Cost-based placement of virtualized Deep Packet Inspection functions in SDN

Mathieu Bouet, Jérémie Leguay and Vania Conan
Thales Communications & Security
4 rue des Louvresses, 92230 Gennevilliers, France
{mathieu.bouet, jeremie.leguay, vania.conan}@thalesgroup.com

***Abstract*—In today's IT systems, cyber security requires fine-grained, flexible, adaptable and cost optimized monitoring mechanisms. The emergence of new networking technologies, like Network Function Virtualization (NFV) and Software Defined Networking (SDN), opens up new venues for large scale adoption of these cyber security tools. In particular, Deep Packet Inspection (DPI) engines can be virtualized and dynamically deployed as pieces of software on commodity hardware. Deploying such software DPI engines is costly in terms of license fees and power consumption. Designing cost effective DPI engine deployment strategies that meet the cybersecurity operational constraints is thus mandatory for the adoption of this approach. For this purpose, we propose a method, based on genetic algorithms, that optimizes the cost of DPI engine deployment, minimizing their number, the global network load and the number of unanalyzed flows. We conduct several experiments with different types of traffic and different cost structures. The results show that the method is able to reach a trade-off between the number of DPI engines and network load. Furthermore, the global cost can be reduced up to 58% when relaxing the constraint on the used link capacity, that is the provisioning rate.**

I. Introduction

In present IT systems breaches take months to be discovered and days to weeks to be contained. Situational awareness is thus necessary for an effective cyber defense. In today's world, organizations must assume that their networks and systems will be compromised. Among all the functionalities required to accelerate breach detection and mitigation, fine-grained realtime monitoring of flows and users activity is inevitable [1]. The emergence of new IT architectures, such as cloud computing, stresses the need for more flexible, adaptable and cost-optimized cybersecurity tools. This paper studies a cost efficient Deep Packet Inspection (DPI) service that can be dynamically deployed as a piece of software on commodity hardware.

Network Functions Virtualization (NFV) is a network technology trend that, adopting the virtualization principles of cloud computing, pushes for the softwarization of network. This approach, supported by services providers [2], consists in delivering network functions as software that can run as virtualized instances and that can be deployed at required locations in the network, without the need to install specific equipment for each new service. It is applicable for any network function, such as Deep Packet Inspection, firewalling, caching, ciphers, load balancers, in both mobile and fixed networks. Virtualizing network functions enables to rapidly scale up (or down) services, that currently necessitate multiple dedicated hardware appliances, as it only requires the installation of virtual appliances on existing server equipment. Furthermore, NFV completes the Software-Defined Networking (SDN) technology where network becomes programmable and run on commodity hardware. Virtual appliances might be configured using SDN capabilities to automate the deployment and the configuration of the network with fine-grained flow policies.

DPI consists in filtering network packets to examine the data part (and possibly also the header) of a packet flow, searching for protocol non-compliance, viruses, spam, intrusions, or any defined criteria. DPI enables to decide whether a packet may pass or not. In case suspicious behaviors or attack mitigation, the decision could be made to route the packet to a different destination or to report it to a security tool. This network function, as many other, is more and more virtualized, that is, embedded in software libraries that can be deployed and used on demand on multicore commodity hardware.

In this paper, we propose a method that enables to optimize the deployment of such DPI engines, especially in SDN environments where flows can be manipulated atomically. It minimizes the number of deployed DPI engines, the induced network load and the number of non-analyzed flows taking into consideration operational constraints such as the maximum used bandwidth per link for provisioning policy and costs of engines, used bandwidth and SLA violations. Reducing the number of deployed DPI engines induces redirecting more flows towards them, thus increasing both global network load and link utilization. The method we propose is based on genetic algorithms, which have been shown to have good properties for this type of problems [3]. We conducted experiments with different types of traffic to evaluate the convergence time and the trade-off between the number of DPI engines and the network load with various costs.

Our cost-based method provides the number and the locations of the DPI engines to be deployed. It can be used at design time to lower costs and reduce capital expenditures by utilizing the appropriate number of software solutions rather than adding offload hardware. It can also be used at runtime to adapt dynamically deep packet inspection capabilities.

The rest of the paper is organized as follows. First, Section II presents related work. Then, Section III details the method we propose, while experimentation results are analyzed in Section IV. Finally, Section V concludes this paper.

This work is partially supported by French FUI RAVIR project.

II. Related work

In recent years, the concept of network virtualization, considered as the way to evolve towards new network architectures, has attracted significant attention [4]. It consists in clearly separating the management of the network into two domains: infrastructure management for physical resources and service management for virtual network on top of the physical infrastructure. This concept has in particular resulted into the concept of Software-Defined Networking (SDN) [5], where the separation of the control plane from the data plane through open API like OpenFlow [6] enables to control both physical and virtual network equipment [7]. Besides, the OpenFlow protocol provides a mean to atomically manage flows, aggregate filter, block or redirect them on a vendor-agnostic basis.
Very recently, a new concept has emerged: Network Functions Virtualization (NFV) [2]. This initiative from the biggest service providers is highly complementary to SDN. It aims to shorten service deployment lifecycle by leveraging standard IT virtualization technology to consolidate many network equipment types onto industry standard high volume servers, switches and storage, which could be located in Datacenters, Network Nodes and in the end user premises. The virtualization of the network appliances concerns firewalls, caches, ciphers, load balancers, intrusion detection systems etc. Several recent works address the performances and the support of network equipment [8] and Deep Packet Inspection [9] on commodity hardware.
Network and network functions virtualization are also pushed by the convergence of computation, storage and networks in cloud computing. A lot of recent work in the literature only concerns the placement of Virtual Machines without an integrated view of computation, storage and networks. Several techniques to optimize their placement with respect to server load balancing or energy saving have been proposed [10], [11]. To the best of our knowledge, this paper presents the first method that addresses the integrated optimization of virtualized network functions deployment (DPI engines in our case). Contrary to the problem of placing virtual machines, that are communication end-points, we consider the placement of functions inside the network with end-points that cannot be arbitrarily moved. This placement induces flow redirections inside the considered network and thus increases the global network load.

III. Optimizing the placement of virtualized DPI engines in SDN

This section presents the method we propose, describing its formalization an then detailing its implementation with a genetic algorithm.

A. Problem description

The problem we address in this paper can be stated as follows: for a given network infrastructure and a given traffic matrix find a DPI engine deployment that minimizes the overall cost of the deployment. This cost is the result of a joint optimization that minimizes i) the number of DPI engines, ii) the overall network load induced by flow redirections through the DPI engines, and iii) different operational constraints. These constraints concern financial costs such as the cost associated to a deployed DPI engine (e.g. license price, CPU utilization, energy consumption...), the cost associated of network resources (e.g. network total cost of ownership, capacity of the network to absorb new traffic), and the cost of penalties due to the incapacity to analyze a flow. The constraints also include management limits such as maximum number of engines to be deployed, the maximum used bandwidth per link (to be able to absorb peaks) and the maximum unallocated flows.

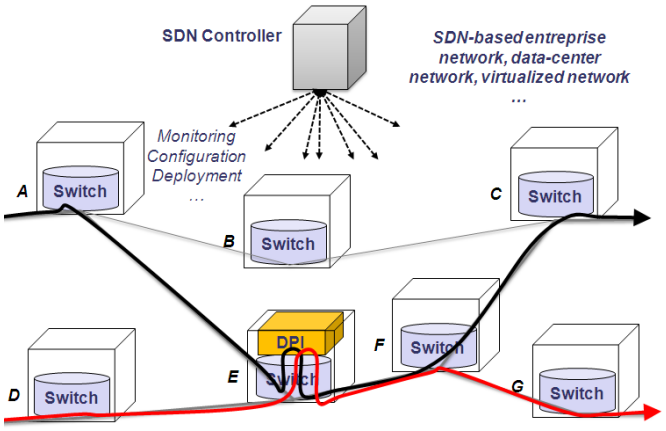


Fig. 1. The objective of minimizing the number of DPI engines is orthogonal to the objective of minimizing the network load.

The two main objectives, that are minimizing the number of DPI engines and minimizing the network load, are in fact orthogonal. Indeed, all the flows have to go through at least one DPI engine to be analyzed. When the number of DPI engines is small, the paths tend to be elongated. Therefore, minimizing the number of engines increases the additional used bandwidth. On the contrary, minimizing the used bandwidth increases the number of DPI engines to be deployed. Fig. 1 illustrates the orthogonality of the objectives. The optimal number of DPI engines, that is 1, induces the redirection of the black flow and thus the increase of network usage. On the contrary, the optimal network load, that corresponds to the shortest path, requires the deployment of at least 2 DPI engines, one on each shortest path.

B. Problem formalization

We formalize the problem described above as follows. We define the representation of a solution. For a topology of n nodes, x is an array of n bits/booleans ([0, 1, 0, 0, ...]) representing the presence of a DPI engine in node i by a 1 and its absence by 0. The solution that corresponds to the deployment on Fig. 1 is: [0, 0, 0, 0, 1, 0, 0], as there is only 1 DPI engine in node E.
The fitness function $F(x)$, that is the global cost function to minimize, is composed of three cost functions for the three objectives to minimize: i) the number of used DPI engines (Eq. 2), ii) the global network load (Eq. 3), and iii) the number of flows that cannot be analyzed (Eq. 4):

$$F(x) = f_{DPI}(x) + f_{bw}(x) + f_{unalloc.}(x) \quad (1)$$

The function $f_{DPI}(x)$ represents the cost to deploy $n(x)$ DPI engines which have a unitary cost (license, energy...) ω_{DPI}

TABLE I. SYMBOL DESCRIPTIONS.

Symbols	Descriptions
x	A set of DPI engines linked to the network nodes
$F(x)$	Fitness function
$f_{DPI}(x)$	DPI cost function
$f_{bw}(x)$	Additional bandwidth cost function
$f_{unalloc.}(x)$	Unallocated flow cost function
ω_{DPI}	DPI weight/cost
$n(x)$	Quantity of used DPI engines
N	Threshold for the number of used DPI engines
ω_{bw}	Additional used bandwidth weight/cost
$bw(x)$	Additional used bandwidth
bw_i	Used bandwidth on link i
BW	Threshold for the used bandwidth per link, in percentage
$\omega_{unalloc.}$	Unallocated flow weight/cost
$u(x)$	Quantity of flows unallocated to any DPI engine
U	Threshold for the number of unallocated flows

with the constraint to have maximum N engines:

$$f_{DPI}(x) = \begin{cases} \omega_{DPI} * n(x) & \text{if } n(x) \leq N \\ \infty & \text{if } n(x) \succ N. \end{cases} \quad \text{with } n(x) = \sum_{i=0}^{i<n} x_i. \tag{2}$$

The function $f_{bw}(x)$ represents the cost to increase the global network load of $bw(x)$ units of bandwidth, with a cost ω_{bw} per unit of bandwidth (e.g. network total cost of ownership). It includes a strong constraint with the threshold BW that defines the maximum percentage of used bandwidth bw_i on each link i of the network. This threshold enables network managers to choose the over-provisioning rate and network capacity to be able to absorb traffic variations or new demands:

$$f_{bw}(x) = \omega_{bw} * bw(x) \text{ with } bw_i \leq BW \tag{3}$$

$bw(x)$ is the difference between the total bandwidth used with DPI engines (that generate redirections and thus path elongations) and the total bandwidth used when there is no DPI engine.

Finally, the cost function $f_{unalloc.}(x)$ represents the sum of the penalties $\omega_{unalloc.}$ for the $u(x)$ flows that cannot be analyzed with the constraint to have maximum U flows that are not allocated to a DPI engine:

$$f_{unalloc.}(x) = \begin{cases} \omega_{unalloc.} * u(x) & \text{if } u(x) \leq U \\ \infty & \text{if } u(x) \succ U. \end{cases} \tag{4}$$

By default, the threshold U is equal to 0, that is all the flows must be analyzed.

C. Problem solving

The problem described in the above sections belongs to the family of the uncapacitated facility location problems (UFLP). They involve locating an undetermined number of facilities to minimize the sum of the (annualized) fixed setup costs and the variable costs of serving the market demand from these facilities. ULFPs are known to be NP-hard for general graphs. However, it has been shown by [3] that approaches based on genetic algorithms (GA) scale up better than approaches based on linear programming for solving UFLPs. We thus use a GA approach and adapt it to the specificities of our multi-objective networking problem.

The GA is a search procedure based on the principle of evolutionary algorithms. It combines the exploitation of past results with the exploration of the new areas of the search space by using the *survival-of-the-fittest* technique coupled with a structured yet randomized information exchange. Fig. 2 presents the GA diagram.

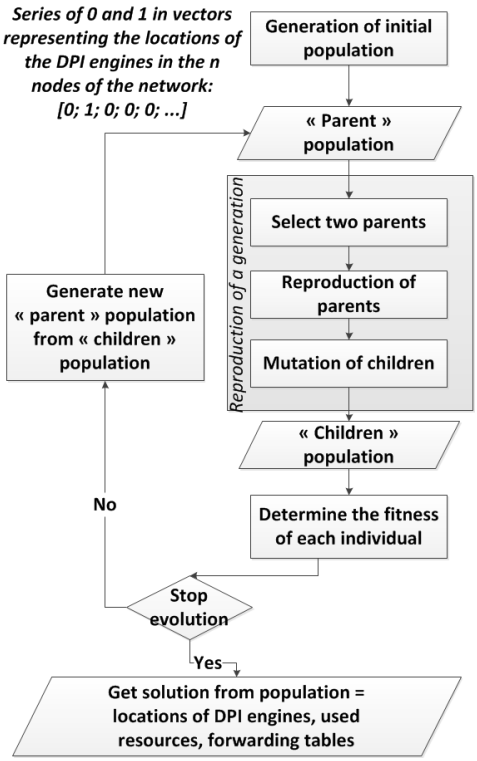


Fig. 2. Genetic Algorithm diagram.

1) Initial Population: the Initial population is a random set of potential solutions to the problem. The population is also called a chromosome and its elements are called genes according to the terminology from genetic engineering. In our case, we use the binary genes defined before to represent the deployment of DPI engines in different nodes of the network. On the initial population and the next generation, selection, crossover, and mutation genetic operators are iteratively applied in the GA.

2) Selection: The selection operation selects good results among the chromosomes by using the fitness function $F(x)$. The calculation of the fitness value of a gene x is detailed in Sec. III-D. The fitness function is used to rank the quality of the chromosomes. A chromosome with a smaller value has a higher probability of contributing to one or more offspring in the next generation.

3) Crossover: A crossover consists in swapping part of the information between a pair of chromosomes to obtain a new chromosome. We use a simple crossover. First members of the newly reproduced chromosomes in the reproducing pool are bred at random with a probability $p_{crossover}$. Second, each pair of chromosomes undergoes crossing over by inclusively swapping the k first elements of the first chromosomes, k being randomly chosen between 1 and the length of the chromosomes, with the k first elements of the second chromosomes. Two new chromosomes are thus obtained.

4) Mutation: A mutation consists in slightly randomly altering to get a new chromosome. The mutation operator is used to introduce a new genetic material. A chromosome is mutated

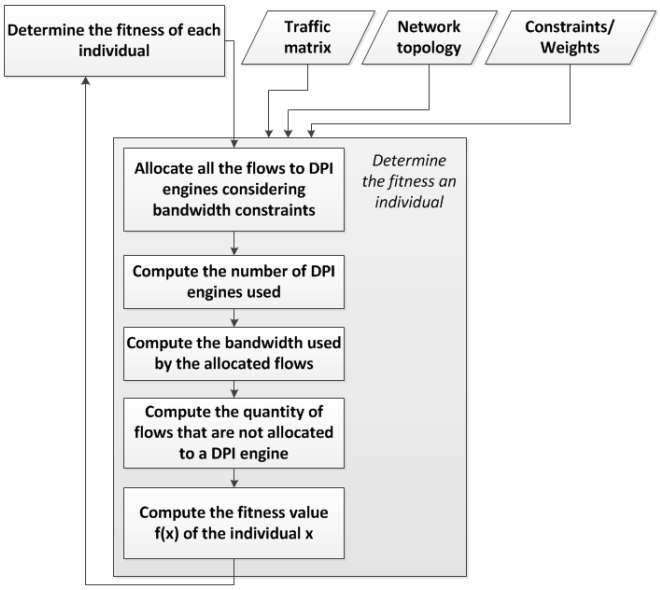


Fig. 3. Determine the fitness value.

with a probability $p_{mutation}$. We introduce a mutation operator specific to our problem. It mutates 1 elements in 0 elements with a probability p_1 and 0 elements in 1 elements with a probability $p_0 \leq p_1$. Hence, it tends to increases the convergence of the algorithm reducing the number of DPI engines (1 elements) yet exploring new areas when mutating 0 elements in 1 elements.

In each new generation, a set of chromosomes is created by using information from the previous ones. The evolution stops after a specified number of iterations. The chromosome with the smallest fitness value is selected as the solution of the multi-objectives DPI engine allocation problem. The algorithm, more specifically the module in charge of evaluating the fitness values, also provides the forwarding rules to push to the network equipment.

D. Fitness value calculation

In the GA, the fitness value of each chromosome is evaluated at every evolution iteration for ranking. The fitness value of a chromosome indicates its quality with respect to the multiple objectives. The smaller the fitness value, the better the quality of the chromosome.

The calculation of the fitness value of a chromosome x involves several steps (Fig. 3). x is an array of n bits representing the presence of a DPI engine in node i by 1 and its absence by 0, n being the number of network nodes able to host a DPI engine. Based on the deployment of DPI engines represented by x, the first step consists in allocating all the flows to a DPI engine. The inputs of the computation are: the traffic matrix, the network topology and the constraint on the maximum used bandwidth on each link (BW). This operation results in a set of paths that go from a source node to the corresponding destination node through a DPI engine, taking into account the bandwidth capacities of the links. The second step corresponds to the calculation of the cost function $f_{DPI}(x)$. If x contains less than N DPI engines, then it is equal to the cost ω_{DPI} of a DPI engine multiplied by the number of DPI engines, else it is infinite, which eliminates the solution. The third step consists in evaluating the cost function $f_{bw}(x)$ which is equal to the cost ω_{bw} of a unit of used link capacity multiplied by the overall network load $bw(x)$ expressed in units of bandwidth. The constraint BW on the maximum used bandwidth per link has been taken into account during the first step. Then, the forth step corresponds to the evaluation of the penalty cost function $f_{unalloc.}(x)$. The number of flows that have not been allocated to any DPI engine is retrieved from the set of paths from the first step. The penalty cost $\omega_{unalloc.}$ is multiplied by this quantity if it is lower than the threshold U, otherwise the penalty value is infinite to eliminate the solution. Finally, the fifth step concerns the evaluation of the fitness value of the solution x by summing the three cost functions.

TABLE II. EVALUATION PARAMETERS.

Parameters	Values
DPI weight	ω_{DPI} varies
Threshold for the number of used DPI engines	$N = \infty$
Additional used bandwidth weight	$\omega_{bw} = 10$
Threshold for the additional used bandwidth	BW varies
Unallocated flow weight	$\omega_{unalloc.} = 1000$
Threshold for the number of unallocated flows	$U = 0$
Probability of 1 elements mutation	$p_1 = 1/3$
Probability of 0 elements mutation	$p_0 = 1/10$
Initial population size	100
Number of evolutions	100

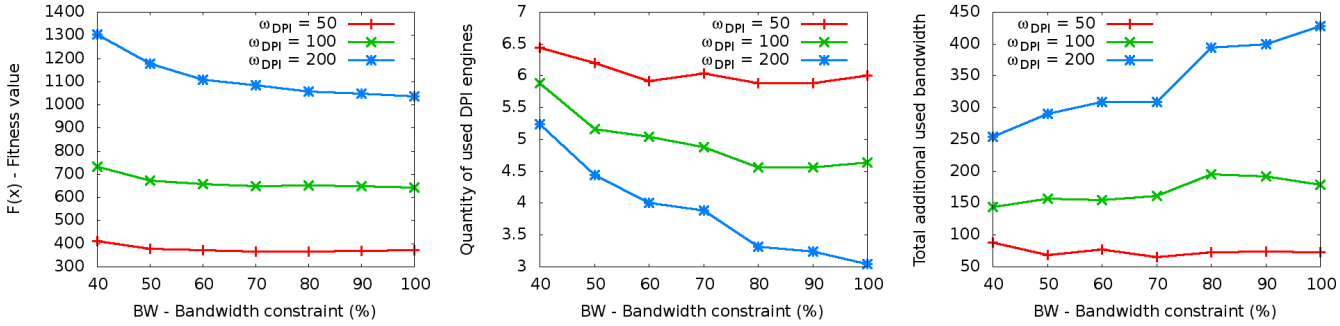
IV. SIMULATIONS AND VALIDATION

In order to evaluate the performance and the behavior of our placement algorithm, we have implemented the multi-objective GA and run simulations on two types of traffic. This section presents the simulation set-up and the experimental results.

A. Simulation set-up

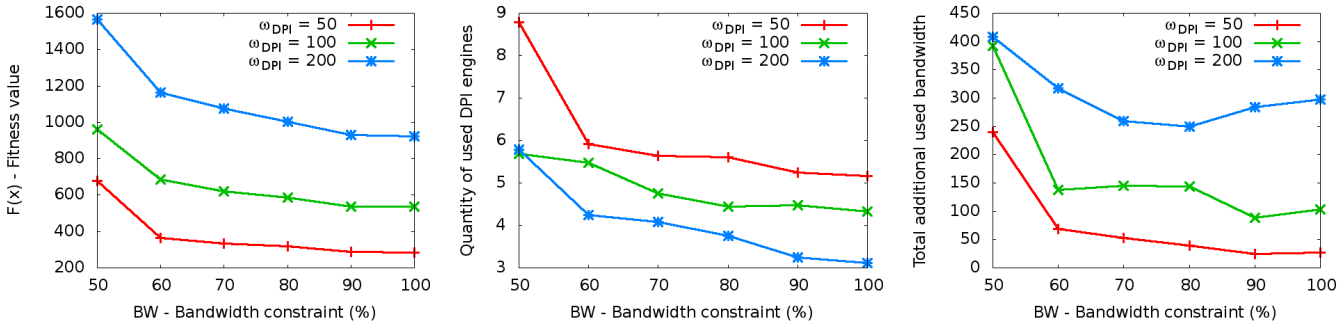
We used the JGAP framework [12], which is a Genetic Algorithms and Genetic Programming component provided as a Java framework. It provides basic genetic mechanisms that can be easily used to apply evolutionary principles to problem solutions. We implemented the component in charge of computing the different cost functions based on a given traffic matrix, a network topology and operation constraints (Fig. 3). We also instantiated and adapted the framework to support binary genes to represent DPI engine deployments and binary mutation (see Sec. III-D). Finally, we implemented a greedy algorithm to perform the allocation of the flow to a deployed DPI engine. This greedy algorithm is a fair approximation for this NP-hard problem when the traffic is largely lower than the global network capacity [13]. For each flow, it considers one by one the different deployed DPI engines. For each of them, it evaluates the shortest path between the source and the destination that goes through the considered DPI engine using the Constrained Shortest Path First (CSPF) algorithm [14] to take into account the available capacity on the links. The shortest path is selected for the flow: the corresponding DPI engine is assigned to the flow and the capacity on the links of the path are updated in order to consider the bandwidth used by the flow. This operation is done iteratively for each flow of the traffic matrix. The result enables to evaluate the fitness values as described in Sec. III-D.

In order to evaluate the performance and the behavior of our placement algorithm, we vary two parameters that can be



(a) Fitness values. (b) Quantity of DPI engines. (c) Additional network load.

Fig. 4. Dense mice traffic.



(a) Fitness values. (b) Quantity of DPI engines. (c) Additional network load.

Fig. 5. Heterogeneous random traffic.

considered as operation constraints: i) the cost ω_{DPI} of a DPI engine (for instance the license price) and ii) the maximum used bandwidth BW on each link of the network (for network operators to define their over-provisioning rate). The rest of the parameters are presented in Tab. II. In the following section, the results for a given configuration correspond to the average of 25 experiments.

Finally, we use a network topology that comprises 14 nodes. Their connectivity degree is equal to 4 and their links have a capacity of 10 units of bandwidth. Such a topology represents an overlay network, interconnected Virtual Machines supporting a service or part of an Information System. These 14 nodes are also source and destination of traffic. We consider two types of traffic matrices to evaluate the behavior of the optimization algorithm:

- **Dense mice traffic**: 13 degree graph with bidirectional link capacities equal to 0.5 units of bandwidth. It represents 182 flows.
- **Heterogeneous random traffic**: 3 degree graph with bidirectional link capacities randomly distributed between 0.5 and 4.5 units of bandwidth. It represents 42 flows.

The first traffic matrix emulates numerous small flows between the peers (e.g. VoIP or chat communications) while the second one emulates heterogeneous flows with a moderate global demand (e.g. classic office traffic).

B. Simulation results

This section presents and analyzes the results obtained through several experiments.

For the considered parameters, the execution of the algorithm lasts in average 18 seconds and converges around 10.1 iterations. Increasing the initial population size would speed up the convergence, but in the meantime increase the execution duration. The size of the network and the number of flows are also dimensioning parameters.

Fig. 4 presents results for DPI engine allocation for three different DPI costs ω_{DPI}, 50, 100 and 200 respectively, with the *dense mice traffic matrix*. Fig. IV-A shows the fitness value $F(x)$, that can be considered as the total cost of the deployment, for different bandwidth constraints on the links. The fitness value decreases while the constraint is relaxed. For example, allowing a maximum link capacity of 90% *reduces the total cost by* 19.2% compared to a threshold of 40% when the DPI cost equals 200. The fitness value is the combination of the DPI cost function $f_{DPI}(x)$, the used bandwidth cost function $f_{bw}(x)$ and the unallocated flow cost function $f_{unalloc.}(x)$. The number of flow unallocation is not presented in a figure. Indeed, it is equal to zero for all the

considered DPI costs and used bandwidth constraints. For this set of simulations, the threshold U for unallocated flows is equal to 0 (Tab. II). Therefore, any DPI engine deployment that has at least one flow allocation violation has an infinite fitness value and is thus rejected from the set of solutions. Fig. IV-A and Fig. IV-A show the number of used DPI engines, that is the value of the DPI cost function divided by the DPI cost, and the total additional used bandwidth respectively. We can observe that the number of used DPI engines decreases while the used bandwidth increases with the relaxation of the constraint on the used capacity of the links. This phenomenon is amplified for high DPI costs ω_{DPI}. For example, for $\omega_{DPI} = 200$, the number of the DPI engines changes from 5.24 to 3.04 when the bandwidth constraint BW goes from 40% to 100%. It corresponds to *a reduction of* 42%. On the other hand, the additional used bandwidth increases. Indeed, when there is few DPI engines, the length of the paths tends to increase. An initial path that is situated far from any deployed DPI engine in the network may be largely deviated and hence will consume much more bandwidth. Our algorithm enables to find out a DPI engine deployment that satisfies the trade-off between the minimum number of engines and the minimum network load for a considered set of costs and thresholds. For this traffic, the trade-off remains almost the same for the different bandwidth constraints when the cost of a DPI engine is low ($\omega_{DPI} = 50$). Our method enables to equally distribute the micro-flows, the DPI cost function remaining low despite the number of used engines.

Fig. 5 presents the results obtained with the *heterogeneous random traffic matrix*. The bandwidth constraint BW on each link starts at 50%. Below this threshold, the results are not exploitable. Indeed, the biggest flow corresponds to a bandwidth of 4.5 units of bandwidth, while the link capacity is set to 10 units. Therefore, a threshold below 45% generates at least one unallocated flow and thus an infinite fitness value. As for the previous traffic matrix, the fitness value, which represents the total cost of the deployment, decreases while the constraints on the used link capacity is relaxed (Fig. IV-A). *It is reduced by* 41.1%, 44.3% *and* 58.0% between a threshold of 50% and 100% for a DPI cost ω_{DPI} equals to 200, 100 and 50 respectively. The heterogeneity of the flow bandwidths induces the following phenomenon. While the number of the DPI engines decreases with the relaxation of the bandwidth constraints BW (Fig. IV-A), the additional used bandwidth decreases for almost all the cases. This is due to the fact that some of the flows have a large bandwidth, up to 4.5 units, with respect to the allowed link capacity BW. Therefore, relaxing the constraint enables to reduce the number of DPI engines, but also to reduce the network load by allowing several flows to share links. For such a type of traffic, our algorithm minimizes both the number of used DPI engines and the network load.

V. Conclusion

In today's IT systems, cyber security requires situational awareness that has to be fine-grained, flexible, adaptable and cost optimized. The emergence of new networking technology trends, like NFV and SDN, provide new ways to build cyber security tools. DPI engines can be virtualized and deployed on commodity hardware as a piece of software. In this paper, we have proposed a cost-based method that enables to find out a DPI engine deployment that satisfies the trade-off between the minimum number of engines and the minimum network load for a considered set of costs engine, bandwidth, violations) and operational constraints (provisioning rate). Our method is based on genetic algorithm. Its outputs are both the locations of the engines to be deployed and the routing tables for flows to be analyzed by them. We conducted several experiments with different types of traffic (dense mice and random traffic) and different DPI engine costs to evaluate its performances. The results have shown that the method provides a trade-off between the number of engines and the network load to minimize the global cost of the deployment. Moreover relaxing the constraint on the used capacity per link, that is, the provisioning rate, enables to drastically reduce the global cost of deployment, up to 58%.

Future works along these lines include the consideration of delay constraints on the flows since redirections and flow analysis may increase it. We also plan to enrich our model to integrate a more accurate performance model of DPI probes with regards to the rules that they execute. Finally, we intend to study dynamic systems where the deployment of additional DPI engines are incremental based on an existing setup.

References

[1] NIST *et al.*, *NIST Special Publication 800-53 Revision 4 Recommended Security Controls for Federal Information Systems and Organizations*, Feb. 2013.

[2] M. Chiosi *et al.*, "Network Functions Virtualization - An Introduction, Benefits, Enablers, Challenges and Call for Action," ETSI NFV, Oct. 2012.

[3] M. Maric, "An efficient genetic algorithm for solving the multi-level uncapacitated facility location problem," *Computing and Informatics*, pp. 183–201, 2010.

[4] D. Mcdysan, "Software Defined Networking opportunities for transport," *IEEE Communications Magazine*, vol. 51, no. 3, pp. 28–31, 2013.

[5] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in Software-Defined Networks," *IEEE Internet Computing*, vol. 17, no. 2, pp. 20–27, 2013.

[6] K. Phemius and M. Bouet, "Implementing OpenFlow-based resilient network services," in *IEEE 1st International Conference on Cloud Networking (CLOUDNET)*, 2012.

[7] R. Sherwood, G. Gibb, K.-k. Yap, G. Appenzeller, M. Casado, N. Mckeown, and G. Parulkar, "FlowVisor : A network virtualization layer FlowVisor: A network virtualization layer," *OpenFlow Switch*, p. 15, 2009.

[8] G. Lu, R. Miao, Y. Xiong, and C. Guo, "Using CPU as a traffic co-processing unit in commodity switches," in *Proceedings of the first workshop on Hot topics in software defined networks (HotSDN)*. ACM, 2012, pp. 31–36.

[9] F. Gringoli, A. Este, and L. Salgarelli, "MTCLASS: Traffic classification on high-speed links with commodity hardware," in *IEEE International Conference on Communications (ICC)*, 2012, pp. 1177–1182.

[10] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera, "A stable network-aware VM placement for cloud systems," in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2012.

[11] J. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM placement and routing for data center traffic engineering," in *INFOCOM, 2012 Proceedings IEEE*, 2012.

[12] K. Meffert *et al.*, "JGAP - Java Genetic Algorithms and Genetic Programming Package." [Online]. Available: http://jgap.sf.net

[13] W. Xiao, B.-H. Soong, C. Law, and Y.-L. Guan, "Evaluation of heuristic path selection algorithms for multi-constrained QoS routing," in *IEEE International Conference on Networking, Sensing and Control*, 2004.

[14] C. Gen-Huey and H. Yung-Chen, "Algorithms for the constrained quickest path problem and the enumeration of quickest paths," *Computers & operations research*, vol. 21, no. 2, pp. 113–118, 1994.