View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by INRIA a CCSD electronic archive server

Server-side performance evaluation of NDN

Xavier Marchal, Thibault Cholez, Olivier Festor

To cite this version:

Xavier Marchal, Thibault Cholez, Olivier Festor. Server-side performance evaluation of NDN. 3rd ACM Conference on Information-Centric Networking (ACM-ICN'16), ACM SIGCOMM, Sep 2016, Kyoto, Japan. pp.148 - 153, 10.1145/2984356.2984364 . hal-01386777

HAL Id: hal-01386777

https://hal.inria.fr/hal-01386777

Submitted on 24 Oct 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Server-side performance evaluation of NDN

Xavier MARCHAL, Thibault CHOLEZ, Olivier FESTOR
LORIA, UMR 7503 (University of Lorraine, CNRS, INRIA)
Vandoeuvre-les-Nancy, F-54506, France
{xavier.marchal, thibault.cholez, olivier.festor}@loria.fr

ABSTRACT

NDN is a promising protocol that can help to reduce congestion at Internet scale by putting content at the center of communications instead of hosts, and by providing each node with a caching capability. NDN can also natively authenticate transmitted content with a mechanism similar to website certificates that allows clients to assess the original provider. But this security feature comes at a high cost, as it relies heavily on asymmetric cryptography which affects server performance when NDN Data are generated. This is particularly critical for many services dealing with real-time data (VOIP, live streaming, etc.), but current tools are not adapted for a realistic server-side performance evaluation of NDN traffic generation when digital signature is used. We propose a new tool, NDNperf, to perform this evaluation and show that creating NDN packets is a major bottleneck of application performances. On our testbed, 14 server cores only generate ∼400 Mbps of new NDN Data with default packet settings. We propose and evaluate practical solutions to improve the performance of server-side NDN Data generation leading to significant gains.

1. INTRODUCTION

Nowadays the majority of the Internet traffic is about delivering content like video streaming which is the first source of data consumption with around 64% of the Internet traffic in 2014 [2]. This ratio increases every year with an estimation around 80% of the Internet traffic in 2019. But current Internet protocols are not optimized for such usage. For example, when many people ask for the same video content over a TCP connection, the distant server responds as many times as the number of demands. In the case of live video streaming, when a user sends data at a rate of 3Mbps and has 1000 viewers, the servers send 1000 times the data rate (3Gbps).

The Named Data Networking (NDN[1]) protocol [4][8] was mainly designed with the intention of reducing network congestion by gathering Interest for a same content at network level and forwarding only the first one, so that the server only needs to respond once for all users. But NDN also provides in-network caching to store data close to users and limit the distance travelled by popular content. In NDN, Data packets must have at least a SHA-256 hash that should be digitally signed to link the data and its name to the provider. Thus, this hash/signature must be verified in order to check the packet integrity, and, if signed, to authenticate the content provider. However, by using signatures, NDN exchanges can become CPU intensive when Data packets are generated. This is a critical constraint for real-time applications (live-video, online-games, VoIP, etc.) that generate fresh Data and cannot be signed in advance.

Most research papers are focused on NDN caching performance evaluation but none considers the performance of NDN Data packets generation while this is also of prime importance for the aforementioned applications. In this paper, we give the first comprehensive evaluation of NDN throughput at the server side while measuring the CPU consumption under different scenarios thanks to NDNperf, an open source tool for NDN performance evaluation we made.

The rest of the paper is organized as follows. Section II presents the related work on NDN performance evaluation and the available tools. Section III introduces NDNperf and the testbed we use in Section IV to conduct our experiments on server-side performance evaluation of NDN. Section V presents and evaluates some possible improvements to reduce the signature overhead. Finally, Section VI concludes the paper.

2. RELATED WORK

Although real performance evaluation was not at the center of current research efforts of the NDN protocol due to its youth, some studies show performance tests to highlight the benefits of their own solutions.

Guimarães *et al.* [3] extend the experimentation done by Van Jacobson [4] in a virtual network over the Internet with their testbed named FITS. They highlight the poor performance of CCN in point-to-point transfers compared to TCP despite the latter being limited by a small link capacity (10 Mbps). Oueslati *et al.* [5] and Carofiglio *et al.* [1], members of the CONNECT project, work on control flow at two levels: at the receiver level with an implementation of the AIMD algorithm and at the router level with fair sharing between

[1]NDN project: `http://named-data.net/`

flows for single and multi-path to avoid congestion and give an efficient bandwidth distribution among users.

Some studies considered both software and hardware performance evaluation of NDN packet forwarding. Yuan *et al.* [7] performed a study of the CCN forwarding daemon in a multi client/server environment with throughput monitoring and profiling. They expose the operational flow of the forwarder and highlight issues regarding the software scalability, like too complex operational flows, and propose some ideas that will help to improve it. Won So *et al.* [6] work on an implementation of an NDN forwarder on Cisco routers with integrated service modules that can take advantage of multi-core processors. They report that their implementation can theoretically achieve high throughput (20 Gb/s) based on the number of packets forwarded.

To our knowledge, no study investigated the impact of NDN Data packet generation on server performances, what we conduct in the following sections.

3. EXPERIMENTAL ENVIRONMENT

We designed NDNperf, an open source tool[2] for NDN server-side performance evaluation and sizing purposes, in order to have an idea of the throughput a server can achieve when it has to generate and transmit NDN Data packets. It is very similar to iPerf and also needs a client and a server to perform the measurements while minimizing the number of instructions between Interest reception and Data emission. It exists in two flavors (Java and C++) and has the following features:

- Periodic report of performances: end-to-end throughput, latency, processing time;
- Fresh NDN Data generation or NDN Data delivery from caches;
- Multi-threaded (one main thread for event lookup and N threads for NDN Data generation);
- Able to use all available signatures implemented in the NDN library, choose the size of the key, and the transmission size of Data packets;
- Message broker implementation (Java version only).

NDNperf features many options regarding the signing process because we identified it as the main bottleneck of application performances. Indeed, code profiling using Valgrind on a running NDN server showed that most of the processing time is dedicated to signing (between 87% and 64% respectively for a Data packet with a payload size of 1024 and 8192 octets) which constitutes the main driver of performance improvement. The second most costly operation is the wire encoding, accounting for 6% for a payload size of 1024 octets and 30% for 8192 octets.

[2] `http://madynes.loria.fr/software/ndnperf_cpp.zip`

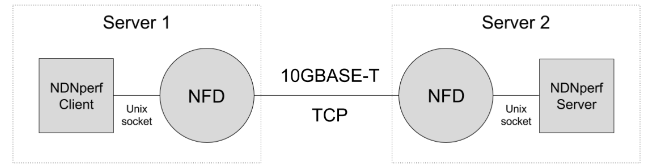

Figure 1: Testbed implementation

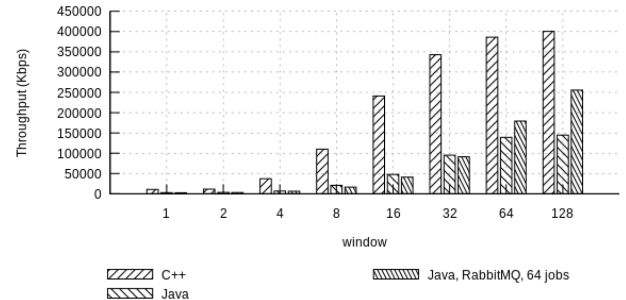

Figure 2: Throughput of new NDN Data for different NDNperf implementations

Our evaluation testbed is composed of two DELL PowerEdge R730 servers. Each server features two Intel 2.4 GHz octo-core Xeon processors (E5-2630 v3) with Hyper-Threading and Turbo enabled, 64GB of RAM and two 400GB SAS SSD in RAID0 for the operating system (Ubuntu 15.04 server). The servers are directly interconnected through Intel 10Gbps Ethernet network interfaces (Intel X540). NDNperf uses the version 0.4.0 of the NDN libraries and each server runs is own NDN Forwarding Daemon (NFD) instance (Figure 1).

In Figure 2 is given an overview of the achievable throughput for RSA signature (SignatureSha256WithRsa) by the three different implementations of NDNperf (C++, Java, Java with message broker) with different window sizes and a payload of 8192 octets. Due to a higher IO latency and the fact that signing packets seems more CPU intensive in Java, this implementation is much slower. Our message broker implementation with RabbitMQ performs better when a large window is used (+75% compared to Java) but at the cost of another dedicated server helping for the signing process. Considering the better performances of the C++ version, it will be used in the next experiments.

Before each test, we send a warm-up traffic to fill the data structures of the two NFD instances. In the upcoming experiments, we use a MTU of 1500 and an Interest window size of 8 packets at the client side. Indeed, Figure 3 shows the average throughput for different window sizes, signing configurations (*DigestSha256* or *SignatureSha256WithRsa*) and packet sizes for a single-thread content-provider. According to these results, the best window for our testbed is of 8 packets, with no real gain past this value. We noticed similar results for other encryption algorithms and for cached Data. When us-

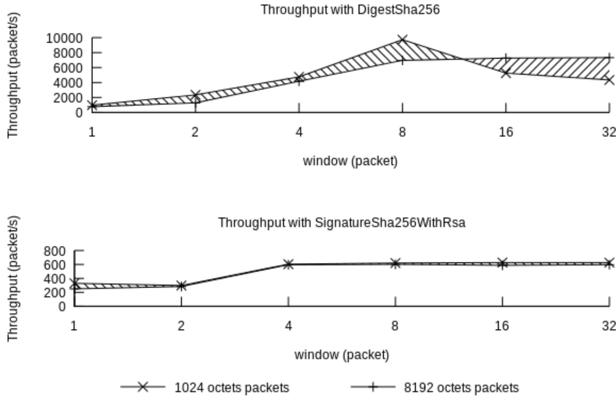

Figure 3: Throughput of new NDN Data for different window sizes

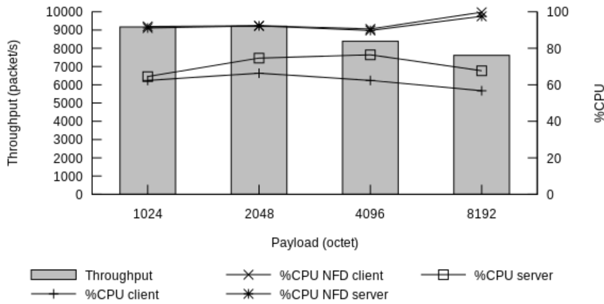

Figure 4: Throughput of new NDN Data with *DigestSha256* and the associated CPU usages

ing the multi-threaded content-provider, we will multiply this 8-packets window size by the number of threads to take advantage from the available CPU cores.

4. EVALUATION

4.1 Evaluation of a single-threaded content provider

Our first evaluation is based on a single-threaded version of NDNperf. The achievable throughput is tested in these four conditions: New Data generation with (1) *DigestSha256* or (2) *SignatureSha256WithRsa*, and Content present in (3) client-side NFD cache or (4) server-side NFD cache.

In the following experiment, we use a freshness value of 0 ms for NDN Data so that the client can never get Data packets from any cache. Figure 4 displays the throughput in packets per second and the percentage of CPU usage for each of the 4 running applications (NDNperf client, client-side NFD, server-side NFD and NDNperf server), and this, for different payload sizes. Packet throughput decreases with the payload size (about -10% for 4096 and 8192 octets payload size) but this is vastly counterbalanced by the fact that the payload size doubles each time, so the global data throughput still increases with a maximum average throughput of 487Mbps. In the case of a simple SHA-256 hash, a mono-threaded application is enough to saturate the NDN Forwarding Daemon (NFD) which constitutes the bottleneck of this experiment.

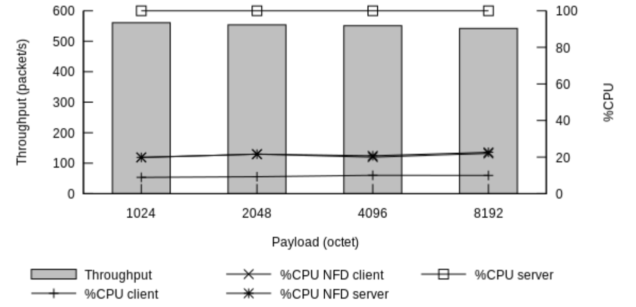

Figure 5: Throughput of new Data with *Sha256WithRsa* and the associated CPU usages

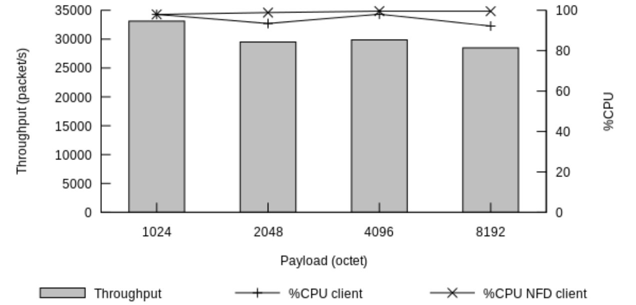

Figure 6: Throughput of client-side NFD cache with the associated CPU usages

In the next experiment (Figure 5), a RSA digital signature is used. Signing packets can be very CPU intensive according to the RSA key length. For this test is applied the default key size used by NFD which is a RSA 2048 bits key. As shown in Figure 5, the limiting factor is now by far the server application that fully uses its allocated processor core in all configurations. The RSA signature costs too much to be handled by only one thread and the result is that only about 21% of the NFD forwarding capability is used. Throughput is limited around 550 packets per second which represents up to 34Mbps of new NDN Data with the largest payload size (8192 octets).

4.2 Evaluation of NFD

The next two Figures (6 and 7) represent the NFD cache performance throughput. For these tests we increased the Content Store capacity of the NDN forwarding daemon from 2^{16} to 2^{21} packets, to have enough NDN Data packets to hold at least one minute per test. For each payload size we generate Data packets with *SignatureSha256WithRsa* before doing our tests, and these packets will constitute the cache.

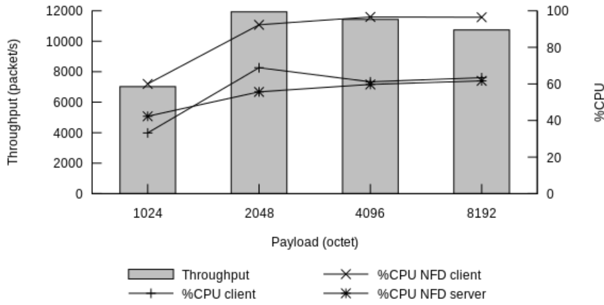

Figure 7: Throughput of server-side NFD cache with the associated CPU usages

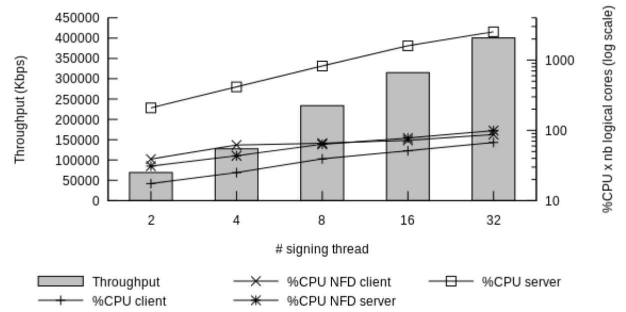

Figure 8: Throughput of new NDN Data with *Sha256WithRsa* with the associated CPU usages (multi-thread server)

Figure 6 shows the throughput and CPU usage for the client application and the client-side NFD instance from which the NDN Data are retrieved. In comparison with the previous tests, the achieved throughput is really far above and more than three times greater than the throughput of the server sending unsigned Data (*DigestSha256*) in Figure 4. With a payload of 8192 octets, the throughput is about 1792Mbps. This difference can be explained because the client-side NFD only needs to look in its Content Store to retrieve the Data packets and doesn't have to access the PIT nor the FIB. This also demonstrates that NDN performs well when cached data are transmitted.

Then, in Figure 7, we retrieve the NDN Data packets from the server-side NFD cache: the client-side NFD doesn't have any of the requested Data packet and forwards the Interests to the serve-side NFD instance which directly answers from its Content Store filled with the needed packets. With a maximum of 671Mbps, the throughput is disappointing but is still above the throughput achieved with newly generated NDN Data. Like with *DigestSha256*, we can observe a slight decrease in packet throughput with the payload size ($\sim$ -5%), except for a payload of 1024 octets for which the result is surprisingly low. For larger payload sizes, the limiting factor is clearly the client-side NFD whereas the server-side NFD only uses around 60% of its CPU core. From this, we conclude that the PIT and FIB lookup process on the client-side NFD have a very high price.

4.3 Evaluation of a multi-threaded content provider

For the next experiment, we use NDNperf as a multi-threaded application and compare the throughput increase for authenticated packets compared to the single-thread setup of Figure 5. Figure 8 shows the throughput regarding the number of threads used on the server. The experiment was done with the default RSA 2048 bits key and a payload of 8192 octets while we run up to 32 signing threads concurrently to match the number of logical cores available on our server. In these conditions, we can saturate an instance of NFD with approximately 25 logical cores generating new NDN Data. But using nearly all the processing resources to prepare NDN Data packets seems inefficient.

In conclusion, NFD is the bottleneck of our testbed configuration when NDN Data are present in caches or generated without RSA signature. When digital signature is used, the content-provider clearly becomes the bottleneck unless it is multi-threaded and can allocate much computation power for signing purpose. This can be a major burden for applications using real-time authenticated NDN Data.

5. REDUCING SERVER SIGNING OVERHEAD

5.1 Changing the signing configuration

The NDN library allows us to use two asymmetric algorithms. Actually we can use RSA and ECDSA, but there are also some hints for future algorithms like HMAC and AES. With the C++ version of the NDN libraries, we can only generate two different key sizes for RSA (1024, 2048) and ECDSA (256, 384). Figure 9 reports the throughput for new NDN Data with these different algorithms and key lengths combinations with the associated CPU usage. This experiment is done without Intel HyperThreading, so the CPU usage corresponds to real processor cores and not logical ones.

For RSA, reducing the key size can drastically reduce the CPU usage of the application with only 6 threads needed instead of 14 to saturate NFD. Of course, decreasing the key size also means decreasing the security level, but this may be an acceptable trade-off for some applications. Then for ECDSA, the throughput is nearly the same than RSA but it provides a higher security level. Indeed, according to the literature, ECDSA-256 provides an equivalent security level than RSA-3072. Moreover, ECDSA is more efficient regarding the computation as ECDSA-256 only needs about 8 threads to saturate NFD instead of 14 for RSA-2048. According to these results, we can currently recommend ECDSA-256 to authenticate NDN Data packets: it provides a

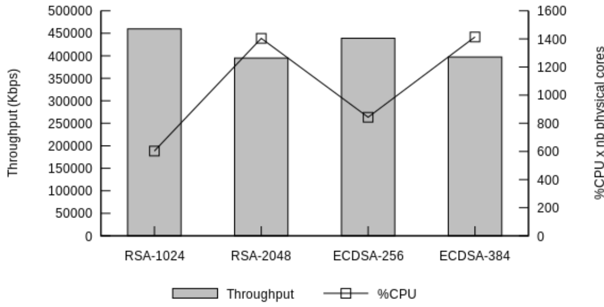

Figure 9: Throughput of new Data with available signatures and associated CPU core usage

Table 1: Throughput and server load under different scenarios

Source of NDN Data	Throughput	Nb of server cores
Client-side cache	1792 Mbps	0
Server-side cache	671 Mbps	0
Server with SHA256	487 Mbps	1
Server with RSA-1024	460 Mbps	6
Server with RSA-2048	394 Mbps	14
Server with ECDSA-256	439 Mbps	8
Server with ECDSA-384	397 Mbps	14

similar throughput and a better security level for a lower CPU cost than RSA-2048 which is still the NDN default signature algorithm. Overall, the best results achieved on our tested for a payload of 8192 octets are synthesized in Table 1.

5.2 Improving the NDN signing function

Each time we ask for a packet signature, the NDN library reads the file containing the private key, what is not efficient. Moreover, the public and private keys' file names are a hash of the NDN name of the key-pair, and the library needs to compute each time twice the hash (one for each key) in order to compute the signature. All these operations slow down the signature generation and could be easily avoided. The idea is to allow the content-provider to store the key he wants to use and the associated information like the *SignatureType* in a dedicated data structure passed as a parameter to the signing function. Figure 10 shows that the delay to sign is vastly reduced (in average 250 microseconds saved per call) with our custom function compared to the one provided by the NDN library for both RSA and ECDSA signatures. This optimization has a positive effect on the throughput when the server is the bottleneck as illustrated in Figure 11. This test was done with 4 signing threads, a payload of 8192 octets and

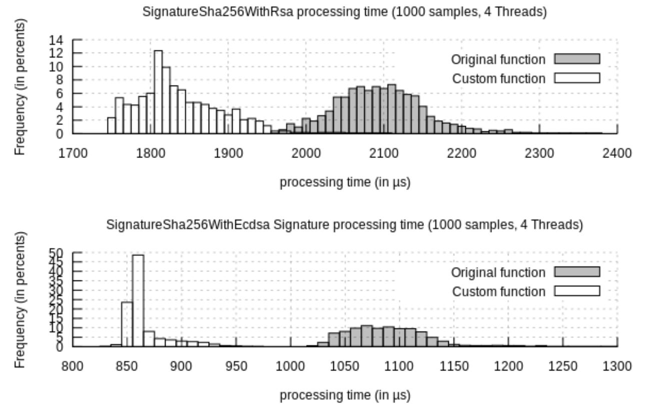

Figure 10: Processing time repartition for RSA and ECDSA with the two signing functions

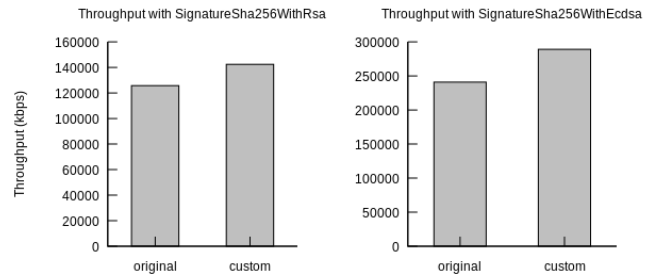

Figure 11: Throughput comparison of the two signing functions with RSA and ECDSA

a RSA-2048 or 256-ECDSA key. In these conditions, our signing function increases by 13% the throughput of RSA signature and by 20% for ECDSA.

5.3 Increasing the size of NDN Data packets

Another way to reduce the overhead is to reduce the number of signatures needed to transmit a resource. A simple way to do that is to increase the amount that a Data packet can carry, but NDN packets are currently limited to 8800 octets. If we analyze this limit regarding the measured size of web contents[3], the current size can only cover around 60% of the web resources population in a single packet. Moreover, larger elements like videos would need fewer packets to be sent and so fewer signatures. As shown in Figure 12, increasing the maximum size of NDN Data packet has a positive effect on the throughput. Even if the number of processed packets decreases with the size, the total throughput still increases with an average of 917Mbps with a payload of 32786 octets. The capacity of applications to benefit from this last improvement may depend of their traffic pattern and delay tolerance.

[3]Data from httpparchive.org, march 15 2016

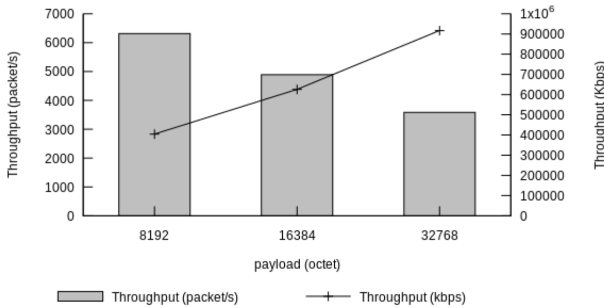

Figure 12: Throughput of new NDN Data with *Sha256WithRsa* for different payload sizes

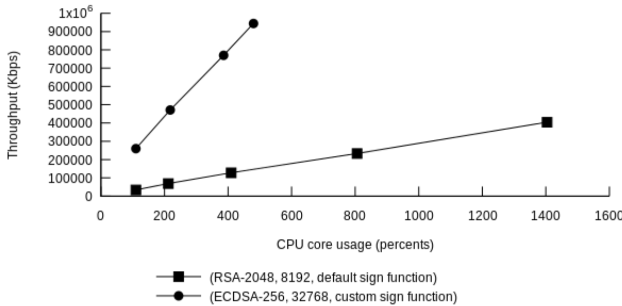

Figure 13: Throughput of new NDN Data according to the CPU capacity for different signing configurations

Finally, Figure 13 compares the efficiency of NDN default signature configuration with the three practical improvements used all together. For a given CPU capacity, the optimized configuration vastly outperforms the throughput achieved by default. For example, with 4 CPU cores, the default NDN signature limits the server throughput to 125Mb/s while it could generate 800Mb/s with the optimized configuration.

6. CONCLUSION

We conducted in this paper a rigorous server-side performance evaluation of NDN. We developed NDNperf, a tool measuring the maximum throughput achievable by a real NDN application. All the tests confirm that no more than one client thread is needed to saturate NFD. On the server side, our findings are different. If we exclude the specific case when no authentication is needed (simple *DigestSha256*), applying signatures using asymmetric cryptography when preparing NDN Data packets needs an important CPU processing power and a multi-threaded content-provider to achieve a decent throughput. Even so, with the current NDN default signature, a dedicated server is not even sufficient to saturate NFD and becomes the bottleneck. But our tests also highlight that ECDSA-256 is a more efficient way to sign NDN packets. We evaluated other possible optimizations like an optimized signing function and a larger packet size and showed that they highly reduce the server load. In our future work, we will leverage more advanced technologies like GPU computation to speed up NDN packet processing and signing.

Acknowledgement
This work is partially funded by the French National Research Agency (ANR), DOCTOR project, under grant <ANR-14-CE28-0001>.

7. REFERENCES

[1] G. Carofiglio, M. Gallo, and L. Muscariello. ICP: Design and evaluation of an Interest control protocol for content-centric networking. In *IEEE INFOCOM Workshops*, pages 304–309, 2012.

[2] Cisco. Visual Networking Index: Forecast and Methodology, 2014-2019 White Paper, 2015.

[3] P. H. V. Guimaraes, L. H. G. Ferraz, J. V. Torres, D. M. Mattos, P. Murillo, F. Andres, L. Andreoni, E. Martin, I. D. Alvarenga, C. S. Rodrigues, et al. Experimenting Content-Centric Networks in the future internet testbed environment. In *IEEE International Conference on Communications Workshops (ICC)*, pages 1383–1387. IEEE, 2013.

[4] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12. ACM, 2009.

[5] S. Oueslati, J. Roberts, and N. Sbihi. Flow-aware traffic control for a Content-Centric Network. In *Proceedings of IEEE INFOCOM 2012*, pages 2417–2425, 2012.

[6] W. So, A. Narayanan, D. Oran, and M. Stapp. Named Data Networking on a router: forwarding at 20gbps and beyond. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 495–496. ACM, 2013.

[7] H. Yuan, T. Song, and P. Crowley. Scalable NDN forwarding: Concepts, issues and principles. In *Computer Communications and Networks (ICCCN), 21st International Conference on*, pages 1–9. IEEE, 2012.

[8] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang, et al. Named Data Networking. *ACM SIGCOMM Computer Communication Review*, 44(3):66–73, 2014.