Enhancing Network Processor Simulation Speed with Statistical Input Sampling

Jia Yu, Jun Yang, Shaojie Chen*, Yan Luo, and Laxmi Bhuyan

Department of Computer Science and Engineering
*Department of Statistics
University of California, Riverside
{jiayu,junyang,yluo,bhuyan}@cs.ucr.edu, *schen009@ucr.edu

Abstract. While cycle-accurate simulation tools have been widely used in modeling high-performance processors, such an approach can be hindered by the increasing complexity of the simulation, especially in modeling chip multi-processors with multi-threading such as the network processors (NP). We have observed that for NP cycle level simulation, several days of simulation time covers only about one second of the real-world network traffic. Existing approaches to accelerating simulation are through either code analysis or execution sampling. Unfortunately, they are not applicable in speeding up NP simulations due to the small code size and the iterative nature of NP applications. We propose to sample the *traffic input* to the NP so that a long packet trace is represented by a much shorter one with simulation error bounded within $\pm 3\%$ and 95% confidence. Our method resulted one order of magnitude improvement in the NP simulation speed.

1 Introduction

Network processors have emerged as a solution to programmable routers with high processing capability. To better understand the effectiveness of the NP architecture, there have been a number of analytical models to quantify both the architecture and the energy features of an NP [2, 3]. Compared with those analytical models, a cycle-accurate NP simulator can faithfully emulate micro-architectural behaviors, which is more appropriate for optimizing designs in order to achieve better processing capability and energy effi ciency. However, cycle-accurate simulation of NPs is extremely time consuming. For example, one second of IXP 2400 hardware execution (600M cycles) corresponds to 10 days of simulation time in the Intel SDK 4.0 environment [1] on a Intel Xeon 3GHz PC with 512 MB memory in our experiment[1]. Similar results are observed even in a much lightweight simulator that we developed previously [8]. Apparently, the slow simulation speed limits its advantages for architectural optimizations and design space exploration over a large time scale.

There have been a number of techniques proposed to accelerate architectural simulation speed. The prevailing methodologies are: 1) truncated execution — terminating execution at a specifi ed point; 2) using reduced input sets — using a smaller input instead of a large input for shorter execution; 3) SimPoint [4] — a tool that selects and

[1] This is obtained from running the "OC12_pos_gbeth_2401" included in Intel SDK 4.0 with one Ethernet and one POS port under uniform arrival rate

simulates representative segments of a program (termed simulation points) via the analysis of the execution frequencies of basic blocks. The intuition behind it is that the overall behavior of the entire execution can be characterized by certain segments of the program executing an input. Machine learning techniques are used to divide execution intervals into *clusters*, and one or more points from each cluster for simulation. The final results are weighted summarized from each cluster result. One challenging issue how to choose the number of clusters *K*. In [16], Perelman *et al.* proposed a *Variance SimPoint* algorithm which used statistical analysis to guide the selection of *K* such that a *K*-clustering will meet a given level of confidence and probabilistic error bound. 4) sampling — the simulation intervals are sampled for execution from the entire simulation duration. Conte *et al.* [15] first applied statistical sampling in processor simulations in which the confidence and the probabilistic error bounds are used to control the accuracy. Another more rigorous statistical sampling method, SMARTS [5], assembles many detailed small simulation points with fast-forwarding (i.e. functional simulations) between them. This approach achieved low error bounds of $\pm$3% on average with a high confidence of 99.7%. A recent study [7] showed that SMARTS and SimPoint are both very accurate, while truncated execution and using reduced input sets have poor accuracy.

However, methodologies in SimPoint and SMARTS cannot be directly applied in NP simulations. The major reasons are: 1) **different benchmark characteristics**. Unlike SPEC2K benchmarks that have large and complex execution paths, the NP benchmarks periodically execute the same programs for processing multiple packets. Also, a single run of the NP program (processing a single packet) is very short, varying from hundreds to tens of thousands cycles only. With such short code paths, there is no need for applying SimPoint to find representative execution points, because most of the instructions are representative. 2) **different architectures and programming models**. For SPEC2K, both SimPoint and SMARTS are applicable for single threaded simulation. However, the NP architecture is typically multi-cored with multi-threading. Each core may be parallel or pipelined with its neighboring cores. Thus, the overall NP performance not only depends on the individual execution path in each core, but also the interactions among the cores and threads. 3) **different simulation focuses**. The standard input sets for SPEC2K are typically used without adaption. Thus results depend more on the program itself. In NP simulation, the metrics depend on not only the application code, but also the incoming traffic workload, e.g. low vs. high traffic volume, Ethernet vs. WWW traffic. Different traffic inputs will cause varying execution paths, and consequently tremendously different architecture states. Therefore, the input traffic analysis is more important for NP performance estimation than the code analysis.

We propose to conduct the input data analysis for accelerating NP simulations, complementary to the existing simulation techniques. We evaluate the effects of different input parameters to the NP architecture states, and observe that redundant simulation do exist. We show theoretically as well as confirm experimentally that sampling network inputs can significantly reduce the input sizes, while still bounding the error. In order to estimate the variation of the collected results, we used a similar approach as in the *Variance SimPoint* [16] algorithm with the difference in that we perform clustering on the input. Also, there is no need to consider warmups or fast-forwarding issues during the sample simulation due to the specialty of NP applications. This is because skipping

packets in the input traces has nearly no affect to the future packet processing due to the low locality in the packet payloads. With our methodology, we can achieve one order of magnitude speed up for the NP micro-architecture simulation.

In the next section, we discuss the characteristics of NP simulation inputs and the correlation between them and performance metrics. In section 3, we study the variations of NP simulation properties. In section 4, the stratified input sampling approach, as well as simple random sampling and systematic sampling are presented. Section 5 presents the results of proposed sampling methods. Section 6 concludes the paper.

2 Correlation between input and NP performance metrics

2.1 NP Programming model and Benchmark Applications

The programming model for a typical NP follows the receive-process-transmit paradigm [1]. The receive task collects the incoming packets from the network interfaces and reassembles sub-packets when necessary. The process task executes certain application functions to the packets such as packet filtering and address translation. The transmit task segments the packets and instructs the hardware to send them to the correct network port. The three tasks are mapped to different PEs with queues latching packet handles between them. There are four benchmarks ported to the original NePSim [8]. We compiled and ported three more cryptographic applications [9] for our experiments, and the descriptions are listed in table 1. More details about the benchmarks can be found in [8, 9].

Table 1. Description of the NP benchmarks

Benchmarks	Description
Ipfwdr	IPv4 Ethernet and IP header validation and trie-based routing-table lookup.
Nat	Network address translation, retrieve a replacement address and port for packets.
Url	Route packets on the bases of their contained URL request.
Md4	Cryptographic algorithm that produce a 128-bit fingerprint, or digital signature.
AES	Cryptographic algorithm incorporated into 802.11i
Blowfish	Cryptographic algorithm incorporated into Norton Utilities
RC4	Cryptographic algorithm incorporated into SSL/TSL, 802.1x

2.2 Characterization of Simulation Inputs

The major traffic parameters that affect the execution paths of packet processing are *packet size*, *arrival rate*, *packet source/destination addresses*, *protocol types*. In the following, we will discuss how much these parameters affect the simulation results.

– *Packet size*. Larger packet sizes usually cause longer packet processing time. For header processing applications, only headers will be decapsulated and processed, so larger packet size does not add much work to the processing stage. However, the receive and transmit stages are lengthened since longer time is spent moving packets between the bus interface and the packet buffer. For payload processing applications, larger packet sizes translate to more work in payload processing, and thus they cause longer processing time.

To show the relationship between the packet size and packet processing time, we run the seven benchmarks in NePSim with increasing packet sizes from 64 to 1088 bytes (i.e. 1 to 17 *mpackets*, and *mpacket* is the unit packet processing size in IXP family NPs), as shown in Figure 1 and 2. All other parameters are fixed. The arrival rate is 1000 packets/second/port with 16 device ports in total. We observe that average packet latencies increase with the packet sizes in all the seven benchmarks, and the cryptographic benchmarks *aes, blowfish, rc4* show linear increasing trend. This is because they spend most of the time in processing the packet payload stored in SDRAM. Thus the number of SDRAM accesses increases linearly with packet sizes for cryptographic benchmarks.

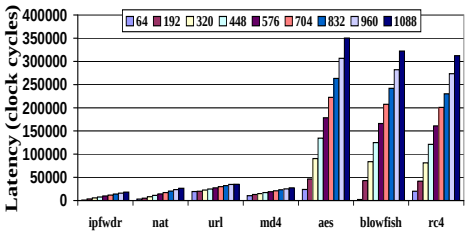

Fig. 1. Average packet latency of different packet sizes.

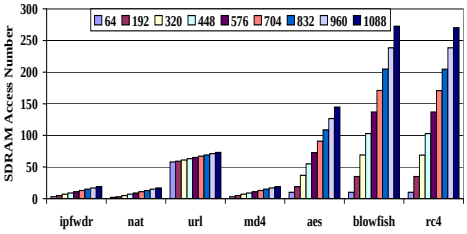

Fig. 2. Average number of SDRAM accesses per packet.

– *Packet arrival rate*. This parameter affects the NP architecture states in several ways. First, the threads execute different parts of the code under different arrival rates. At low traffic workload, the threads might be busy polling the *packet descriptor queues* to see if a new packet is ready. This is the case with functional pipelining[1] where the PEs are interleaved with *packet descriptor queues*. Under high traffic volumes, the threads work mostly on packet processing. Since the code segments for packet polling and processing show different computation requirements, the simulation results for low/high traffic arrival rates are different. Second, under high traffic volumes, the average memory access latency can also be increased because of higher demands on the memory and the longer queuing time when the memory is saturated. As a result, the total packet processing time becomes longer.

  Figure 3 shows the relationship between the packet arrival rate and packet latency. Similarly, we only vary the arrival rate from 0.1Mbps, 1Mbps, 10Mbps to 100Mbps per port with 16 ports in total. The packet size is set fixed as 128 bytes, and the remaining parameters are unchanged. As we can see, the packet latencies do not change much until the benchmarks approach their maximum throughput. For *ipfwdr* and *nat*, their maximum throughput are approximately 1.3Gbps. Therefore, we observe longer packet latency when the input traffic is 1.6Gbps. For the payload applications, we observe longer packet latency at 160Mbps arrival rate, because their maximum throughput are around 150Mbps. Figure 4 and 5 report the SDRAM and SRAM average access latencies at different packet arrival rates. The memory latency here includes the memory queuing time, and it increases sig-

nifi cantly when the inputs approach the applications' maximum throughput. The memory here more or less becomes the performance bottleneck.

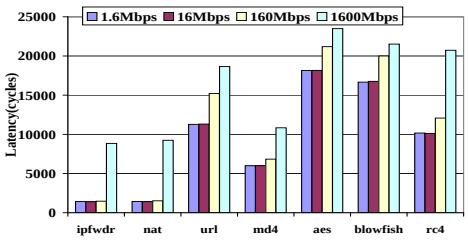

Fig. 3. Average packet latencies (128B/pkt).

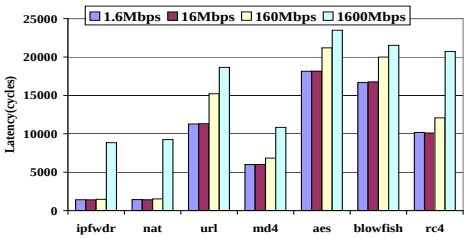

Fig. 4. Average SDRAM access latencies (128B/pkt).

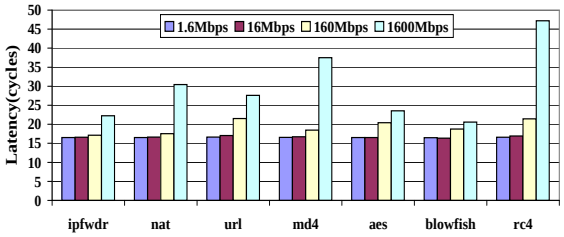

Fig. 5. Average SRAM access latencies (128B/pkt).

– *Packet source/destination addresses and protocol types*. Different values in packet headers trigger different execution paths. Packet source/destination addresses are important information for applications that need to perform routing table lookups such as the IP forwarding benchmark. However, through experiments, we found the variation in destination addresses only cause at most 2% variation in packet latencies for *ipfwdr*. This is because the percentage of the code for address look up is relatively small. Therefore, the variance in the destination addresses does not affect the NP performance signifi cantly.
The NP can be programmed to process packets of different protocol types. However, this does not necessarily cause signifi cant differences in NP architecture states when packets of one protocol type dominates. Take the *ipfwdr* as an example, the code length and complexity for different protocols are usually similar. The execution paths of processing different IP packets only differ by a small amount in the packet header decapsulation part, while the IP address lookup and exception checking/handling parts are the same. Therefore, in our input sampling technique, we do not consider the IP packet distribution from different network protocols.

2.3 Correlation Between Inputs and NP Performance Metrics

In this section, we demonstrate that the NP architecture states are correlated with the network traffi c inputs. Only if they are correlated, can we say a particular traffi c pattern corresponds to a certain architecture state (e.g. throughput, packet latency). Thus by

identifying representative subsets of traffic, we can skip the simulation of those packets that produce similar architecture states.

To quantitatively measure the correlation, we divide the simulation time to n windows with window size as 0.01 second (will be explained in section 4). We then calculate the *correlation coefficient* r [13], for data pairs: (arrival rate, packet loss), (arrival rate, packet latency), (arrival rate, PE idle time) and (arrival rate, IPC).

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 \sum_{i=1}^{n}(y_i - \overline{y})^2}}, i = 1, ..., n \quad (1)$$

$r > 0$ means the data pairs are *positively correlated*, and $r < 0$ means that data pairs are *negatively correlated*. A value of $|r| > 0.8$ means that the data pairs have strong linear relation. A value of $|r| < 0.3$ means that the linear relation is relatively weak. We plot the values of r for *nat* given a set of arrival rates (from 0 to 1800Mbps with 128 bytes per packet and Poisson distribution for the packet arrival rate in Figure 6. Other benchmarks show similar trend as Figure 6.

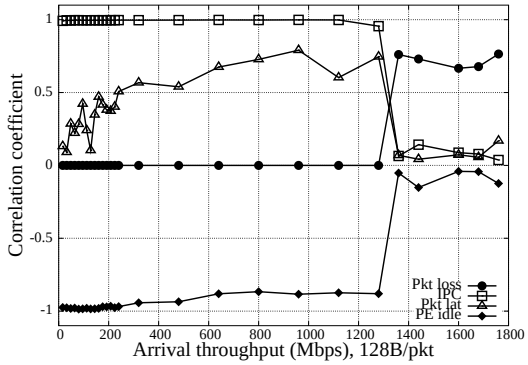

Fig. 6. Correlation of arrival rate and NP metrics (Poisson arrival pattern, 128B/pkt).

We can see from Figure 6 that there are three sections where different metrics show different degrees of correlation with the packet arrival rate. Under low traffic volumes (<300Mbps), there is no packet loss and thus this metric has no correlation with the packet arrival rate. The packet latency has weak correlation with the arrival rate, which means a little variation of arrival rate at light traffic volume does not cause much difference in the average packet latency. The other two metrics, IPC and the PE idle time are both strongly correlated with the packet arrival rate. IPC increases and the PE idle time decreases almost linearly with the increasing packet arrival rates.

With medium traffic volume (300Mbps<packet arrival rate<1300Mbps), the correlation stays about the same for all the metrics except for the packet latency. The r value for data pairs (arrival rate, packet latency) approaches to or exceeds 0.8. This means the packet latency has relatively strong correlation with the packet arrival rate at medium traffic volume which is the typical case for an NP.

With high traffic volume (packet arrival rate>1300Mbps), all the metrics we measure exhibit different correlations with the packet arrival rate. First of all, the benchmark

nat reaches its maximum throughput at 1300Mbps so some of the incoming packets will be dropped. At this time, the input traffic already saturates the NP processing capability, so all metrics (IPC, packet latency, PE idle time) no longer fluctuate with the input except for the packet loss. This metric starts to become positively linear to the packet arrival rate.

From this study, we know that the performance results (IPC, packet latency and PE idle time) under medium to high traffic volume are linearly correlated to the inputs. Thus sampling inputs at medium to high traffic volume can effectively help calculate the corresponding architecture states.

3 Variation of NP Simulation Properties

In this section, we will examine the variation of architecture states, and its relationship with the variation of traffic input. The study of the architecture states is important because their variation is used to estimate the required sample sizes to achieve a certain confidence level. A large variation corresponds to a large sample size.

According to the sampling theory [13], the estimate of a property x such as the IPC, average packet latency etc. can be represented as:

$$\bar{X} \pm z \times \frac{\sigma_X}{\sqrt{n}} \tag{2}$$

where $\bar{X}$ is the *mean* value of property x in simulation, z is percentile of the standard normal distribution (z=2.0 for 95% and z=3.0 for 99.7% confidence), n is the *sample size*, and σ_X is the *standard deviation* of property x which depends on both simulation configurations and the input workload. To measure the sampling variation of the estimate relative to the mean of the property x being estimated, it is a standard practice to use the *coefficient of variation* V_x ($V_x = \frac{\sigma_X}{\bar{X}}$). Large V_x means the sampled values vary significantly around the mean value $\bar{X}$, thus a large sample size is needed. Small V_x means the sampled values are quite homogeneous, thus a small size is enough. Formula 2 can also be written as:

$$\bar{X} \pm z \cdot \frac{V_x}{\sqrt{n}} \cdot \bar{X} = \bar{X} \cdot (1 \pm z \cdot \frac{V_x}{\sqrt{n}}) \tag{3}$$

To bound the error rate $z \cdot \frac{V_x}{\sqrt{n}}$ at *confidence interval* $[-\varepsilon \cdot \bar{X}, \ +\varepsilon \cdot \bar{X}]$ and ε is a confidence percentile chosen by users, e.g. 3%, 5%, the sample size n should be changed proportionally to V_x^2, $n \geq (\frac{z \cdot V_x}{\varepsilon})^2$. A large V_x requires a large sample size n to meet a certain confidence. The coefficient of variation V_x (x is a performance metric) is rarely available unless we run the full simulation. Often, an estimation of V_x, $\hat{V}_x$ is used. For example, in SMARTS, $\hat{V}_x$ is estimated from a large initial sample set, as long as the IPC in later periods do not vary significantly. In NP simulation, a large initial sample set cannot estimate the true V_x accurately because the traffic volume after the initial period may change significantly. Correspondingly, the value of V_x will be different. Next, we analyze the possibilities and the challenges of estimating the V_x.

We define $V_{performance}$ as $MAX((V_{x_1}, V_{x_2}, \cdots, V_{x_m})$, where $x_1, \ x_2 \ \ldots \ x_m$ represent our target simulation metrics (e.g. packet loss ratio, packet latency etc.). Hence, the V_xs are subsumed in the $V_{performance}$. Recall from what was discussed in section 2, the

variation of an architecture metric is correlated with the variation of the input traffic, or the workload of the NP. Let $V_{workload}$ be $MAX(V_{arrival\_rate}, V_{packet\_size}...)$, which can be easily obtained by parsing the input traffic. $V_{workload}$ is correlated with $V_{performance}$ and serve as an estimate to V_x.

If $V_{workload} \geq V_{performance}$, we can conservatively use $V_{workload}$ to replace $V_{performance}$. Hence, the calculated sample size $\hat{n} = (\frac{z \cdot V_{workload}}{\varepsilon})^2 \geq (\frac{z \cdot V_{performance}}{\varepsilon})^2 = n$. In other words, taking $\hat{n}$ elements can bound the error rate at a certain confidence level.

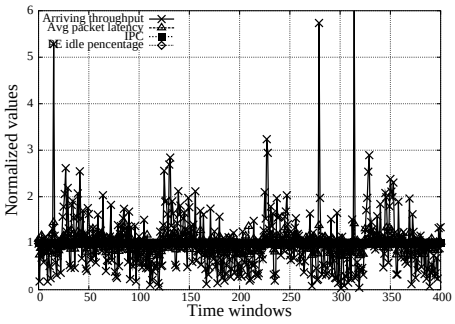

Fig. 7. The variation of the packet arrival rate is significantly larger than variation of NP performance metrics for *url* under relatively low traffic volume (40Mbps). The Y axis is normalized to the mean value.

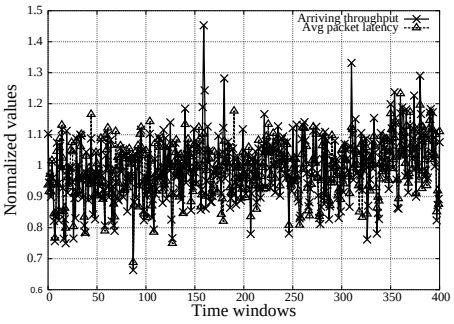

Fig. 8. The variation of the packet arrival rate is close to the variation of packet latency for *rc4* under low traffic volume (40Mbps). The Y axis is normalized to the mean value.

To see if $V_{workload} \geq V_{performance}$ is generally true, we run simulations in NePSim with different input traces. All the input traces are extracted from real world traces from NLANR[12]. Each trace contains 50 second network packets (roughly 500K to 2500K packets). For all our targeting NP metrics, The relationship between $V_{workload}$ and $V_{performance}$ can be categorized into three groups:

- *Group 1:* $V_{workload} > V_{performance}$. Some performance metrics such as IPC and PE activities (execution, stall, idle, abort) vary less significantly than the traffic volume. This phenomenon happens because of the special NP architecture and programming model. When there is a traffic spike, the IPC and PE activities do not change dramatically because the internal buffering scheme in NP's receive-process-transmit model smooths out them effectively when the traffic volume is not overly high.
  The packet latency in relatively low traffic volume shows less variation than the traffic input also. Figure 7 plots such an example. In this figure, the traffic volume varies between 0 to 6x of the mean value. However, the corresponding average packet latency only varies within a small window: [0.5x, 1.5x].
- *Group 2:* $V_{workload} \approx V_{performance}$. For cryptographic benchmarks, the variation of the packet latency is very close to $V_{workload}$. Figure 8 shows the normalized variation of packet latency with respect to the packet arrival rate for *rc4*. We can see that the packet latency changes proportionally to its arrival rate.
- Group 3: $V_{workload} < V_{performance}$. At a high traffic volume, we observe that the $V_{performance}$ might be larger than $V_{workload}$, especially for packet loss and packet

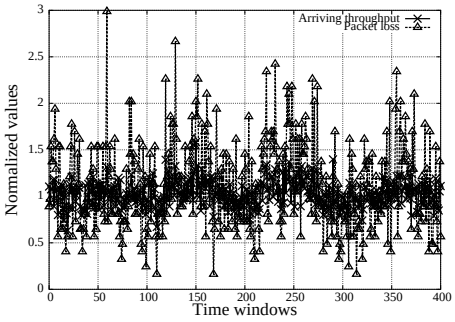

Fig. 9. The variation of the packet arrival rate is less than the variation of packet loss for *ipfwdr* under high traffic volume (1000Mbps). The Y axis is normalized to the mean value.

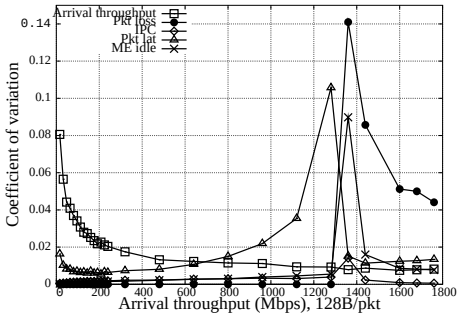

Fig. 10. The variation of all the metrics for *nat* with inputs that follow the Poisson distribution.

latency. Figure 9 shows that the packet loss metric has higher variation than input arrival rate, when the arrival rate approaches 1000Mbps for *ipfwdr*. This is because if the NP is close to saturation, the queuing systems in NP lengthen the service latency signifi cantly (as shown in Figure 3 to 5), which magnifying the variations of internal performance metrics.

If all the simulations belong to group 1 and 2, we can safely use $V_{workload}$ to calculate a sample size. However, we did fi nd that some simulation periods belong to group 3. In Figure 10, we plot the $V_{workload}$ and V_x of different metrics for *nat* with packet arrival rate varying from 0 to 1800Mbps. We observe that under low and extremely high traffi c volumes, the coeffi cient of variance of our targeting metrics tend to be small. These periods belong to group 1, and a small sample size is enough for a good simulation. However, with medium to high traffi c volume, the variation of the NP performance can be larger than the variation of the input. Using $V_{workload}$ to replace $V_{performance}$ might underestimate the sample size and introduce inaccuracy. Therefore, we cannot completely replace $V_{performance}$ with $V_{workload}$, and we still need to estimate the $V_{performance}$ from an initial simulation. The difference between ours and SMARTS is that we extract the initial simulation points according to the distribution of the workload. We will illustrate our method in the next section.

4 Statistical Input Sampling Technique

Developing an input sampling mechanism for NP simulation requires the evaluation of the trade-off between the sample size and the simulation accuracy. Our goal is to fi nd a sample size that accurately describes the entire population of the packets. As we discussed earlier, $V_{performance}$ cannot be known unless a full simulation is conducted. Here we propose a two-phase sampling process to solve this problem. The idea of this process is to get *an estimation of* $V_{performance}$, $\hat{V}_{performance}$ through an initial simulation. The input for the initial simulation is sampled from original input trace, according to $V_{workload}$. Recall that $V_{workload}$ is easily estimated by parsing the input trace. If the initial sample size is large enough, $\hat{V}_{performance}$ will be quite close to $V_{performance}$.Thus we can

use $\hat{V}_{performance}$ to estimate a more accurate sample size. The process of two-phase sampling mechanism is illustrated in Figure 11.

In the first phase we conduct a pre-survey through an initial simulation. We use $V_{workload}$ to estimate the initial sample size n_1. We then feed the initially sampled packets to the simulator and calculate $\hat{V}_{performance}$ as an estimation of $V_{performance}$. If $\hat{V}_{performance} > V_{workload}$, we use $\hat{V}_{performance}$ to estimate the accurate sample size n_2. We sample the additional $n_2 - n_1$ elements from the input trace and simulate them. If $\hat{V}_{performance} \leq V_{workload}$, we know enough elements have been simulated, and the additional sampling is not necessary. The post processing stage will multiplex the sampled simulation results to get the final results. In the following, we will describe the input sampling using three methods: *simple random*, *systematic random* and *stratified random samplings*, and discuss their trade-off.

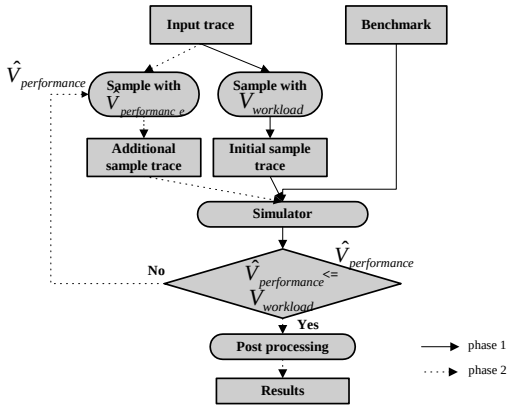

Fig. 11. The two-phase input sampling methodology for NP simulation.

4.1 Simple Random and Systematic Random Samplings

Suppose the full input trace has N-second of packets. Each sample contains U-second of traffic packets. If there are n samples, we get the total sample size as $n \cdot U$.

The *simple random sampling* uniformly selects n periods of traffic packets from the full input trace at random. The *systematic random sampling* selects the sample periods at a fixed sampling interval k such that $n = N/k$. As explained in section 3, to bound confidence interval $\pm\varepsilon \cdot \bar{X}$, the sample size should satisfy:

$$n \geq (\frac{z \cdot V_x}{\varepsilon})^2 \tag{4}$$

4.2 Stratified Random Sampling

Simple random sampling and systematic sampling, each involves taking a sampling from a population as a whole, neither requires identification of subdomains or subgroups before the sample is taken [10]. In other words, they do not differentiate between the high traffic volume and low traffic volume. Additionally, they do not exploit the frequent similar behaviors in network traffic traces, resulting many redundant sample collecting. This observation motivates us to first partition the traffic inputs into groups

or strata (low, medium, high traffic etc.), and sample separately within each stratum. The resulting sampling design is the *stratified random sampling*. The purpose of stratification is to minimize the intra-stratum variance while maximizing the inter-stratum variance. Thus a small sample size within each stratum can meet the desired confidence. In addition, the sample sizes taken from strata can be non-uniform, depending on the variations in the strata.

The process of stratified trace sampling is illustrated in Figure 12. We first cluster the traffic trace into K strata based on the packet arrival rate and the packet size using the K-means algorithm [11]. The K-means algorithm is one of the fastest clustering algorithms. It clusters the elements according to their distances to the cluster centroid, and iteratively change the cluster centroid until element membership cease to change. The ratio between the *intra-cluster* and *inter-cluster coefficient of variations*, denoted by β_{CV} [14], is a useful guide to determining the quality of the clustering process. The smaller the value of β_{CV}, the better the clustering. From statistical experience, we choose the stratum size K from 3 to 15 so that the sample sizes in different strata are not too small. Within a particular stratum h, we select a sample of n_h elements from the

Stratified Trace Sampling

1. Divide the traffic trace to N periods, measure the throughput and average packet size of each period.
2. Cluster the N periods using K-means algorithm according to the throughput and average packet size.
   a. Select the number of strata $K(3 \leq K \leq 15)$ that produces strata with less than 90% of minimum β_{CV} and sample size.
   b. Calculate required sample size in each stratum, based on the strata sizes strata variations and desired confidence interval.
3. Sample the traffic trace in each stratum and calculate weight of each sampled period, which is proportional to the total number of elements in the stratum.

Fig. 12. Process of stratified trace sampling

N_h elements in the stratum, and each element is measured with respect to some variable x (i.e. throughput, packet size).

To calculate n_h, we apply the *optimal sample allocation* method[6] [10]. The intuition behind this method is that the optimal number of sample elements to be taken from a given stratum is proportional to N_h, the total number of elements in the stratum, and to σ_{hx}, the standard deviation of x among all elements in the stratum. To ensure that the central limit theorem [10] holds, n_h should not be too small. In our implementation, we constrain the minimum sample size for each stratum to be 30.

The stratified trace sampling should generate *weights* for each sampled element, because sampled elements selected from larger stratum should be of greater importance. The weights will be provided to NP simulator, which will scale the simulation result in the post processing stage. We define *weight* of a sampled element as $\frac{N_h}{n_h}$, and the estimated performance metric x can be represented as

$$\sum_{i=1}^{N} X_i \approx \sum_{h=1}^{K} \sum_{i=1}^{n_h} \frac{X_i}{n_h} \cdot N_h = \sum_{h=1}^{K} \sum_{i=1}^{n_h} X_i \cdot weight_i \tag{5}$$

4.3 The Sample Unit Size Selection

The choice of sample unit size U will affect the variation among the periods, and consequently the total sample size. Therefore, we should use U that produces smaller sample sizes. In addition, the U value should be larger than per packet processing time. Next, we show an example of how to determine the value of U.

Figure 13 shows the traffic volume of a 12-hour traffic trace. We divide it into 24 half-hour traces and use different sample unit size $U = 0.001sec$, $0.01sec$, $0.1sec$, $1sec$, $10sec$ to sample each trace. As we can see from Figure 15, the *coefficients of variation* approximately reduce from 1.8 to 0.25 as U increases. This phenomenon is intuitive because the network spikes or dips are smoothed out over longer periods. When $U \geq 0.1sec$, we observe the required sample sizes increase significantly. This is because U scales much faster than the reduction of n (in proportion to ${V_x}^2$). In this example, $U = 0.1sec$ is suitable for sampling because it achieves small sample size and it is significantly longer than per packet processing time. For comparison purpose, we plot the required sample size for the 24 traces in Figure 14. We can see that the required sample size is correlated with the traffic shape in Figure 13. The network spikes appear in trace 16 and 6 on the x-axis, so do the sample sizes.

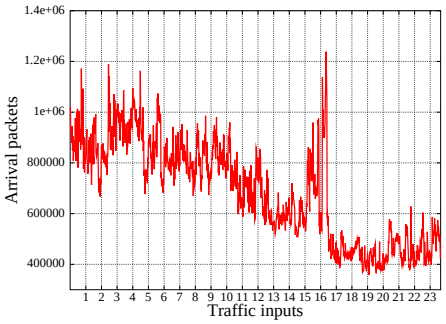

Fig. 13. Traffic volume of Leipzig trace 2002-11-12 from NLANR, the packets arrived in every 30 seconds.

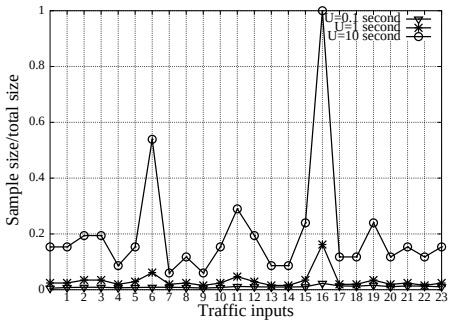

Fig. 14. Required sample sizes to achieve $\pm$ 3% error with 95% confidence in systematic sampling using 0.1sec, 1sec 10sec sampling unit size

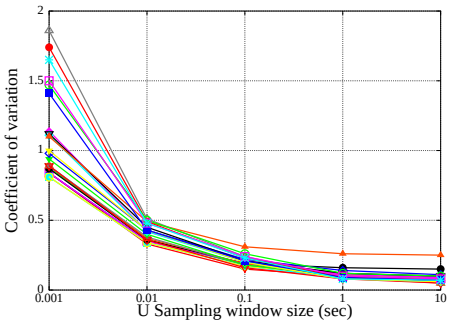

Fig. 15. The coefficients of variation of the traces with various sample unit sizes.

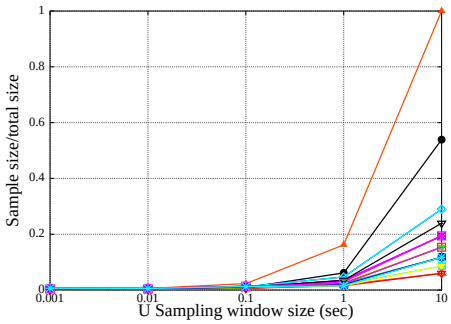

Fig. 16. Required sample sizes to achieve $\pm$3% error rate with 95% confidence

4.4 Comparison of The Three Sampling Methods

Simple random and systematic random sampling are comparatively easy to implement, and they are useful to sample traces that have low simulation variations. Inputs with low or extremely high traffic volumes belong to this category. Stratified sampling is effective to reduce the sample size for traces with medium to high traffic volume. In simple random sampling case, usually we need a larger sample size to achieve a specified confidence level because we know little information about the whole population. When the population is highly non-homogeneous, the stratified sampling usually performs better than the systematic random sampling and simple random sampling [10].

5 Experiment

In this section, we present the experiment results of input sampling. Since the three sampling methods can all produce accurate results with large sample sizes, we will compare in two ways: 1) sample sizes for achieving same accuracy 2) for the same sample size, how much accuracy attainable for the three sampling methods. We run the full simulation for the original traces, and compare the results with simulations of sampled traces to get the error rates. We used six inputs extracted from three real world NLANR [12] traces *Leipzig-I, Abilene-I and Tera-I*. *Leipzig-I* is an OC-3 trace collected from an edge access link. *Abilene-I* is an OC48 trace collected at an Indianapolis core router. *Tera-I* is an OC192 10GigE ethernet trace. Each of the six inputs is 50-second long, and costs 2 days for full simulation in NePSim. Two of the input traces are low or medium traffic volume (less than 300Mbps), the other four are high traffic load where packet loss occurs (higher than 300Mbps). They have different $V_{workload}$, varying from 0.1 to 0.8. For sample size estimation, we bound the error rate at 3% within 95% confidence. We use 0.01 second as sample unit size, because the tested trace is only 50 second long and $U = 0.01$ can effectively reduce sample size.

We expect the simulation time in terms of core cycles will be proportional to input trace length. The speedup attainable is dependent on the sample size, and thus is affected by the architectural variation and total trace length. An extreme case is that the incoming packets are uniformly distributed. In that case, one element is enough to estimate the overall performance, so the speedup would be very large. However, real world traces are rarely uniform, so the speedup attainable is lower.

Figure 17 plots the speedups achieved for simulating the six traces over the three sampling techniques for a common error bound and confidence interval. The systematic or simple random samplings can only achieve 1.8x speedup. While using stratified sampling, the input traces can averagely be reduced by 11 times. We do not observe higher speedups, because the original traces are short and we constrain the sample size in each stratum to be at least 30 (due to central limit theory).

Next, we show the results when the same sample size is used across different sampling techniques. Since stratified sampling requires the smallest sample size among the three sampling methods, we use this smallest sample size for the other sampling methods and compare the accuracy. Figure 18 shows the error rate of traffic throughput processed by an NP. Both systematic and stratified sampling can achieve less than 3% error. The simple random sampling produces higher error rate, e.g. 7% on average for *url*. This shows the given sample size is not enough for simple random sampling. Figure

19 shows the error rate of average packet latency. We can see that the three sampling methods are bounded within 3% error for most benchmarks. This is because packet latency has comparatively smaller variance than throughput, thus the given sample size is enough. For *md4*, we observe 4% error rate for simulating one trace. We find the population of this trace is rather homogeneous according to its variance. In this case, the three sampling methods do not differ greatly. In addition, the error rate of *md4* is not bounded by 3%. It is due to the bias of the estimation of $V_{performance}$ in the pre-survey process for this particular trace.

Figure 20 shows the error rate of simulated packet loss ratio. The simple random sampling has error rates larger than 5%. In worst case, the error rate even achieves 80% for a highly non-homogeneous input. For the systematic sampling the average error rates for *nat, url, blowfish, rc4* are larger than 3%. It shows that systematic sampling requires larger sample size to achieve 3% error rate than stratified sampling. Regarding the remaining performance metrics, such as PE idle percentage and IPC, all three sampling methods can achieve less than 3% error rate, because these metrics have smaller variation than packet loss and throughput. In this set of experiments, we observe that stratified sampling performs better than the other two sampling methods for traces that have large variations.

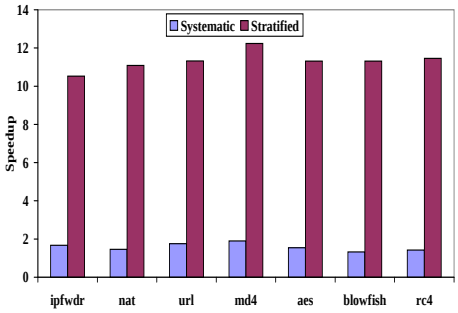

Fig. 17. Speedup achieved using two input sampling methods within ±3% error rate with 95% confidence. *Simple random sampling* has similar speedup as systematic sampling.

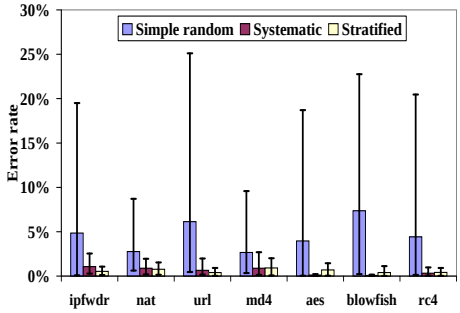

Fig. 18. Error rate of throughput using a same sample size

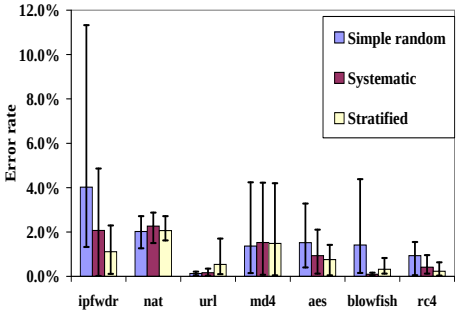

Fig. 19. Error rate of simulated average packet latency using a same sample size

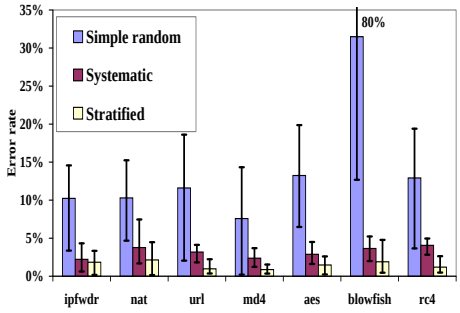

Fig. 20. Error rate of simulated packet loss ratio using a same sample size

6 Conclusion

We developed a statistical input sampling methodology for accelerating the NP simulations.We experimented several sampling techniques and conclude that the stratified sampling results in the smallest sample size and best accuracy. The speedups we obtained is dependent on the length of the input trace. The longer the trace, the higher the speedup. The outcome of this study can be used for fast estimating NP activities and network performance metrics for non-saturated and non-uniform network traffic. It can also be applied for measurements exploiting dynamic optimizations for better performance and energy efficiency.

References

1. Intel Corporation. "Intel IXP2XXX Product Line of Network Processors." http://www.intel.com/design/network/products/ npfamily/ixp2xxx.htm
2. S. Lakshmanamurthy, K.Y. Liu and Y. Pun. "Network Processor Performance Analysis Methodology," *Intel Technology Journal*, vol. 06, 2002.
3. M.A. Franklin and T. Wolf, "Power Considerations in Network Processor Design," *Workshop on Network Processors in conjunction with HPCA-9*, 2003.
4. T. Sherwood, E. Perelman, G. Hamerly and B. Calder. "Automatically Characterizing Large Scale Program Behavior, " *the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.
5. R.E. Wunderlich, T.F. Wenisch, B. Falsafi and J.C. Hoe. "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling," in *International Symposium on Computer Architecture*, 2003
6. R.E. Wunderlich, T.F. Wenisch, B. Falsafi and J.C. Hoe. "An evaluation of Stratified Sampling of Microarchitecture Simulations," in *Workshop on Duplicating, Deconstructing and Debunking*, 2004
7. J. J. Yi, V. Kodakara, R. Sendag, D.J. Lilja and D.M. Hawkins, "Characterizing and Comparing Prevailing Simulation Techniques, " in *International Symposium on High-Performance Computer Architecture*, 2005
8. Y. Luo, J. Yang, L. Bhuyan and L. Zhao, "NePSim: A Network Processor Simulator with Power Evaluation Framework," in *IEEE Micro*, Sept/Oct, 2004
9. Z.X. Tan, C. Lin, H. Yin and B.Li, "Optimization and Benchmark of Cryptographic Algorithms on Network Processors" in *IEEE Micro*, Sept/Oct, 2004
10. P.S.Levy and S. Lemeshow, "Sampling of Populations: Methods and Applications", John Wiley and Sons, Inc., 1999
11. D.Pelleg and A. Moore. "Accelerating Exact K-means Algorithms with Geometric Reasoning," in *Proceedings of the Fifth International Conference on Knowledge Discovery in Databases*, 1999
12. Network traces, http://www.nlanr.net
13. S. M. Ross, "Introduction to Probability and Statistics for Engineers and Scientists," Elsevier Academic Press, 2004.
14. D.A. Menasce, V.A.F. Almeida, "Capacity Planning for Web Services", Prentice Hall, Inc. 2001
15. T.M. Conte, M.A. Hirsch and W.W. Hwu. "Combining trace sampling with single pass methods for efficient cache simulation," *the 1996 International Conference on Computer Design*, 1996.
16. E. Perelman, G. Hamerly and Brad Calder, "Picking Statistically Valid and Early Simulation Points" *International Conference on Parallel Architecture and Compilation Techniques*, 2003.