BRUNO: A High Performance Traffic Generator for Network Processor

Gianni Antichi, Andrea Di Pietro, Domenico Ficara, Stefano Giordano, Gregorio Procissi, Fabio Vitucci
Dept. of Information Engineering, University of Pisa, ITALY
Email: \{g.antichi, a.dipietro, d.ficara, s.giordano, g.procissi, f.vitucci\}@iet.unipi.it

Keywords: Traffic Generator, Network Processor, Extensible design, Timing accuracy, Error reduction

Abstract

The current software tools for traffic generation suffer from poor performance in terms of frames per second and timing/rate accuracy, because of the intrinsic limitations of the PC architecture. This paper proposes a different approach, based on a cooperative PC/NP architecture: an advanced software tool runs on a host PC and instructs the processing engines of an Intel IXP2400 Network Processor, which take care of the actual traffic generation. This way, we keep the high flexibility of PC solutions while outperforming them in terms of packet rate. In addition, a time correction mechanism is introduced, in order to improve the system precision in traffic models reproduction. The adoption of such a mechanism is justified through statistical analysis of the overall system: the experiments carried out prove the effectiveness of this approach in reducing the mean interdeparture time error. Moreover, they show the capability of BRUNO in generating 1.5 Mpps with high accuracy.

1. INTRODUCTION

In the last few years, interest in modern Internet applications has been constantly growing and a significant number of such applications has imposed strict demands on network performance. This has required reliable networks offering high transmission capacity, which in turn has paved the way to the need of network testing to measure performance and reveal any “weaknesses”. The evaluation of modern networks, however, is a very difficult task. In fact, given the high speed of current networks, simulations of their behavior (for example by means of tools such as the largely diffused ns2) are not possible with the proper accuracy: the unavoidable simplifications required by simulations has become not acceptable.

Therefore, the only viable direction to test modern networks is the emulation. This requires to generate traffic flows which are very similar to the actual internet traffic, in terms of data rate and statistical properties. This is a critical task due to the lack of reliable software tools to generate traffic at high rates. To address this issue, a very accurate traffic generator, called BRUTE (Brownny and RobUst Traffic Engine), has been implemented by our research group [1]. It has a flexible architecture and an extensible design, by providing a number of library modules implementing common traffic profiles. Although BRUTE outperforms all the widespread software tools (KUTE [2], RUDE [3], MGEN [4]) in terms of both the achieved throughput and the time precision, it is still limited by PC capabilities in terms of sustainable bit rates (i.e., it is able to generate a maximum traffic load of 400 Mbps).

As already said, such poor performance is due to the intrinsic limitations of the PC architecture, for which all these tools are designed. Therefore, different solutions have to be devised. The use of flexible hardware platforms to improve performance and accuracy looks unavoidable, and network processors (NPs) appear as appealing solutions for such purpose. Traffic generators based on NP platform are presented in works [5] and [6] and have inspired our activity.

This paper presents BRUNO (BRUte on Network prOcessor), a traffic generator built on the IXP2400 Intel Network Processor and based on a modified BRUTE version. BRUTE is designed to run on the PC that hosts the NP-card, with the aim of computing departure times according to given traffic models. Then the host PC writes these information in the memory shared with the packet processing units of NP (i.e., the microengines), which in turn use such data to generate packets and send them with the proper timeliness. The simulation of application code has shown a sustainable rate of 1 Gbps and a great accuracy in models reproduction, guaranteed by a feedback scheme for time correction.

Next section presents the previous works in the traffic generation area. In particular, BRUTE and NP-based solutions are analyzed. Sec. 3. illustrates the overall design of BRUNO, by describing all the components of our applications. In sec. 4. a performance analysis is made in terms of system delays and the time correction mechanism is carefully examined. The experimental results are illustrated in sec. 5., while sec. 6. presents the final considerations about this work.

2. RELATED WORKS

In this section we briefly introduce some of the most used and powerful traffic generators developed for PC, FPGA and NP architectures and related works of interest.

As for terminology, Paredes-Farrera et al. [7] provide a simple definition for precision and accuracy: the first is related to the quality and stability of the system, the latter measures the similarity among the created values and the true ones. Therefore, since we are interested in timeliness of generated packets, we refer to precision as the standard deviation
of generated times, while accuracy describe the average error.

Several open-source tools for traffic generation on commodity PCs have been proposed over the years, most of them are designed for the Linux Operating System. KUTE [2] (an evolution of the former UDPgen) is an UDP traffic generator which is designed to achieve high performance over Gigabit-Ethernet. It is based on a Linux kernel module that operates directly on the network device driver bypassing the Linux kernel networking subsystem. This means that its architecture is strictly related to the kernel, thus limiting its extensibility.

RUDGE [3], MGEN [4], ITG [8] and BRUTE [1] are userspace tools. The first one is able to instantiate simultaneous patterns of traffic, but it does not provide any explicit support for extensible interfaces and is not suitable to work at high rates, especially with small frames, as shown in [1].

MGEN provides both a command line and a GUI for user-friendly traffic generation in user-space. It runs on different Unix-based Operating Systems such as FreeBSD, Linux, NetBSD and Solaris, but its accuracy is limited by the system timers it exploits (e.g.: in the Linux kernel on PC-platforms, the timer resolution used by MGEN is only 10ms [3]).

The Internet Traffic Generator (ITG) [8, 9] aims to reproduce TCP and UDP traffic and replicate appropriate stochastic processes for interdeparture time and packet size. It is based on daemon processes that are contacted through Inter Process Communications by the interfaces. It is able to achieve performance comparable to that of RUDGE and MGEN but provides more traffic patterns and runs also under Windows™.

The Browny and RobUst Traffic Engine (BRUTE) [1] takes advantage of the Linux kernel potential in order to accurately generate traffic flows up to very high bit rates. Because of its excellent flexibility due to a simple script language and an extensible architecture, it has been chosen as the basis for the development of our generator BRUNO. BRUTE provides extensibility by means of optimized functions and an interface (API) which enable the implementation in C language of additional traffic sources (named T-modules) by users. For the issue of flexibility (at the expense of a slight loss in terms of latency), it uses POSIX.1B FIFO process type and has been designed as an user space application.

The work presented in this paper has been inspired by the need for a generator combining the high flexibility of PC-based tools such as BRUTE and the high performance of dedicated hardware instruments.

2.1. The Intel IXP2400 Network Processor

The Intel IXP2400 NP [12] is a multi-core processor dedicated to packet processing. It consists of nine programmable RISC (Reduced Instruction Set Computer) cores: 1 X-Scale and 8 packet processing engines (called microengines) organized in 2 symmetric clusters. Since BRUTE uses the term “micro-engine” to refer to traffic engines, in the following sections we call the IXP2400 microengines cores simply as μEs to avoid confusion. The X-Scale core (ARM V5STE compliant) supports an embedded Montavista® Linux [13] and can be programmed in C language. It is in charge of management functions and exception-handling. The μEs support 8 or 4 threads in hardware with zero-overhead context switches. Their specific set of about 50 instructions is dedicated to packet processing for the fast data-path. Intel provides an Integrated Development Environment called Developer’s Workbench [14] with a proprietary assembler (for Microcode Assembly) and a MicroC compiler for the development of μEs code. However, the microcode assembly’s support for macros and different common constructs make pro-
Also memory is organized in a hierarchical manner: the programmer can use 2.5 KB local memory per µE, a 16 KB "scratchpad" (on-chip SRAM) memory, up to 32 MB off-chip QDR SRAM (in 2 channels) and up to 2 MB DDR DRAM.

2.2. Traffic generators on the IXP2400 NP

The University of Kentucky developed IXPktgen [5], a generator based on the Intel IXP2400. Because of the lack of specific informations about this generator, it is not possible to perform a comprehensive analysis on it. Nevertheless an accurate study of its source code shows its structure. It employs 4 µEs (working in 8-threads mode) which are used for traffic generation. This implies that, since each thread is statically assigned a single flow, only 32 flows can be generated at the same time. IXPktgen is developed in microcode-assembly and can generate any kind of ethernet frames.

The Pktgen [6] is a traffic generator proposed by the University of Genova. It is based on the Radysis ENP-2611 board equipped with the Intel IXP2400 NP. It can generate Constant Bit Rate and burst traffic with high throughput. In its design, 5 µEs (working in 4-threads mode with a single flow per thread) are in charge of traffic generations. Therefore it is possible to generate only 20 flows at the same time. The code is developed in microC, whose compiler is not as optimized as the microcode-assembler (according to Intel’s guidelines [14]).

3. BRUNO

The target of BRUNO is to combine the flexibility of software-based generators with the high performances achievable only by hardware-assisted applications. Therefore in our architecture we exploit both a general purpose PC and an ENP2611 Radisys pci-board equipped with the Intel IXP2400 NP. The DRAM and SRAM memories on the board, accessible through PCI bus, set up the link between PC host and NP in terms of shared data structures.

The user interface, as well as the parsing process and the creation of flow structures are assigned to the host PC, while the actual traffic creation is committed to the IXP2400. More precisely, the host PC, through an ad-hoc modified version of BRUTE (that we simply call BRUTE in the following), computes departure times and packet lengths according to the user specifications and stores them in the DRAM. On the NP side, a µE named Load Balancer (LB) is in charge of reading data from DRAM and applying a correction algorithm on packet departure times, while 4 µEs named Traffic Generators (TGs) create packets for transmission.

3.1. Design of BRUNO

In fig. 2 the design of our solution is depicted. The first µE, represented by the tagged box on the left (Load Balancer), reads the packet timeline that BRUTE (on the PC) writes in DRAM and SRAM. Then it properly modifies and sends it to the µEs called Traffic Generators, through a ring structure. The rings are circular, fast and small FIFO queues allocated into the scratchpad memory of the IXP2400 [12]. Since the scratchpad memory is the only shared memory that is embedded in the NP, such rings represent an optimal solution for the communication among the processing units. Traffic Generators finally send packet transmission requests to the transmitter (TX) µE, which, in turn, is connected to the Load Balancer through the feedback ring.
The choice of this particular design comes from the need to overcome some limitations that the other NP-based generators have shown. The maximum number of flows that can be simultaneously generated is one of them. This is mainly due to the fixed association between flows and µE threads. For these reasons, in BRUNO a given flow is not strictly associated to a particular thread, thus obtaining an unlimited number of simultaneous flows.

Indeed, if each thread is in charge of a single flow, it is likely that some threads work more than others, or even that all threads on a certain µE work while other µEs just sleep. This is not desirable since a high number of active threads on the same µE could affect the timeliness of packets and hence the precision of the system. In BRUNO each thread processes packets regardless of flows which they belong to and the LB µE guarantees an equal balance of load among TG µEs, by distributing the packet generation requests in a round robin fashion. This way, whenever a single flow has to be generated, all the threads in the TGs can work for it.

The feedback ring is introduced in order to obtain great accuracy in traffic generation. Indeed, this mechanism makes the observed real transmission times available to the LB for a comparison with the ideal departure ones. While the packets request are kept in DRAM in a memory window that is continuously refreshed by the PC with new data, the traffic parameters (e.g. L2 and L3 addresses, as we will see later) are kept in SRAM as they need to be accessed very frequently for the creation of each packet. Both the DRAM and SRAM memory banks on the board are accessible through the local PCI bus, which, in turn, is connected to the PCI bus of the host PC through the Intel 21555 non-transparent PCI-to-PCI bridge [15]. Since the address plan referring to the two buses is different, we configure the memory translation map through some control registers associated to the 21555 bus. After the initialization of the bridge, both the SRAM and the DRAM memory banks are accessible as PCI resource regions by the host PC operating system and can be read and written using system calls referring to memory mapped I/O.

### 3.2. Load Balancer

The LB µE draws data from DRAM related to a packet generation, properly modifies its departure time and then sends all the data to a TG µE. Fig. 3 depicts the structure for a packet generation request (PR), which is loaded in DRAM by the BRUTE application running on the host PC. The first 32 bits contain the interdeparture time, the packet size (16 bits), the pointer to the flow structure in SRAM (Flow Index, 15 bits), and the IP version (IPv4 or IPv6, 1 bit) follow.

The threads of LB are divided into two groups.

- Even Threads: once departure times have been converted from "relative" into "absolute", they move PRs from DRAM to the rings in the local memory of µEs. 8 PRs are processed at a time since the DRAM is read in blocks of 16 words of 32-bits.
  - Odd Threads: they draw PRs from local memory, adjust departure times and forward the new requests to TGs. The process of departure times correction is accurately explained in section 4.

Since the timestamp counter in the TGs is limited to 16 bits, the LB should not send to TGs any PRs scheduled more than 2¹⁶ clock ticks ahead in the future. Therefore odd threads stop when the difference between the time written in the PR and present time is greater than a parameter (AWAITING_THRESHOLD), which has to be set at the start of BRUNO application. This parameter can be at most 2¹⁶ ticks. Since the timestamp counter is increased every 16 clock cycles and the µE clock frequency is set to 600 Mhz:

\[
1 \text{tick} = \frac{16}{600 \cdot 10^6} \text{seconds} = 26 \text{ns}
\]

The choice of the AWAITING_THRESHOLD must be carefully considered. In fact, low values lead to an underutilization of the Traffic Generators that may not respond properly to abrupt changes in the traffic. On the other hand, high values of this parameter can easily saturate the scratch rings between the Load Balancer and Traffic.

#### 3.3. Traffic Generators

As shown in fig. 2, 4 µEs are designed for packet creation. The thread of Traffic Generators process a packet at a time, by taking the corresponding PR from the communication ring between LB and TG. Through the field Flow Index in the PR, the structure describing the flow which the packet belongs to is accessed. Fig. 4 shows an example of such structures, which are loaded into the SRAM by BRUTE in the initialization phase, according to the user settings.

Output_Port indicates the physical output port for the flow. Protocol, TOS, Source_Port, and Destination_Port provide the corresponding fields of L3 and L4 packet headers. SRC_ADDs and DEST_ADDs point to two SRAM locations which contain a list of source and destination addresses respectively. Total provides the number of these addresses, while Index indicates the next address to be read if the choice is made in a linear way (otherwise, in random mode, the proper address is suggested by a random number generator).

<table>
<thead>
<tr>
<th>Interdeparture Time</th>
<th>Packet Size</th>
<th>Flow Index</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(31-0)</td>
<td>(31-16)</td>
<td>(15-1)</td>
<td>0</td>
</tr>
</tbody>
</table>

| Figure 3. Structure of a packet request. |
Figure 4. Structure for a flow.

The two bits of Ind_Type indicates the selection mode for both source and destination addresses (which in addition can be IP or MAC addresses).

From these data, a thread is able to create packet metadata and L2, L3 and L4 headers. Then the thread is placed in a state of sleep, until the time to send the packet arrives. At this point, the thread wakes up and sends the packet transmission request to the transmission block, which will provide to transmit the packet.

4. PERFORMANCE EVALUATION

4.1. System delays

In this section, we analyze the system behavior in order to estimate the goodness of our design in terms of performance. In particular, we will try to understand if the system is able to generate the maximum packet rate for a gigabit ethernet: 1488000 pps (with 64 bytes per packet). As stated by code simulation, the hardest workload among the µEs is in the charge of the Traffic Generators, so we focus on them.

The mean time (hereafter we call "T" the mean times) spent by a thread of a Traffic Generator µE for its overall processing of a packet is:

\[ T_c = T_{r,scr} + T_{el} + T_{w,SRAM} + T_{w,DRAM} + T_{wait} + T_{w,scr} \]  

(1)

where \( T_{r,scr} \) represents the mean time spent for reading the MPR from the scratching, \( T_{el} \) for processing the request, \( T_{w,SRAM}, T_{w,DRAM} \), and \( T_{w,scr} \) for writing metadata in SRAM, the whole packet in DRAM, and the packet transmission request in the transmission scratching respectively. Finally, \( T_{wait} \) represents the time a thread must wait when it is placed in the sleep state, as we have described above.

Now we use Little’s law to compute the available time budget for the overall processing of a packet by the TGs:

\[ T_c \cdot \lambda = n \]  

(2)

where \( \lambda \) represents the load (in our worst case 1488000 pps) and \( n \) the number of entities that take care of packet generation. In particular, in our design, \( n = n_m \cdot n_t \), where \( n_t = 8 \) is the number of threads per microengine and \( n_m = 4 \) the number of Traffic Generator µEs. With such values, we obtain a time budget for a packet of \( T_c \approx 12900 \) clks.

<table>
<thead>
<tr>
<th>Source_Port</th>
<th>Protocol</th>
<th>TOS</th>
<th>Ind_Type</th>
<th>Res</th>
</tr>
</thead>
<tbody>
<tr>
<td>(31-16)</td>
<td>(14-7)</td>
<td>(6-5)</td>
<td>(4-0)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Destination_Port</th>
<th>(15-0)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>T_{r,scr}</th>
<th>T_{el}</th>
<th>T_{w,SRAM}</th>
<th>T_{w,DRAM}</th>
<th>T_{wait}</th>
<th>T_{w,scr}</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>200</td>
<td>100</td>
<td>100</td>
<td>250</td>
<td>60</td>
</tr>
</tbody>
</table>

Table 1. Mean times for each operation in clock cycles.

Then we have measured by means of the Develop Workbench the mean times above listed: tab. 1 reports these values in terms of clock cycles. Their sum amounts to 770 clks, which is widely within the computed budget. Therefore our system looks able to support the maximum packet rate in a gigabit ethernet, and the experimental results shown in sec. 5. confirm this analysis.

4.2. Timing correction

In fig. 5 we represent a schematic view of BRUNO as a system with an input (the ideal departure time \( t(n) \) for the \( n \)-th packet) and an output (the actual departure time \( \tau(n) \)). This representation comes in handy in order to describe the actual system implementation and also the timing correction algorithm introduced in the Load Balancer µE. The introduction of a correction algorithm is motivated by the large number of phenomena that, in a complex multi-core system such as our IXP2400 NP, could affect the accuracy of the traffic generation. As an example it suffices to say that the latency of each memory access to any SRAM or DRAM is strictly dependent on the number and the state of all the other threads referring to that memory. A large variety of events (that imply memory and bus accesses) also occurs on the XScale core because of the regular OS house-keeping (e.g.: timing interrupts, memory paging and swapping). All these phenomena may affect a number of packet departures because of their duration in time. Therefore they are modeled in our scheme as a noise \( \omega(n) \) with a non-null autocorrelation. In addition, we point out that, since the noise represents a sum of different phenomena that introduce delays, its mean value is positive: \( \mathbb{E}[\omega(n)] > 0 \). Hence the reason for a correction algorithm (\( f(\cdot) \) in fig. 5).

Figure 5. Schematic view of BRUNO as a system

However, because of the limited instruction set of the µEs and to limit the delay it introduces, our error-correction algorithm must be devised to be fast and simple, requiring the
minimal amount of instructions. Therefore we choose an exponential moving average:

\[ \varphi(n) = A \cdot \varphi(n-1) + B \cdot [\hat{\Delta}(n-k) - \Delta(n-k)] \quad (3) \]

where \( \varphi(n) \) represents the correction applied to the \( n \)-th packet departure time, \( \Delta(n-k) = \tau(n-k) - (t(n-k) - t(n-k-1)) \) is the measured interdeparture time of the \( (n-k) \)-th packet (taken from the feedback scratching) and \( \Delta(n-k) = t(n-k) - t(n-k-1) \) is the ideal interdeparture time (kept in local memory) of the same packet. The parameters \( A \) and \( B \) are real and positive numbers, with \( A + B = 1 \), while the term \( k \) takes into account the feedback and system delay (shown in fig. 5 as \( z^{-k} \)). In fact, when the LB is working on the \( n \)-th MPR, there are a certain number of MPRs in the TGs and on the rings, moreover the feedback mechanism is obviously not instantaneous. Notice that the correction function is applied to the difference of interarrival time rather than on the absolute time themselves: indeed, interarrival times must be very precise while the presence of a possible constant offset between \( t(n) \) and \( \tau(n) \) is not relevant.

In the following, we assume this term \( k \) to be fixed and known and we analize the system in fig.5 as a discrete-time linear system where packet departure times define the time-domain. Notice that, dealing with a discrete time system that evolves on a packet generation basis (i.e., events are not necessarily equally spaced in time), the mathematical approach is still valid though the frequency parameter cannot be interpreted in the standad way and measured in Hertz.

By simple calculations, it is easy to derive the transfer function \( H(z) \) that describes the output of the corrector block as a function of the noise process \( \omega(n) \) as:

\[ H(z) = \frac{\varphi(z)}{\omega(z)} = \frac{Bz^{-k}(1-z^{-1})}{1-Az^{-1} + Bz^{-k}(1-z^{-1})} \quad (4) \]

According to figure 5, and by indicating the impulse response of the system \( H(z) \) with \( h(n) \), one has:

\[ \tau(n) = t(n) - \varphi(n) + \omega(n) \]
\[ = t(n) + \omega(n) - h(n) \odot \omega(n) \quad (5) \]
\[ = t(n) + \epsilon(n) \]

The error term \( \epsilon(n) \) associated with the absolute generated times can be calculated as the output of the system:

\[ L(z) = 1 - H(z) = \frac{1-Az^{-1}}{1-z^{-1} + Bz^{-k}(1-z^{-1})} \quad (6) \]

which receives as an input the sequence of noise \( \omega(n) \).

As above mentioned, though, the error sequence of interest is that of the interarrival time, that is:

\[ \hat{\Delta}(n) - \Delta(n) = \tau(n) - \tau(n-1) - (t(n) - t(n-1)) \]
\[ = \epsilon(n) - \epsilon(n-1) \]
\[ = \epsilon(n) \quad (7) \]

In other words, we can express the sequence of errors of interarrival times \( \epsilon(n) \) in terms of the noise process \( \omega(n) \) through the transfer function of the equivalent system:

\[ G(z) = (1-z^{-1})L(z) = \frac{(1-Az^{-1})(1-z^{-1})}{1-Az^{-1} + Bz^{-k}(1-z^{-1})} \quad (8) \]

The effectiveness of the timing correction mechanism can then be evaluated through the characteristics of the equivalent system \( G(z) \).

By assuming the noise process as wide sense stationary, it turns out that the average error is null:

\[ \mathbb{E} [\epsilon(n)] = \mathbb{E} [\omega(n)] \cdot G(1) = 0 \quad (9) \]

and that its power spectral density \( S_{\epsilon}(f) \) is given by:

\[ S_{\epsilon}(f) = |S_{\omega}(f) |^2 \quad (10) \]

**Figure 6.** Energy of the impulse response of \( G(z) \).

In lack of any statistical information on the noise, the selection of parameters \( A \) and \( B = 1-A \) should be made in order to minimize the energy of the system \( G(z) \) so as to minimize the variance of the error \( \epsilon(n) \) in the case of flat spectral density of the noise process.

Figure 6 shows the energy of the system \( G(z) \) with respect to \( A \) for several values of \( k \). From this figure, the choice of \( A > 0.5 \) seems to be suitable in that the energy approaches 1 and it is very little sensitive with respect to \( k \).

In our experiments (figure 7), though, the noise \( \omega(n) \) turns out to be colored as it exhibits stronger components at low frequencies (notice that the peak at \( f = 0 \) is mainly due to the non zero mean of \( \omega(n) \)). As shown in figure 8 shows, the frequency response \( G(f) \) evaluated for \( A = 0.75 \) and \( k = 12 \) (the maximum value of \( k \) observed in our experiments) indeed proves a clear high pass behavior, thus effectively filtering out the low frequency relevant components of noise.
5. EXPERIMENTAL RESULTS

We evaluate the actual performance of BRUNO through a wide variety of experimental tests.

In the first set of experiments, the benefits introduced by the error correction mechanism are investigated. We instruct BRUNO to generate CBR traffic flows within a wide range of bit rates, spanning from 100 to 600 Mbps, and we run, for each value of bit rate, both the standard version of BRUNO and a modified version in which the feedback correction mechanism was disabled. The measurements are taken by means of the Spirent AX4000 traffic analyzer, which is an ASIC-based tool supporting a precision of the order of nanoseconds. Table 2 reports the reduction in the interdeparture time variation due to the introduction of the correction mechanism. We have measured the standard deviations of interdeparture times ($\sigma_e$ for the system with error correction, $\sigma_{ne}$ for the simple one), obtaining for both versions extremely small values (in the order of hundreds of nanoseconds). However, the use of the timing correction mechanism increases the performance of the system. In any case, these benefits will likely be more evident with increasing complexity of the traffic model generated. In fact with a CBR model there are few variable factors that can cause a consistent deviation from the ideal behaviour, and, therefore, the variance reduction achievable with an error correction mechanism is limited.

We also compare the average interdeparture times of packets generated by BRUNO with the theoretical values. The results, plotted in figure 9, show very little differences, always smaller than 26 ns. Since within the Network Processor the time is expressed in terms of ticks and 1 tick $\approx 26$ ns (as shown in sec. 3.2.), such a difference can be explained as a quantization error.

Finally, we compare the performance of BRUNO with those of BRUTE. We chose BRUTE as a benchmark because, as we said, it is the foundation of BRUNO and outperforms the other well known software traffic generators. The improvement of BRUNO over BRUTE is evident from the figure 10, which reports the short term packet rate (calculated as a mean over intervals of 0.10 s): BRUNO is able to generate up to 1488000 pps with a high precision.

All these results seem to justify the implementation of a traffic generator on hardware platforms as Network Processors. Moreover, they confirm the goodness of our design and our implementation choices.

<table>
<thead>
<tr>
<th>rate (Mb/s)</th>
<th>$\sigma_{ne} - \sigma_e (ns)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>14.3</td>
</tr>
<tr>
<td>200</td>
<td>12.4</td>
</tr>
<tr>
<td>300</td>
<td>3.9</td>
</tr>
<tr>
<td>400</td>
<td>3.1</td>
</tr>
<tr>
<td>500</td>
<td>14.3</td>
</tr>
<tr>
<td>600</td>
<td>13.1</td>
</tr>
</tbody>
</table>

Table 2. Interdeparture time variation reduction achieved by the correction mechanism.
6. CONCLUSIONS AND FUTURE WORK

The paper presents the design and the implementation of a novel traffic generator. Our solution adopts a cooperative PC/NP architecture for overcoming the limits of PC-based tools while maintaining their flexibility. More precisely, this work is based on BRUTE, a very flexible and high performance software, and on the processing power of an Intel IXP2400.

In our design, BRUTE, running on a host PC, computes packet lengths and departure times, while in the NP-side a Load Balancer μE properly rectifies such times and the Traffic Generator μEs are in charge of real packet generation.

![Figure 10. Comparison between Brute and Bruno.](image)

For the issues of precision and accuracy in traffic models reproduction, a feedback mechanism and a time correction scheme are introduced and carefully analyzed: the transmission μE reports in a feedback ring the actual packet departure times, which are then used by the Load Balancer for an adaptive time modification.

The experimental results have shown the goodness of our architecture and the usefulness of the time correction mechanism: we are able to generate up to 1.5 Mpps with a great precision, outperforming the best available software tools.

In the future, we plan to test different scheduling policies for the Load Balancer and to refine the time correction scheme. Moreover, in order to push the system towards its upper limiting performance and to achieve the highest possible sustainable packet generation rates, we plan to port our code onto more powerful Network Processor platforms.

REFERENCES


