A High Performance IP Traffic Generation Tool Based on the Intel IXP2400 Network Processor

Raffaele Bolla, Roberto Bruschi, Marco Canini, and Matteo Repetto

Department of Communications, Computer and Systems Science,
DIST–University of Genova, Via Opera Pia 13, 16145 Genova, Italy
{raffaele.bolla, roberto.bruschi}@unige.it, marco@reti.dist.unige.it,
matteo.repetto@unige.it

Abstract. Traffic generation is essential in the processes of testing and developing new network elements, such as equipment, protocols and applications, regarding both the production and research area. Traditionally, two approaches have been followed for this purpose: the first is based on software applications that can be executed on inexpensive Personal Computers, while the second relies on dedicated hardware. Obviously, performance in the latter case (in terms of sustainable rates, precision in delays and jitters) outclasses that in the former, but also the costs usually grow of some order of magnitude. In this paper we describe a software IP traffic generator based on a hardware architecture specialized for packet processing (known as Network Processor), which we have developed and tested. Our approach is positioned between the two different philosophies listed above: it has a software (and then flexible) implementation running on a specific hardware only slightly more expensive than PCs.

1 Introduction

The current Internet is characterized by a continuous and fast evolution, in terms of amount and kind of traffic, network equipment and protocols. Heterogeneous equipment, protocols and applications (also referred to as "network elements"), high transmission rates in most of Internet branches and the need of fast development (to quickly fulfil new services' requirements and to reduce time to market) are at present the most critical issues in Internet growth. In this context, every network element should be carefully tested before being used in real networks: an error in a device or a protocol could easily result in technical problems (e.g., packet losses, connection interruptions, degradation in performance) and it could lead to significative economic damage in production environments.

Many of the tests on a new network element require the ability to generate synthetic traffic off-line; such traffic should be sufficiently complex, fast (high-rates) and, in a word, realistic to cover a good deal of operating conditions. Thus, the possibility to create realistic network traffic streams is very useful today, and it is essential to reduce development times and bugs in new network elements. Moreover, the availability of good synthetic traffic sources helps researchers in

understanding network dynamics and in designing suitable modifications and improvements in current protocols and equipment.

The generation of traffic streams is a quite "simple" task in simulated environments; on the contrary, the generation of real traffic on a testbed setup involves some critical issues related to precision, scalability and costs. Until now, two antithetical approaches have been traditionally followed for network traffic generation. The first one is based on software applications that do not require specific hardware, but can be executed on general purpose machines, such as inexpensive Personal Computers (PC). The second one is based on the development of specialized hardware.

The characteristics and functionalities of software tools can be very sophisticated but, at the same time, they can maintain a high flexibility level and (often) offer an open source approach: in other words, they are modifiable and adaptable to the specific requirements of the experiments to carry out. The main drawback of this approach is the architecture of the PC, which limits the precision and the maximum reachable performance; this is a heavy limitation today, with Internet traffic that increases continuously and network equipment that must manage a huge amount of traffic (from Gigabits to Terabits per second).

Several techniques can be used in PC-based network traffic generation, depending on the final goal. Quite often, it is useful to generate only a single stream of IP packets, characterized by the inter-arrival time between two consecutive packets; this stream can be used to test network equipment such as routers and switches with standard performance analyses, as those defined in RFC 2544 [1] and 2889 [2]. Examples of applications oriented to this type of generation are *Iperf* [3] and *Netperf* [4]. These two tools generate IP packets by starting from a fixed structure and by varying some fields (e.g., the source and destination addresses, the TOS, etc.); more advanced applications are able to keep into account also the behaviour of the transport protocols, by representing the traffic generation as a set of file transfers over TCP or UDP (see, for example, *Harpoon* [5]). All these tools do not care about packet contents; however, when the element under test is an application (such as a Web server or a DNS), this is not acceptable. Thus, other tools have been designed with this goal, for example *WebPolygraph* [6]. One common drawback of the above cited software tools is that each of them can generate only a specific type of traffic stream, which is not representative of the heterogeneous traffic flowing on a generic Internet link. Software such as *Tcpreplay* and *Flowreplay* [7] log the traffic on some links and then generate identical streams, by possibly changing some parameters in the IP packets. Unfortunately, in this case, the traffic streams are always the same and the registration of long sequences of traffic requires large amounts of fast memory.

The second approach, namely the custom hardware devices, is the most popular solution in the industrial environment. Suitable architectures have been designed (often by using also standard components) to minimize delays and jitters introduced in packet generation; here all the functionalities are implemented "near" the hardware, without the intermediation of a general purpose operating

system (that may be present at a higher level to facilitate the configuration and the control of the instrument and the management of the statistical data). The development of such equipment is usually very expensive, based on proprietary solutions and carried out for professional and intensive use only. Thus, the final result is that the tools are configurable, but not modifiable or customizable, and very expensive. All these elements, together with the high final cost of each device, make this kind of tools not much attractive, especially for academic researchers or small labs. Examples of such products are Caldera Technologies *LANforge-FIRE* [8], CISCO IOS *NetFlow* [9], Anritsu *MD1231A IP/Ethernet Analyser* [10] and SmartBits *AX/400* [11].

Starting from these considerations, we have decided to build an open source traffic generator that could be situated in the middle between the existent approaches: it should be flexible and powerful enough to be useful in most of the practical situations, but less expensive than professional equipment. To realize all these objectives we need to work near the hardware, but we also need to implement all the functionalities in software, to reduce development costs and make customization effective. Among different technologies we have taken into account, we found our ideal solution in the Network Processors: these are devices conceived for fast packet processing, often with the high degree of flexibility needed for implementing the desired algorithms; moreover their cost is lower than custom hardware devices.

Several Network Processor architectures are available from different manufacturers [12]; we have found the Intel IXP2400 to be the best compromise between computation power and flexibility. Moreover, this chip is available on an evaluation board, the Radisys ENP-2611, which represents the cheapest way to access this kind of Network Processors. Until now, we know only another similar approach to the problem of traffic generation [13], but it implements a very simple structure, which is unable to generate realistic traffic. In this work we describe a structure for a high-speed IP/Ethernet traffic generator based on the Intel IXP2400 Network Processor that we have developed and tested. The requirements of this device are to be able to transmit more than 1 Gbps of traffic (to saturate all the Gigabit Ethernet lines of the ENP-2611) and to generate multiple traffic streams simultaneously, each of them with different statistical characteristics and header contents. Our aim is to model the more representative traffic classes in the Internet, such as real-time and Best Effort; moreover, we would like to have the chance to manage Quality of Service (by using the TOS field).

In the following, after briefly describing the IXP2400 architecture, we focus on two main issues. The first concerns the question if the Network Processor can be effectively utilized for traffic generation: the IXP2400 has been designed for packet processing, not explicitly for packet generation, and we were not sure that it can saturate the Gigabit interfaces, as we would like for a high-performance tool. The second issue regards the design of an application framework that exploits the Network Processor architecture to build a flexible tool for traffic generation. We have named such framework PktGen and, as will described in detail

later, it consists of several applications running on different processors of the IXP2400. PktGen is very simple to use, as it provides a comfortable user graphical web interface; moreover, it can also be modified without much effort, as it is mostly written in C. Preliminary tests have been carried out to demonstrate the correcteness and the performance of such tool.

The rest of the paper is organized as follows: Sect. 2 gives a brief overview of the Intel IXP2400, whereas the successive Sect. 3 explains how the Network Processor architecture is used to build a packet generator. Section 4 describes PktGen in details by analysing all its components, whereas in Sect. 5 we report some performance tests about PktGen and a comparison with UDPGen, a well-known software PC-based traffic generator. Finally, in Sect. 6 we give our conclusions and we report some ideas for future work.

2 The IXP2400 Architecture

Figure 1 shows the main components of the Intel IXP2400 architecture and the relationships among them. The IXP2400, as some other Network Processors, provides several processing units, different kinds of memory, a standard interface towards MAC components and some utility functions (e.g. hash and CRC calculation). The "intelligence" resides in the processing units, which can use the other peripherals though several internal buses.

For what concerns processing, the IXP is equipped with two kinds of microprocessors which play very different roles in the overall architecture: one XScale CPU and eight MicroEngines. The XScale is a generic RISC 32-bit processor, compatible with the ARM V5 architecture, but without the floating point unit. This processor is mainly used to control the overall system and to process network control packets (such as ICMP and routing messages). Due to its compatibility with the industry based standards (ARM), it is possible to use on it both the *Linux* and *VxWorks* [14] (derived from *Windows*) Operating Systems (OS); this is a great advantages as many existent applications and libraries can be compiled and used on this processor (greatly decreasing development time and increasing efficiency). For what concerns Linux, two different distributions are available at the moment: *Montavista* [15, 16], based on a 2.4.18 kernel, and *Fedora* [17], with a more recent 2.6.9 kernel.

Accordingly to our aim of building an open-source tool, we have opted for Linux OS; in particular we have chosen the Montavista distribution, which was already available at the beginning of the project (Fedora has been released only recently). This choice enables us to develop software for this processor in standard C or C++ languages.

The other processing units of the IXP are the Microengines (ME). They are minimal RISC processors with a reduced set of instructions (about 50), optimized for packet processing; they provide logical and arithmetic operations on *bits*, *bytes* and *dwords* but not the division nor the floating point functions. The MEs have access to all the shared units (SRAM, DRAM, MSF, etc.) and they are used along the *fast data path* for packet processing; they can be used

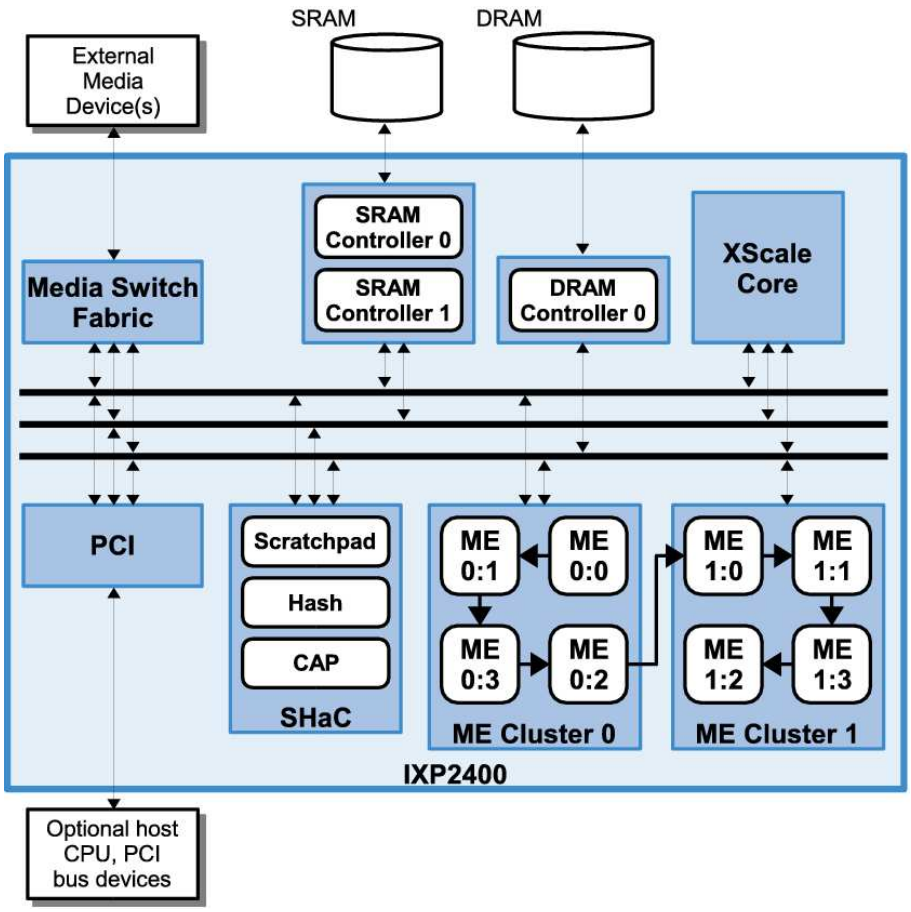


Fig. 1. The IXP architecture.

in different ways (e.g., in parallel or sequentially) to create the framework that best fits the computational needs of network equipment.

No operating system is required on the ME, thus no software scheduler is available for multi-threaded programming. A simple round robin criterion is used to execute more threads (up to 8) on a ME; the programmer has the burden to write the code for each thread in a way that periodically releases the control of the ME to the other ones. Internally, each ME is equipped with specific registers for at most 8 hardware contexts (corresponding to the threads) and a shared low latency memory; moreover, some dedicated units are available for specific tasks (CRC computation and pseudo-random number generation).

ME programming can be made by assembler language or by microC; the latter has the same instruction set as C plus some non-standard extensions,

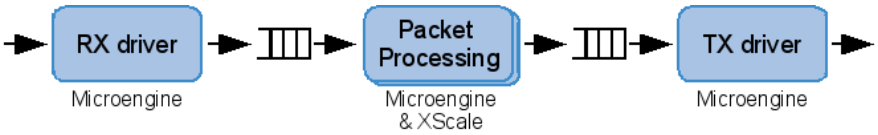


Fig. 2. A standard IXP2400 fast-path.

and it makes the learning of the language itself and the programming of the MicroEngines simpler and faster.

The IXP2400 can use three types of memories: scratchpad, SRAM and DRAM. The scratchpad has the lowest latency time, it is integrated into the IXP2400 chip itself, but it is only 16 KB in size. The main feature of this memory is the support for 16 FIFO queues with automatic get and put operations, very useful to create transmission requests for a transmission driver.

The SRAM and the DRAM are not physically placed internally to the IXP-2400, but the NP chip provides only the controllers. The SRAM has a lower latency than the DRAM, but its controller supports up to 64 MB against 1 GB of the DRAM one. DRAM is usually used for storing the packets, while SRAM is reserved for the packet metadata, that are smaller but need to be accessed more frequently than packets themselves.

The last notable component is the *Media Switch Fabric* (MSF), that is an interface to transfer packets to/from the external MAC devices. Inside the IXP2400, the MSF can access directly the DRAM, resulting in high performance in storing/retrieving packets, while the external interface can be configured to operate in the UTOPIA, POS-PHY, SPI3 or CSIX modalities.

Finally, we can mention the hash unit, the PCI unit (used to provide a communication interface towards a standard external PCI bus) and the *Control and status register Access Proxy* (CAP), for the management of the registers used in the inter-process communication.

In our environment the IXP2400 is mounted on a Radisys ENP-2611 development board, that includes three optical Gigabit Ethernet ports (for fast-path data plane traffic), SRAM e DRAM sockets, a PCI connector (to access the IXP2400 SRAM and DRAM modules) and a further FastEthernet port for "direct" communications with the XScale processor.

A more detailed knowledge of the architectures of the IXP2400 and the Radisys development board is useless for the purpose of the paper. On the contrary, it is interesting to analyse how a standard packet processing phase takes place in the fast-path of the Network Processor.

Figure 2 shows a standard fast-path structure: packets are received from one Ethernet interface by a RX driver, passed to the Microengines (and eventually to the XScale) for the required processing and finally delivered to the TX driver for transmission over the physical line. The events that occur inside the Network Processor during this path crossing are the following:

1. A packet is received from the physical interface and delivered to the MSF.

2. The MSF buffers the packet and it notifies the RX driver, running on a MicroEngine.
3. The RX driver commands the MSF to transfer the buffered packet in the DRAM; then it creates a metadata structure for that packet, which is inserted in the SRAM. An identificator of the packet is put in a specific ring queue of the scratchpad, for successive processing.
4. A Microengine thread gets the packet identificators (queued from the RX driver) and starts the packet processing phase. The operations that occur on the packet are application-specific and they may range from a simple routing between ports to more advanced features, such as *firewalling*, *natting*, etc; these operations can be carried out by a single thread or by multiple threads (each of them should perform a simple task), and sometime by the XScale. The latter is usually involved very rarely (e.g., it processes the routing update packet exchanged between routers or the control information directed to the Network Processor), as the presence of an operating system enables more complex operations, but results in slower execution times. During this phase the data are stored in the SRAM and both the packet data and metadata could be modified; finally the packet identifier is inserted in a ring queue for transmission (again in the scratchpad memory).
5. The transmission driver continuously checks the transmission ring queue, looking for new identifiers of the packets to be transmitted; each time it finds a new identifier, it gets the metadata from the SRAM and it instructs the MSF for transmission.
6. The MSF gets the packet from the DRAM and transfers it to the physical interface for the transmission.

3 Traffic Generation

In order to generate network traffic, the very basic tasks that we need to realize are the creation and transmission of the packets. The first issue to solve is the choice of where to locate the functions for creating and then transmitting the packets; in particular the two alternatives are the Xscale processor, which offers developers a well known environment and programming language, or the Microengines, faster for this task but less easily programmable.

In this context we have carried out several tests, in which the same algorithm (which transmits the same 60[1] bytes sized packet for a given number of times) has been implemented as:

1. a kernel space application running on XScale, which has been developed by using the Resource Manager[2] library;

[1] The actual packet size is 64 bytes, but the last 4 bytes are the CRC computed and appended by the PM3386 MAC device located on the Radisys board.

[2] Resource Manager is part of Intel IXA Portability Framework; although the IXA is not supported by the Radisys ENP-2611, in this case we succeeded in using part of it for our code.

Table 1. Comparative results for different generation methods.

Generation method	***Average pps[3] rate***
From XScale - kernel mode	468 kpps
From XScale - user mode	658 kpps
From Microengine	842 kpps

2. a user space application running on XScale, which uses the "*mmaped*" memory to access hardware features;
3. a microcode application running on a Microengine.

In the first and second case the algorithm has been implemented as C code and compiled by using the available GCC compiler with all the optimizations enabled. The effort required to develop a kernel module is greater then that required to build a user space application. However, given the availability of the Resource Manager library, the overall readability of code results enhanced with respect to the direct mapped memory usage in the user space application. On the other hand the Resource Manager library is specifically engineered for control plane tasks, thus it is not optimized to handle the transmission of packets efficiently.

For the third approach we have chosen to implement the algorithm in microC language, which is certainly easier than microengine assembly, while it preserves the same potential strength. In this case we were faced with a new programming model and payed extra time to get a base skill for it. Indeed, the results (Table 1) were not fulfilling the expectations and the attempts to tune the code did not give effective performance gains.

The main difference between generation from XScale (in both ways) and from Microengines is that in the first case we write a packet in DRAM for each transmission, while in the second case we always use the packets already present in the DRAM: actually, the high latency times of this kind of memory prevent the XScale code to keep up with higher packet generation rates.

For our purpose, it is more important to send similar packets at high rates rather than sending potentially completely different packets at low rates. Thus, we choose to have a set of packets (in the simplest case only one) always present in memory and to use Microengines to transmit each time one of the available packets. With this approach we can reach the maximum physical rate for a single port: 1.488 kpps, which for a packet size of 64 bytes is equivalent to 1 Gbps.

Obviously, transmitting the same packet can be of limited interest (especially for what regards header field contents, such as the source/destination addresses and the TOS); moreover, to store a great number of packets differing only for a few fields (or some combination of them) is not a clever solution. Thus, we introduced the concept of packet template to indicate a common structure of

[3] Packets per second

packet with a few fields that can be assigned dynamically at each transmission (according to some rules or some predefined sets of values), while the others have a fixed value.

We can make the concept of packet template clearer with an example. Suppose we want to generate a stream of UDP packets. Now suppose that we assign a fixed value to the source IP address and leave the destination IP address unspecified, since we have a list of possible destination addresses. Without the template mechanism we would create a duplicate of the same packet for each destination IP address; instead, with the template mechanism, only one packet is buffered and the list of possible destination IP addresses has to be passed to the packet generator software. The same software, at run time, will put one of those addresses in the packet's destination IP address field. In this way, a lot of memory has been saved and can be used for other packet templates.

The great benefit of this model is that we can reach the highest transmission rate while preserving all the flexibility required for a network traffic generator. The disadvantage is that all packet templates need to be available in memory before the generation can start.

This implies that the available physical memory represents an upper bound to the number of usable packet templates, but this is not a great issue, since the size of DRAM memory is large enough to store an amount of packet templates adequate for the characterization of many different traffic streams (the actual number of packets depends on their size).

4 The PktGen Framework

The IXP2400 architecture is well suited to be used to create a network traffic generator framework, including both a packet generator engine and a flexible and intuitive configuration interface. We have realized a software tool including three main components (see Fig. 3):

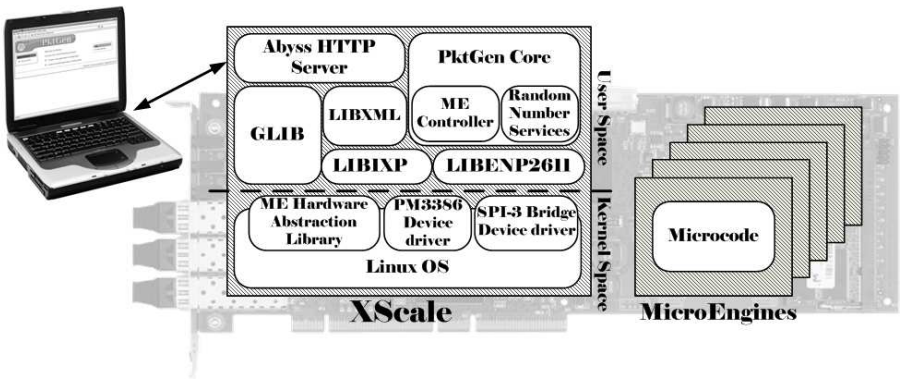


Fig. 3. Software architecture of PktGen.

– the packet generator, an application written in microC and running on the MicroEngines, whose main task is the generation of packets with predefined statistical characteristics; using multiple instances of the application it is possible to simultaneously generate a set of traffic streams with homogeneous or heterogeneous characteristics;
– the PktGen core controller, an application running on the XScale, with the goal to initialize and control the packet generators on the MEs, including the packet templates to be used;
– a graphical interface, HTML based and accessible by a standard web browser, used to easily configure the packet generator parameters.

At present PktGen is able to generate two kinds of traffic, which we consider very significant in testing IP-devices; they can be identified as *Constant Bit Rate* (CBR) and *Best Effort* (BE) or *bursty* streams.

For the BE streams we use a bursty model in which packets are generated in random-sized bursts with random inter-arrival times. The statistics of the two random variables can be modified quite simply, by passing the related probability density functions to PktGen; in our tests, we have used an exponential density function for both the inter-arrival times and the burst sizes. We think to introduce more kinds of traffic in the future; the structure of PktGen is flexible enough to perform this task in a simple and quick manner.

The Packet Generator The packet generator is a microC code compiled to run on a single MicroEngine in 4-threads mode. This means that actually there are 4 instances of packet generators concurrently executing on the same MicroEngine. Multiple instances of packet generators can run on more MicroEngines without conflicting with one another; in our configuration 5 MicroEngines are reserved for the packet generation task, thus we are free to simultaneously use up to 20 packet stream generators. Indeed, up to 7 MicroEngines can be used for packet generators; one ME must be reserved to the TX driver.

This PktGen component has been engineered to efficiently handle the generation of packets for both CBR and BE traffic models. Independently of the traffic model, the packet generator provides a setup and an operational interface controlled by the core component. The references to packet templates are forwarded to the packet generator through the setup interface, while the operational interface is used to control the generator state (running, stopped).

The two main tasks of the packet generators are the insertion of the variable fields in packet templates and the generation of random variables for statistical traffic characterization (inter-arrival times and burst sizes). At present, we have chosen to work with 4 variable fields in the packet template: Source and Destination IP addresses, TOS value and Total Length. Indeed, there is a fifth dynamic field, the Header Checksum, but it is defined indirectly by the values of the other fields. Moreover, each of the four instances running on the same MicroEngine is able to store up to 160 packet template references: thus, we can obtain a great number of packet patterns that can differ, for example, for payload contents (protocol type and data).

To enable statistical traffic characterization, a hardware uniform pseudo-random number generator is used in conjunction with a set of off-line samples for an arbitrary probability distribution function (that are evaluated by the core controller at setup time, as explained in the following) to dynamically generate values for a given probability density function.

The Core Controller The core controller component of PktGen running on the XScale processor is built in C language. As shown in Fig. 3, this component is located on top of the software stack that starts from the Linux kernel. The two major sub-components are the ME Controller and Random Number Services. The former allows the PktGen Core Controller to start, stop and setup packet generators, by encapsulating all the hardware details (features coming from kernel modules as ME Hardware Abstraction Library, PM3386 and SPI-3 device drivers) and by interfacing with the user space libraries provided with the IXP2400 (*libixp* and *libenp2611*, see Fig. 3). The latter is dedicated to handle all the mathematics involved in computing the samples for a given probability density function that are passed to packet generators and used at run time to give a statistical characterization to each traffic stream.

The flexibility of changing the probability density function is one of the main features of PktGen; however, the lack of a floating point unit and the scarceness of memory make the random number generation one of the most critical issues of the entire framework. For this reason a few more words on this topic are needed.

By using the graphical interface the user can specify the analytical formula of any desired density function; the core controller can compute the distribution function (by means of the *libmatheval* [18] and *gsl*[4] [19] libraries) used to find a set of values with the inversion method [20]. Such values are first converted into an integer representation (despite the lack of the floating point unit the XScale can work with this kind of numbers by means of software libraries, but the MEs cannot) and then passed to the packet generator that randomly picks up one of them with a uniform distribution.

The set of values representative of the desired probability density function must be stored in the SRAM memory, because the latency of the DRAM is too high. Unfortunately, the SRAM on the Radisys board is only 8 MB; thus we cannot store a great number of probability samples; the user can choose to use 256 or 64K samples (corresponding to one or two byte per sample).

Clearly, the integer conversion and the use of a limited number of samples (up to 64K) introduce a precision loss in the final traffic statistical characterization; we are still investigating the effects on the traffic generation and looking for techniques to minimize the errors introduced.

The Graphical Interface A simple, yet powerful, WEB-based interface is provided to facilitate the use of PktGen. This interface allows the user to create

[4] Gnu Scientific Library

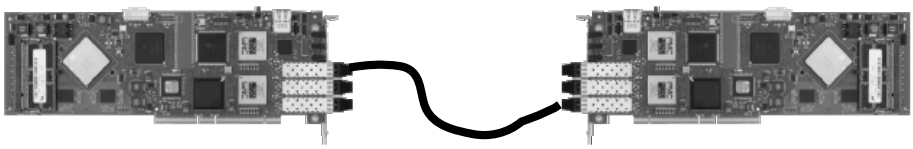


Fig. 4. Testbed 1: Measuring the maximum performance of PktGen.

a different configuration for each desired traffic profile, to save the configuration for later reuse and to run simulations.

To ensure fast code development and maintenability we have used the Abyss HTTP server to provide the graphical interface, while to conform to standard file formats used by many existent tools we have adopted an XML coding of the configuration files.

5 Results

The main result of our activity is PktGen itself: it is a proof that our initial objectives were feasible and we succeeded in achieving them. As a matter of fact, we were able to fully saturate all the three Gigabit interfaces of the Radisys board.

Nonetheless, we are interested in a thorough evaluation of the performance of our tool, expecially for what concerns precision in generation (conformance of packet flows to the statistical description); most of these tests are planned for the immediate future, but some of them have been already carried out and can give an idea of the potentialities of the framework.

The main difficulties we have met concern our lack of a measurement powerful enough testing our generator. The PC architecture cannot sustain the packet rate from PktGen and thus no software tools can be used as meters; indeed, we would need hardware devices that are currently out of the project's budget.

Thus, we have carried on only a few simple measurements on PktGen performance, and to do this we have written a simple packet meter in microC to run it on one MicroEngine. This tool can measure the mean packet rate for any kind of traffic and the jitter for CBR traffic only.

Our measurement tests have been devoted to two main issues: the first was to evaluate the PktGen maximum performance (see Fig. 4), and the second was to compare PktGen with a software tool running on a high-end PC (Fig. 5).

The PktGen maximum performance has been evaluated by means of the testbed shown in Fig. 4, where PktGen runs on one Network Processor and the other is used as the meter. In the worst case of minimum packet size (64 bytes), as reported in Fig. 6, we were able to generate a maximum Constant Bit Rate traffic of 1488.096 kpps, corresponding to 1 Gbps, with a high level of precision: the maximum measured jitter is below 2% of packet delay interarrival time in the CBR stream.

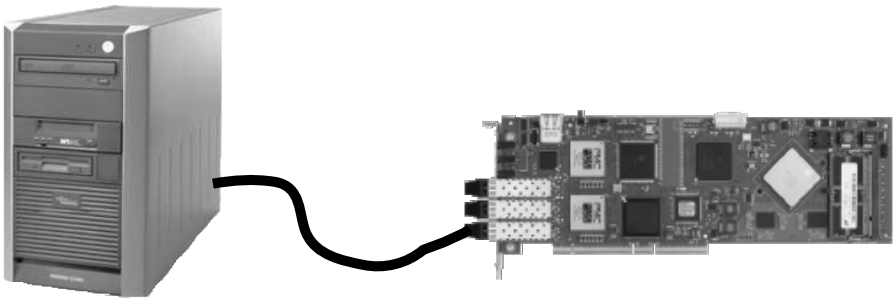
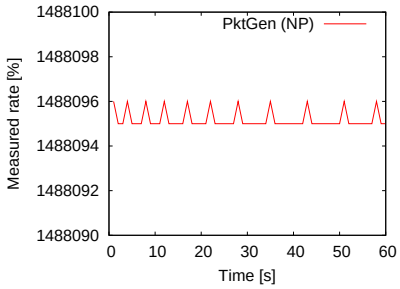
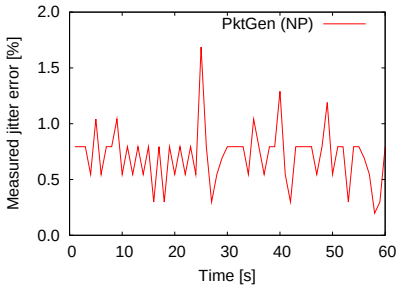


Fig. 5. Testbed 2: Comparison with generation from UDPGen.



(a) Measured generation rate over 60 s.



(b) Delay variations (jitters) at peak rate.

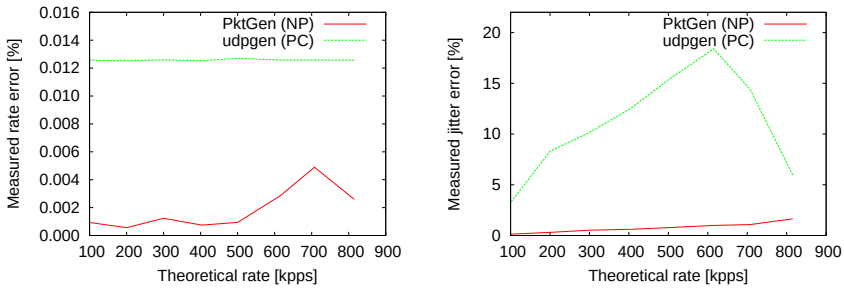
Fig. 6. Maximum performance of PktGen.

In the second testbed, we compared PktGen with a well-known software tool, namely UDPGen. The latter is used to transmit UDP packets and thus is quite similar to our application, which, however, works at the IP layer. We can see from Fig. 7 how PktGen overcomes UDPGen in terms of precision for what concerns both mean rate and jitter. Again, tests have been carried out for CBR traffic in the worst case of 64 byte packet size. Finally, it is worth noting that this analysis is limited to a maximum mean rate of 814 kpps, because this is the peak rate obtainable with UDPGen (whereas PktGen can reach about 1488 kpps).

6 Conclusions and Future Work

We have shown that high-performance traffic generation with a Network Processor (in particular with the Intel IXP2400) is possible.

The main result of our activity, namely PktGen, is an open source framework written mostly in C and is easily customizable by everyone; also the microcode



(a) Error of measured rate vs desired rate.

(b) Jitter at different desired rates.

Fig. 7. Comparison between PktGen (running on the Network Processor) and UDPGen (running on a high-end PC) with Constant Bit Rate traffic.

used in the MicroEngines is quite intuitive and simple to learn. PktGen is also able to fully saturate the three Gigabit interfaces of the development board (the ENP-2611), reaching a maximum aggregate packet generation rate of about 4464 kpps (3 Gbps).

We can therefore conclude that our aim is completely achieved: we have demonstrated how it is possible to build an open source, customizable, high-speed and precise packet generator without specific hardware; the cost is less expensive than that of a professional device (the Radisys ENP-2611 costs about $ 5000).

We consider this only a first important step in this field: in fact, we think that more work has to be done in this direction.

First of all we have planned to extend the PktGen functionalities to include also a packet meter, able to collect detailed and precise information on network traffic streams (mean rate, jitter, packet classification, etc.); this is necessary to have a counterpart to the generator, with the same noteworthy characteristics (open source, low cost, high performance and precision).

The second important goal is to port PktGen to a more performing architecture, for example to the IXP 2800, in order to realize a more and more powerful traffic generator.

Other minor evolutions are foreseen, such as the introduction of additional traffic shapes in PktGen, the comparison with professional devices and its utilization in our research activities on Quality of Service, high-speed networks, and Open Router architecture [21].

7 Acknoledgements

We would like to thanks Intel for its support to our work. Intel gave us the two Radisys ENP-2611 development board with IXP2400 Network Processor that we have used for the development of PktGen and have funded our research in the last year.

References

1. Bradner, S., McQuaid, J.: Benchmarking methodology for network interconnect devices. RFC 2544, IETF (1999) Available online, URL: http://www.ietf.org/rfc/rfc2544.txt.
2. Mandeville, R., Perser, J.: Benchmarking methodology for LAN switching devices. RFC 2889, IETF (2000) Available online, URL: http://www.ietf.org/rfc/rfc2889.txt.
3. Tirumala, A., Qin, F., Dugan, J., Ferguson, J., Gibbs, K.: Iperf. Available online, URL: http://dast.nlanr.net/Projects/Iperf/ (2005)
4. Jones, R., Choy, K., et al.: Netperf. Available online, URL: http://www.netperf.org/ (2005)
5. Sommers, J., Barford, P.: Self-configuring network traffic generation. In: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement 2004, Portland, OR - USA, IMT 04 (2004)
6. Rousskov, A., Wessels, D.: Web Polygraph. Available online, URL: http://www.web-polygraph.org/ (2004)
7. Turner, A.: Tcpreplay. Available online, URL: http://tcpreplay.sourceforge.net/ (2005)
8. Caldera Technologies, C.: LANforge-FIRE. Available online, URL: http://www.candelatech.com/ (2005)
9. Cisco Systems, C.: IOS netflow feature. Available online, URL: http://www.cisco.com/en/US/tech/tk812/tsd_technology_support_protocol_-home.html (2005)
10. Anritsu, C.: IP/Ethernet analyser, Model: MD1231A. Available online, URL: http://www.eu.anritsu.com/products/default.php?p=97&model=MD1231A (2005)
11. Smartbits: AX/400. Available online, URL: http://www.netcomsystems.com/ (2005)
12. Comer, D.E.: Network Systems Design using Network Processors - Agere version. Pearson Prentice Hall, Upper Saddle River, New Jersey - USA (2005)
13. University of Kentucky, L.f.A.N.: IXPKTGEN project. Available online. URL: http://protocols.netlab.uky.edu/ esp/pktgen/ (2004)
14. Wind River, C.: Wind River Operating Systems. Available online, URL: http://www.windriver.com/products/device_technologies/os/ (2005)
15. Montavista: Montavista Linux Preview kit. Available online, URL: http://www.mvista.com/previewkit/index.html (2004)
16. Montavista: Montavista Linux Professional Edition. Available online, URL: http://www.mvista.com/products/pro/ (2004)
17. Buytenhek, L.: Port of Fedora for XScale processor". Available online, URL: http://skrybele.wantstofly.org/ (2005)

18. GNU: Libmatheval library. Available online, URL: http://www.gnu.org/software/libmatheval/ (2005)
19. GNU: Gsl library. Available online, URL: http://www.gnu.org/software/gsl/ (2005)
20. L'Ecuyer, P. In: Random Number Generation. Handbook of Computational Statistics. Springer-Verlag (2004) pp. 35–70
21. Bolla, R., Bruschi, R.: A high-end linux based open router for IP QoS networks: tuning and performance analysis with internal (profiling) and external measurement tools of the packet forwarding capabilities. In: Proc. of the 3rd International Workshop on Internet Performance, Simulation, Monitoring and Measurements (IPS MoMe 2005), Warsaw - Poland, Institute of Telecommunications, Warsaw University of Technology (2005)