Parallel arithmetic encryption for high-bandwidth communications on multicore/GPGPU platforms[*]

Ludovic Jacquin, Vincent Roca
INRIA Planète team, France
{ludovic.jacquin,vincent.roca}@inria.fr

Jean-Louis Roch [†], Mohamed Al Ali
Laboratoire d'Informatique de Grenoble (LIG), France
{jean-louis.roch,mohamed.alali}@imag.fr

June 23, 2010

Abstract

In this work we study the feasibility of high-bandwidth, secure communications on generic machines equipped with the latest CPUs and General-Purpose Graphical Processing Units (GPGPU). We first analyze the suitability of current Nehalem CPU architectures. We show in particular that high performance CPUs are not sufficient by themselves to reach our performance objectives, and that encryption is the main bottleneck. Therefore we also consider the use of GPGPU, and more particularly we measure the bandwidth of the AES ciphering on CUDA. These tests lead us to the conclusion that finding an appropriate solution is extremely difficult.

[*] This work is partially supported by French the French Ministère de l'Économie, de l'Industrie, et de l'Emploi and the Global competitive cluster Minalogic, project SHIVA no 09-2-93-0473.

[†] INRIA, Grenoble Université, Institut Nation Polytechnique de Grenoble (INPG).

1 Introduction

During the past few years, communications have experienced tremendous throughput increases since 10 Gb/s Network Interfaces Controllers (NIC) are now common on high performance machines. This situation authorizes the deployment of large scale clusters for distributed computing. In this context, security is more and more often a requirement and communications between sites must be encrypted to avoid critical information leaks. This is the case for instance with medical applications that require the processing power of computational grids but manipulate highly confidential data from patients for instance [17]. This is also the case for inter-site secure communications, where the aggregated traffic can reach high throughput during database synchronization or remote backup procedures for instance. The Moore law has long been sufficient to keep a reasonable equilibrium between the available processing power and the physical link throughput. However, this is no longer the case with transmission throughput in the order of 10 Gb/s when ciphering is required. Indeed, at 10 Gb/s, the available processing time to encrypt or decrypt data is in the order of a few CPU cycles only.

This work addresses the feasibility of achieving high-bandwidth secured communications using generic off-the-shelf components, such as multicore CPUs and General-Purpose Graphical Processing Units (GPGPU). Based on our detailed experimental analysis, we conclude that high-bandwidth secure networking does require high-speed arithmetic facilities.

This article is structured as follows. Section 2 gives an overview of the state of the art in the domain, both from the hardware and software points of view. Section 3 describes the benchmark architecture on which the chosen proof-of-concept tests are performed. In Section 4, we inspect the algorithms and protocols used to secure communications, as well as their parallelization. Then the results of our experimentations are presented, on multicore architectures in Section 5, and on GP-GPU in section 6. Finally, Section 7 presents the conclusions from this study and the perspectives.

2 Related works

To our knowledge, no reference that couples together high-bandwidth and parallel encryption exists nowadays. But there are some work that has been done in fields near our center of interest.

Specialized Hardware Solutions exist for encryption and high-bandwidth networking, but they are based on specialized hardware provided by companies such as Cisco [2] or Cavium [1]. The main disadvantages of these solutions are:

- scalability: for example a Cisco encryption board upper limit is 2,5 Gb/s, and four units are needed to reach 10 Gb/s.
- upgradability: once the hardware is installed, it is no more possible to change the chips.
- recyclability: in the future, when we will be deploying about 100+ Gb/s links, 10 Gb/s hardware will become obsolete and useless.
- price: such solutions cost 10 times more than an average server.

Therefore we chose not to consider this approach in our work.

High-bandwidth networking Previous works focused on routing and have explored the capabilities of generic servers to serve as high bandwidth routers. When the need for security is considered, the experiments carried out are rather limited. Yet their two main conclusions are that: (1) 10 Gb/s routing is possible, although it uses all the CPU resources [15], and (2) that when using IPsec and AES 128 bits encryption, they only achieve 1,5 to 4,5 Gb/s transmission speeds (depending on the packet size) [16].

Cipher parallelism On CPU, Roche et al [14] propose to use a block cipher in counter chaining mode, a mode that is well-suited for parallelization in addition to feature a high security level. The authors use a very effective method based on work-stealing and loop rescheduling for DES encryption. Using this method and when considering only in memory operations, they obtain a near-optimal speedup on multicore systems. However, when considering network I/O (our case), the speedup is significantly reduced, by up to 50%.

On GPUs, Andrea Di Biagio et al [13] present two implementations of the AES algorithm that are parallelized for GPUs. They propose a fine-grained approach for parallelizing the inner operations of the AES round, and a coarse-grained approach that focuses on the parallelism outside the round operations an data blocks. Thanks to these techniques, they obtain great

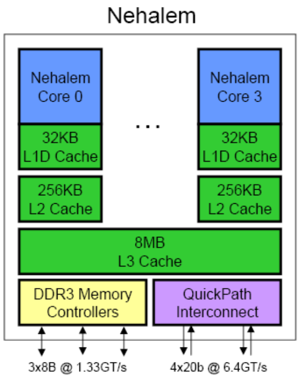

Figure 1: Nehalem cache architecture.

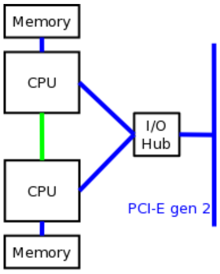

Figure 2: New Nehalem motherboard architecture.

speed-ups when ciphering large files. However this is not directly applicable to our case since we need to consider smaller sizes (we follow a per-packet approach).

3 Benchmark architecture

In this section, we describe the server we used for our experiments and its basic performance.

Processor architecture The Intel Nehalem architecture [4] introduces parallelism mechanisms at the hardware level. Those mechanisms concentrate on two aspects of the architecture: cache/memory accesses and I/O connections.

As far as memory is concerned, each CPU chip introduces a new level of

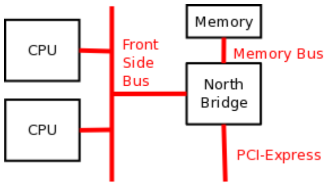

Figure 3: Old motherboard architecture.

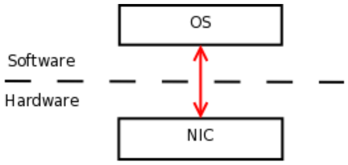

Figure 4: Old Network Interface Cards (NIC).

cache (L3) that is shared by the four cores, and two physical buses (RAM and QuickPath) (figure 1). To each CPU chip is associated a privileged memory bank (connected to the CPU chip RAM bus) that is accessed directly by the associated CPU, and that can still be accessed by remote CPU chips through a CPU-to-CPU interconnection (figure 2). This new memory hierarchy enhances parallelism since each CPU can now access its own memory bank independently, without creating any contention with respect to other CPUs. As far as I/O communications are concerned, Intel also removed the classic bottleneck by replacing the old shared-bus architecture (e.g. the Front Side Bus, FSB, of figure 3) by point-to-point connections between each CPU and a I/O hub (figure 2).

Network Interface Card (NIC) In traditional NIC architectures, the NIC driver is the only entity capable of accessing the NIC (figure 4). On the opposite, recent NIC also include parallelism mechanisms at the hardware level, and more precisely new NIC define multiple reception and transmission queues (figure 5). Thanks to this change, the traditional bottleneck at the hardware/software edge is removed. Even though it was primarily designed for virtualized server [5], the new NIC architecture is useful in our use-case because they can help spreading the network charge over different cores, over different OS threads.

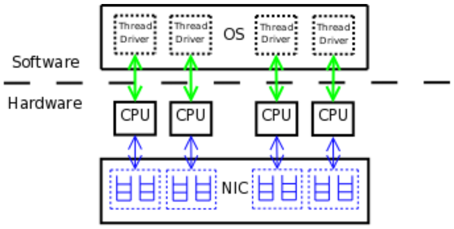

Figure 5: New Network Interface Cards (NIC), with multi-queue.

Table 1: First performance evaluation.

Without IPsec	9,2 Gb/s (best)
IPsec in tunnel mode (AES-CTR)	0,8 Gb/s
AES-CTR cipher (mono-thread)	0,8 Gb/s

First performance evaluation Using a Nehalem based server, with multi-queue NIC and running GNU[9]/Linux[7], we carried out several bandwidth tests, using a TCP/IPv4 bulk transfer, with/without IPsec, in order to have a first idea of the performance of an ”out of the box” solution. More precisely, we use a bi-Xeon 5530 server (for a total of 8 cores), equipped with a 10 Gb/s NIC based on the Intel 82598EB chipset. The average bandwidth at application level (i.e. without counting the Ethernet, IP and TCP header overheads) is summarized in table 1. The evaluation tool used was Iperf[6].

Without IPsec/encryption, we can saturate the physical link, with average transmission rates between 8 and 9.2 Gb/s depending on the server load. However, when IPsec (ESP / tunneling mode, see section 4, using AES in counter mode) is used encrypt communications, performances drop by more than a factor 10. One can notice that this is roughly the same throughput as the AES-CTR cipher of the libcrypto library [8] in mono-thread mode.

The conclusions is that using the latest hardware (Nehalem processors, a multi-queue NIC) and kernel-space IPSec support is not sufficient to achieve high throughput: an ”out of the box” GNU/Linux solution fails to achieve 10 Gb/s with a bulk encrypted traffic. More fundamentally, during the past 10 years, we observed a fundamental shift in CPU development, from frequency to parallelism, and this shift is also impacting motherboards and NIC. Therefore, the seek for high performance communications requires that developers take this situation into account and develop highly parallelized applications, which is the only possibility to take benefit of current and future servers,

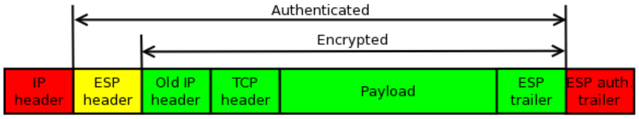

Figure 6: IPsec packet format.

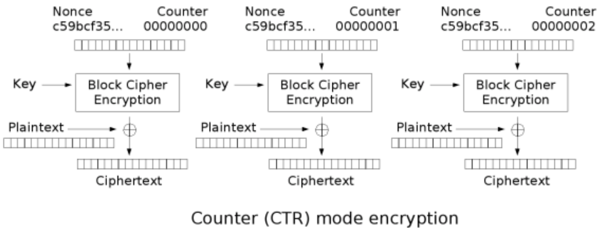

Figure 7: Cipher counter mode.

instead of counting on the raw performance growth of CPU cores.

4 Encrypted communications and parallelism

IPsec A common solution for network-level encrypted communications is IPsec [10]. IPsec is mostly used with ESP (Encapsulated Security Payload) in tunnel mode, and provides confidentiality, source authentication and integrity verification services. When this mode is used in a security gateway, packets coming from the local network are encapsulated in a new IP packet as shown in figure 6. The ESP header allows the remote host to identify and use to same cipher and key. One of the ciphers supported by IPsec is AES which is considered as one of the best symmetric key algorithm of the moment. Therefore we concentrate our effort on this cipher in counter mode (section 4). Finally, IKE (Internet Key Exchange) is another protocol related to IPsec which allows two hosts to safely exchange the session keys. However since IKE does not impact our tests, we do not consider it.

Counter mode From the five ciphering modes of operations (Electronic Code-Book, Cipher Block Chaining, Cipher-FeedBack, Output FeedBack and Counter), we choose to use the Counter mode since it allows an easy parallelism[3]. More precisely (figure 7), using a set $\{nonce, counter\_0, f\}$

(where f is a function that produces a sequence guaranteed not to repeat for a long time and that enables to easily obtain $counter\_i = f(counter\_0, i)$) and a secret key K (constant during the communication), we can generate a pseudo random bit-stream (or keystream). This keystream is then `XORed` to all the blocks M_i of the plaintext message M, during the encoding process, and the same keystream is `XORed` to all the blocks of the ciphertext message during the decoding process. Since the various blocks are encrypted/decrypted independently, the Counter mode allows an easy parallelization, by dispatching block processing over the computing units.

Parallelism and packet streams We are dealing with TCP flows, and therefore the TCP segments should be delivered in order to the receiving TCP engine. Indeed mis-ordered TCP segments are considered as the sign of a potential packet erasure over the network, which leads the receiver to generate immediate duplicated acknowledgements. Upon receiving three such duplicated acknowledgements, the TCP sender will immediately enter in congestion avoidance state and reduce the connection throughput accordingly. Therefore the parallel processing of incoming packets must preserve the ordering within a given TCP connection.

[12] introduces a work-stealing based algorithm to distribute tasks for a given stream over different processors in such a way that it guaranties the output ordering. The relative speedup is a factor 6 when using 8 processors, which is rather good compared to the factor 4 obtained with classic parallel algorithms.

5 Experimenting bandwidth limitations on multicore environments

Experimental settings We know (section 3) that even though the Nehalem architecture can handle 10Gb/s traffic without encryption, the standard IPsec implementation limits us to 0.8 Gb/s (which is also the monothread bandwidth of AES-CTR). We now want to assess our server *peak performance level* (see section 3 for server description), in the most ideal configuration, at the sender. To that purpose, one thread is dedicated to TCP/IP processing (no encryption) and several threads performs encryption over unrelated buffers (these buffers are not related to the TCP/IP packets, even if in a real

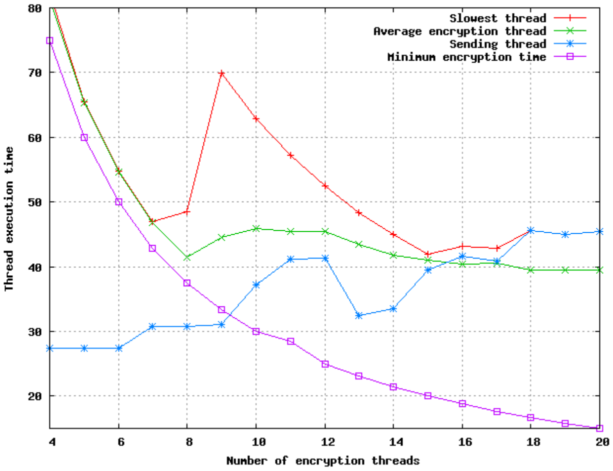

Figure 8: Communication and encryption thread execution times as a function of the number of encryption threads.

situation, they would be IPsec packets). More precisely, the sending application sends 240 Gbit of data with the communication thread, and distributes the encryption of the corresponding number of data packets over the set of encryption threads. We then evaluate the upper encryption performance as 240 Gbit divided by the processing time of the slowest thread. As can be seen, these tests fail to catch the real behavior of an IPsec protocol stack. However it is sufficient to provide an upper bound of the performance, and is useful to assess the feasibility of the problem: is a 10 Gb/s encryption feasible or not on this architecture?

Optimal number of threads First of all, we evaluate the number of threads that enable us to achieve the best performance. Figure 8 plots the measured time used by each thread as a function of the number of threads.

As one can predict, under 7 encryption threads, we are limited by the encryption thread processing time, which in turn is limited by the AES bandwidth. This test was performed on 240 Gbit of data, split in buffers of 3 kB. So for the 6 encryption threads (each having to encrypt 40 Gbit), one cannot expect better results than 50 seconds for each thread. We can see that threads are only a few seconds slower than the monothread bandwidth of the AES cipher itself. For the networking thread, we are under 30 seconds which corresponds to more than 8 Gb/s. In this part of the curve, there is a

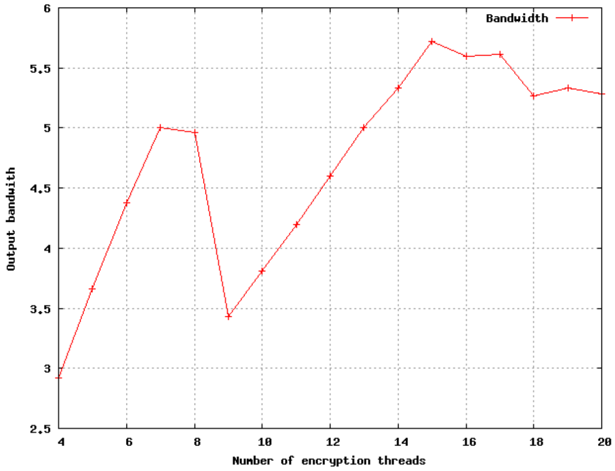

Figure 9: Bandwidth as a function of the number of encryption threads.

linear gain with the number of available threads.

Between 8 and 15 encryption threads, we are above the maximum number of threads that can truly run in parallel over the 8 cores. However, the CPU hyperthreading technique (HT) enables the parallel execution of these threads. By looking more carefully at the results, at first we see that the bandwidth decreases (figure 9) and HT seems to be a drawback. Then, between 13 and 15 encryption threads, the HT mechanism improves the achievable throughput.

To better understand the HT behaviour, we measure the thread execution time distribution for the 2 extremes (values are averaged over 10 tests): with 9 and 15 encryption threads (figure 10), to which we need to add the communication thread. More precisely, we performed 10 tests for each configuration, and we calculate the execution thread distribution. We see that the distribution is uniform with 15 encryption threads. On the opposite, with 9 encryption threads the distribution exhibits an important tail: we have six threads around 36 seconds, one thread between 45 and 50 seconds and two threads near 70 seconds. What happens here is that the first 8 threads use the 8 ”real” cores (one is in fact slower because of the impacts of other system processes), and the remaining threads are scheduled using the HT capability of the processors. This experiment shows that two threads are delayed and do not get access to a fair share of the available CPUs, which significantly impacts the estimated throughput that only takes into account

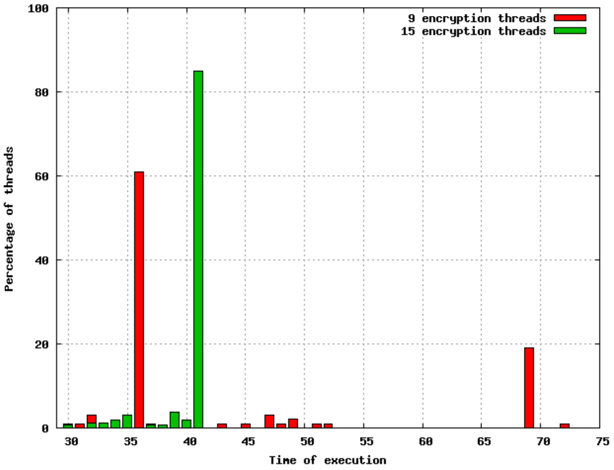

Figure 10: Communication and encryption thread execution time distribution.

the slowest thread (bottleneck). We do not experience such a phenomenon with 15 encryption threads, which shows that scheduling is done in a fair way.

Finally, with 16 or more encryption threads we do not experience any benefit in increasing the number of threads. We can conclude that optimum performance is achieved with 15 encryption threads, plus one communication thread, and that the maximum achievable throughput is around 5.75 Gb/s.

Data size impact Now that we identified the appropriate number of threads (15 for encryption, 1 for communications), we consider the influences of the data buffer size. Figure 11 shows that the execution time of all threads (encryption and communications) is a linear function of the data size. We also noticed that encryption is the bottleneck compared to communication.

Extrapolation to the receiver In a high-bandwidth context, a receiver will most likely use the polling mechanism instead of interruptions that generate high CPU consumption. This is possible because of the high-bandwidth context which implies the reception of millions of packets per second, which will be handled by a dedicated thread. Therefore the situation is rather similar to the sender side where a thread is dedicated to the sending operations. We can anticipate that the same bottleneck remains the same at the receiver

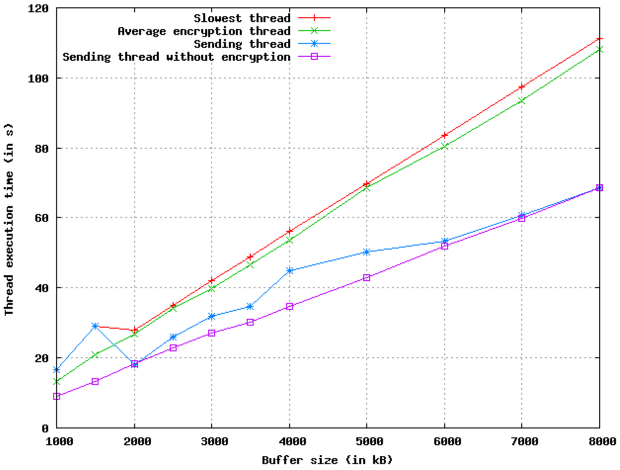

Figure 11: Sending time as a function of the data size.

side (this claim remains to be confirmed).

Partial conclusions Previous experiments have shown that even an up-to-date 8-core server (two Nehalem CPUs) cannot handle secure communications at the full 10 Gb/s Ethernet speed, and that encryption is the bottleneck. So we need to search another way of doing encryption without overloading the CPUs.

6 Cyphering capabilities of GPUs

GPU and CUDA The domain of Graphical Processing Units (GPU) recently made major progress, much faster than CPUs, and as a consequence, some GPUs now have more transistors than new quad-core CPUs. This is due to the fact that they are specifically designed for intensive and highly parallel computing, as is needed for image rendering. Since these GPUs have a highly parallel structure, many complex algorithms have been redesigned to take advantage of it. This approach also offloads computing intensive tasks on to the GPU, which saves a lot of processing time in the general purpose CPU.

The two major GPU manufacturers, NVidia and AMD, have both released development systems for GPU hardware, respectively CUDA and

Stream. The Nvidia CUDA framework extends the C language to give access to the GPU, allowing C functions to run on the GPU stream processors in parallel. With CUDA, a programmer can use both regular C code and GPU code in the same file which simplifies the development process. CUDA abstracts the parallelism and gives the notion of ”blocks” and ”threads”: several ”threads” run within each ”block”. Each ”block” is independent from other ”blocks”, and if ”threads” from different ”blocks” need to communicate they need to use the global memory of the GPU. Communication within a ”block” is done using the block’s memory. Finally, data needs to be moved from the host memory to the GPU memory, which is one of the main limitation of the approach because of the associated latency.

AES parallelization The AES algorithm is a standard algorithm, widely used in communication systems. With the CTR mode of operation, AES can be easily implemented in a parallel manner (section 4), and different parts of the cleartext message are processed on different GPU processing units. However, since each round in the algorithm depends on the previous round result, we cannot go any further and introduce a finer grain parallelism.

Experimental settings During the experiments, we use an NVIDIA GeForce GTX 295 GPU. We consider buffers of size in the range 1KB to 128KB, since our focus is on network packets whose size depends on the target physical layer, but is usually small. We then compare the results with those achieved on a CPU (an Intel bi-Xeon 5530 running at 2,4 GHz in this case).

Performance Evaluation We did several modifications on the implementation of [11] in order to get better results when encrypting files of small sizes. Indeed, using small files prevents to use the GPU in an optimal way for two reasons: first of all, the GPU cores are not all used in this case, and secondly, a lot of overhead is introduced when we wait for previous files to be processed. In order to solve this problem, we propose to use threads or what is known as ”CUDA streams”. In our experiments we define 1024 streams to cipher files of the same size. The idea is that operations are done asynchronously, so while computations are done on previous files, new files are transfered to the GPU. In addition, since the files are relatively small, the ciphering of each file is performed on one CUDA block in order to reduce the overhead of distributing the file across many blocks. These techniques allow to use all

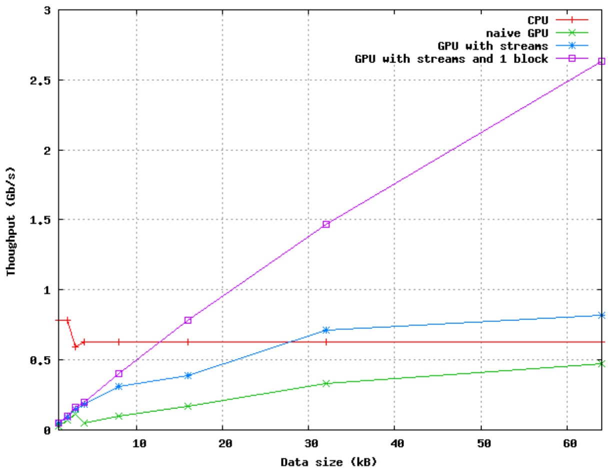

Figure 12: Cipher Throughput.

the stream processors of the GPU efficiently.

The results (figure 12) show that the CPU speed is almost constant, regardless of the file size. Concerning GPU, we compare three strategies: the first one is the naive implementation that was used for large file sizes; the second implementation uses CUDA streams to cipher asynchronously; and the last implementation limits the third implementation to use one block per file in order to reduce the overhead. The speed-up obtained from the third implementation over the naive one is 4 times for files of size 4KB. This increases up to 8 times in the case of files of size 128KB, reaching a throughput of 3,7 Gb/s. It should be noted that there is almost no significant speed-up in the case of files smaller than 4KB. Nonetheless, the throughput of very small files is still relatively low. To overcome this in terms of network packets, we propose the use of multiple GPUs (our server authorizes the use of up to four GPU cards) and the grouping of packets together to have files of higher size, almost 64KB or even 128KB. This may allow the reach of a throughput higher than 8Gb/s.

7 Conclusions and perspectives

This paper analyzes the raw capabilities of today's generic server for high performance secure communications. We show that relying on multi-core processors only is not sufficient to encrypt and send the data over the network at 10 Gb/s speed. We also show that the hyper-threading feature of processors should be used very carefully: correctly used, hyper-threading can help increase the sustainable bandwidth by distributing the processing load uniformly over the various cores, but certain configurations can also seriously decrease performances.

Since our experiments highlight the need for additional techniques, we also consider GPGPU for encryption operations. Although previous works in the domain have shown great results, it appears that for small data (i.e. one network packet) a GPGPU is slower than a CPU. However, we managed to optimize the AES implementation for the CUDA API and to achieve higher ciphering throughput than was previously feasible. We are continuing this effort in the hope to have similar results for data sizes in the order of a packet size.

Future research efforts will also address the use of multiple GPUs on a given server, as another possible way of improving performances. We are also considering the possibility to group several packets and to encrypt them together, as another way of improving the use of GPGPUs. However the practical impacts of doing so, both from a protocol point of view and interdependence point of view have to be seriously considered.

References

[1] Cavium networks. http://www.caviumnetworks.com/.

[2] Cisco systems. http://www.cisco.com/.

[3] Counter mode. http://http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation#Counter_.28CTR.29.

[4] Intel nehalem architecture. http://www.intel.com/technology/architecture-silicon/next-gen/.

[5] Intel virtual machine device queue technology. http://www.intel.com/network/connectivity/vtc_vmdq.htm.

[6] Iperf. http://perf.sourceforge.net/.

[7] Linux. http://kernel.org/.

[8] Openssl's libcrypto manual page. `http://www.openssl.org/docs/crypto/crypto.html`.

[9] GNU. `http://www.gnu.org/`.

[10] IPsec. `http://tools.ietf.org/html/rfc4301`.

[11] C. Berk Guder. AES on CUDA. `http://github.com/cbguder/aes-on-cuda`, January 2009.

[12] J. Bernard, J.-L. Roch, and D. Traore. Processor-oblivious parallel stream computations. In *16th Euromicro International Conference on Parallel, Distributed and network-based Processing*, Toulouse, France, Feb 2007.

[13] A. D. Biagio, A. Barenghi, G. Agosta, and G. Pelosi. Design of a parallel AES for graphics hardware using the CUDA framework. *Parallel and Distributed Processing Symposium, International*, 0:1–8, 2009.

[14] V. Danjean, R. Gillard, S. Guelton, J.-L. Roch, and T. Roche. Adaptive loops with kaapi on multicore and grid: Applications in symmetric cryptography. In A. publishing, editor, *Parallel Symbolic Computation'07 (PASCO'07)*, London, Ontario, Canada.

[15] M. Dobrescu, N. Egi, K. Argyraki, B. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting Parallelism to Scale Software Routers. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP)*, Big Sky, MT, USA, 2009. ACM.

[16] N. Egi, A. Greenhalgh, M. Handley, G. Iannaccone, M. Manesh, L. Mathy, and S. Ratnasamy. Improved forwarding architecture and resource management for multi-core software routers. In *Network and Parallel Computing, 2009. NPC '09. Sixth IFIP International Conference on*, pages 117–124, Oct. 2009.

[17] P. Vicat-Blanc/Primet, V. Roca, J. Montagnat, J.-P. Gelas, O. Mornard, L. Giraud, G. Koslovski, and T. T. Huu. A scalable security model for enabling dynamic virtual private execution infrastructures on the internet. In *9th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'09), Shanghai, China*, May 2009.