer packet delays (see Figure 9). This is inherent to all runtime system that allocate some tasks to some subset of processors. Only the Ideal system, where all processors can process all applications, can avoid this problem. It may be desirable to investigate



(a) Ideal System



(b) Full Processor Allocation System



(c) Partitioned Application Allocation System

Fig. 13. System Performance Over Time.

if such an Ideal system can be implemented as it performs well and would be trivial to manage during runtime.

- The analytical result can estimate the trend of the system performance nearly as accurately as simulation. This enables runtime system designers to utilize analytical performance modeling for design-space exploration and runtime performance estimation.

VIII. Summary and Conclusions

In summary, this paper presents a methodology for evaluating runtime systems for network processors, we use both analysis and simulation for system performance evaluation. The motivation for this work lies in the need for a standardized evaluation process for runtime systems that are currently being developed. The presented methodology considers the workload of the system in terms of processing characteristics and traffic characteristics. To exercise evaluated system with a wide range of traffic patterns, we present a mechanism for generating synthetic traffic traces for different scenarios. The system model uses a queuing network abstraction to represent different runtime systems and allows for an analytical performance evaluation. We present results that compare two existing runtime systems that have been published in literature using both analysis and simulation method. The results are compared to a static configuration and an ideal system. The results show that runtime adaptation is necessary but causes overhead over an idealized system. Our methodology allows us to quantify this overhead for different workload scenarios. Also, the results show that in most cases analytical results can provide valid performance estimations that are comparable to those obtained from simulation. Therefore, analytical evaluation provides an easy way to estimate system performance trend and compare the performance of different runtime systems. We believe this is an important step towards developing a benchmark for network processor runtime systems.

Acknowledgements

This paper is based upon work supported by the National Science Foundation under Grant No. CNS-0447873.

References

[1] J. Allen, B. Bass, C. Basso, R. Boivie, J. Calvignac, G. Davis, L. Frelechoux, M. Heddes, A. Herkersdorf, A. Kind, J. Logan, M. Peyravian, M. Rinaldi, R. Sabhikhi, M. Siegel, and M. Waldvogel. IBM PowerNP network processor: Hardware, software, and applications. *IBM Journal of Research and Development*, 47(2/3):177–194, 2003.

[2] AMCC. *np7510 10 Gbps Network Processor*, 2003. `http://www.amcc.com`.

[3] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22(2):248–260, Apr. 1975.

[4] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queuing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. John Wiley & Sons, Inc., New York, NY, Aug. 1998.

[5] J. D. Brutlag. Aberrant behavior detection in time series for network monitoring. In *Proc. of the 14th Systems Administration Conference*, pages 139–146, New Orleans, LA, Dec. 2000.

[6] D. Clark, K. Sollins, J. Wroclawski, D. Katabi, J. Kulik, X. Yang, B. Braden, T. Faber, A. Falk, V. Pingali, M. Handley, and N. Chiappa. New Arch: future generation internet architecture. Technical report, Dec. 2003.

[7] P. Crowley and J.-L. Baer. A modelling framework for network processor systems. In *Proc. of First Network Processor Workshop (NP-1) in conjunction with Eighth IEEE International Symposium on High Performance Computer Architecture (HPCA-8)*, pages 86–96, Cambridge, MA, Feb. 2002.

[8] W. Eatherton. The push of network processing to the top of the pyramid. In *Keynote Presentation at ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)*, Princeton, NJ, Oct. 2005.

[9] EZchip Technologies Ltd., Yokneam, Israel. *NP-1 10-Gigabit 7-Layer Network Processor*, 2002. `http://www.ezchip.com/html/pr_np-1.html`.

[10] M. A. Franklin and S. Datar. Pipeline task scheduling on network processors. In *Proc. of Third Network Processor Workshop (NP-3) in conjunction with Tenth IEEE International Symposium on High Performance Computer Architecture (HPCA-10)*, Madrid, Spain, Feb. 2004.

[11] A. Gavrilovska, K. Schwan, O. Nordstrom, and H. Seifu. Network processors as building blocks in overlay networks. In *Proc. of Hot Interconnects*, pages 83–88, Stanford, CA, Aug. 2003. ACM.

[12] S. D. Goglin, D. Hooper, A. Kumar, and R. Yavatkar. Advanced software framework, tools, and languages for the IXP family. *Intel Technology Journal*, 7(4):64–76, Nov. 2003.

[13] M. Gries, C. Kulkarni, C. Sauer, and K. Keutzer. Exploring trade-offs in performance and programmability of processing element topologies for network processors. In *Proc. of Second Network Processor Workshop (NP-2) in conjunction with Ninth IEEE International Symposium on High Performance Computer Architecture (HPCA-9)*, pages 75–87, Anaheim, CA, Feb. 2003.

[14] Intel Corporation. *Intel Second Generation Network Processor*, 2005. `http://www.intel.com/design/network/products/npfamily/`.

[15] E. Khan, M. W. El-Kharashi, A. Ehtesham Rafiq, F. Gebali, and M. Abd-El-Barr. Network processors for communication security: a review. In *Proc. of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing 2003. PACRIM. 2003 IEEE Pacific Rim Conference onCommunications, Computers and signal Processing (PacRim 2003)*, Waikiki, HI, Feb. 2003.

[16] R. Kokku, T. Riché, A. Kunze, J. Mudigonda, J. Jason, and H. Vin. A case for run-time adaptation in packet processing systems. In *Proc. of the 2nd Workshop on Hot Topics in Networks (HOTNETS-II)*, Cambridge, MA, Nov. 2003.

[17] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft. Structural analysis of network traffic flows. *SIGMETRICS Performance Evaluation Review*, 32(1):61–72, June 2004.

[18] R.-T. Liu, N.-F. Huang, C.-H. Chen, and C.-N. Kao. A fast string-matching algorithm for network processor-based intrusion detection system. *Transactions on Embedded Computing Systems*, 3(3):614–633, Aug. 2004.

[19] J. Lu and J. J. Analytical performance analysis of network-processor-based application designs. In *Proc. of the The 15th Ineternational Conference on Computer Communications and Networks (ICCCN 2006)*, page 33–39, Arlington, VA, Oct. 2006.

[20] G. Memik and W. H. Mangione-Smith. NEPAL: A framework for efficiently structuring applications for network processors. In *Proc. of Second Network Processor Workshop (NP-2) in conjunction with Ninth IEEE International Symposium on High Performance Computer Architecture (HPCA-9)*, Anaheim, CA, Feb. 2003.

[21] W. Plishker, K. Ravindran, N. Shah, and K. Keutzer. Automated task allocation for network processors. In *Proc. of Network System Design Conference*, pages 235–245, Oct. 2004.

[22] R. Ramaswamy and T. Wolf. PacketBench: A tool for workload characterization of network processing. In *Proc. of IEEE 6th Annual Workshop on Workload Characterization (WWC-6)*, pages 42–50, Austin, TX, Oct. 2003.

[23] L. Ruf, K. Farkas, H. Hug, and B. Plattner. Network services on service extensible routers. In *Proc. of Seventh Annual International Working Conference on Active Networking (IWAN 2005)*, Sophia Antipolis, France, Nov. 2005.

[24] N. Shah, W. Plishker, and K. Keutzer. NP-Click: A programming model for the intel IXP1200. In *Proc. of Second Network Processor Workshop (NP-2) in conjunction with Ninth IEEE International Symposium on High Performance Computer Architecture (HPCA-9)*, pages 100–111, Anaheim, CA, Feb. 2003.

[25] J. Sommers and P. Barford. Self-configuring network traffic generation. In *Proc. of the 4th ACM SIGCOMM conference on Internet measurement (IMC)*, pages 68–81, Taormina, Italy, Oct. 2004.

[26] T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb. Building a robust software-based router using network processors. In *Proc. of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 216–229, Banff, AB, Oct. 2001.

[27] Teja Technologies. *TejaNP Datasheet*, 2003. `http://www.teja.com`.

[28] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. Design space exploration of network processor architectures. In *Proc. of First Network Processor Workshop (NP-1) in conjunction with Eighth IEEE International Symposium on High Performance Computer Architecture (HPCA-8)*, pages 30–41, Cambridge, MA, Feb. 2002.

[29] T. Wolf and M. Franklin. Performance models for network processor design. *IEEE Transactions on Parallel and Distributed Systems*, 17(6):548–561, June 2006.

[30] T. Wolf, N. Weng, and C.-H. Tai. Design considerations for network processor operating systems. In *Proc. of ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)*, pages 71–80, Princeton, NJ, Oct. 2005.

[31] L. Zhao, Y. Luo, L. Bhuyan, and R. Iyer. Design and implementation of a content-aware switch using a network processor. In *Proc. of 13th International Symposium on High Performance Interconnects (Hot-I05)*, Stanford, CA, Aug. 2005.

[32] W. Zhou, C. Lin, Y. Li, and Z. Tan. Queue management for QoS provision build on network processor. In *Proc. of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*, page 219, San Juan, PR, May 2003.

Xin Huang (S'06) received the B.S. and M.S. degrees, both in electrical engineering, from Tianjin University, China, in 2000 and 2003, respectively. She is currently pursuing the Ph.D. degree in the Department of Electrical and Computer Engineering at the University of Massachusetts, Amherst. Her research interests are network system design and performance evaluation. She is a student member of the IEEE.

Tilman Wolf (M'02–SM'07) received a Diplom in informatics from the University of Stuttgart, Germany, in 1998. He also received a M.S. in computer science in 1998, a M.S. in computer engineering in 2000, and a D.Sc. in computer science in 2002, all from Washington University in St. Louis. Currently, he is an associate professor in the Department of Electrical and Computer Engineering at the University of Massachusetts, Amherst.

He is engaged in research and teaching in the areas of computer networks, computer architecture, and embedded systems. His research interests include network processors, their application in next-generation Internet architectures, and embedded system security.

Dr. Wolf is a senior member of the IEEE and member of the ACM. He has been active as program committee member and organizing committee member of several professional conferences, including IEEE INFOCOM and ACM SIGCOMM. He is currently serving as treasurer for the ACM SIGCOMM society.