Providing Cognitive Components with a Bidding Heuristic for Emergent NFV Orchestration

Frederico Schardong*†, Ingrid Nunes*, Alberto Schaeffer-Filho*
*Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil
†Instituto Federal do Rio Grande do Sul, Rolante, Brazil
Email: {fschardong, ingridnunes, alberto}@inf.ufrgs.br

***Abstract*—Network function virtualisation (NFV) decouples network functions from the underlying hardware by means of virtualisation, thus enabling the replacement of proprietary middleboxes by software components that can be dynamically deployed and configured on demand. The selection of the most appropriate virtual network functions (VNFs) to achieve a particular objective, and the decision on where to deploy these VNFs and through which paths they will communicate, are the responsibilities of an NFV orchestrator. In this paper, we propose to orchestrate VNFs using interacting cognitive components structured with the BDI architecture, leading to emergent solutions to address network challenges. More specifically, we extend a previously proposed reverse auction protocol and propose a novel bidding heuristic that can be used to make decisions regarding the orchestration tasks. We validate our model in a DDoS attack case study, in which we demonstrate that our components can successfully mitigate the attack.**

I. Introduction

The increasing complexity of computer networks together with highly volatile requirements cause network management to become a challenging and time-consuming task. Network functions—or middleboxes—are often employed to enable networks to successfully meet specific requirements. For example, a firewall and an intrusion prevention system can help maintain security, and a caching component can improve performance. However, rising service-driven networks with short-lived demands render the clumsy procedures of physically installing and configuring vendor-specific middleboxes into a hard task.

Network function virtualisation (NFV) [1], [2] rises on top of middlebox virtualisation as a means of managing future networks. Traditional middleboxes with cumbersome deployment procedures are replaced by virtual network functions (VNFs). Their functionality is decoupled from the underlying hardware through virtualisation. Differently from middleboxes, multiple virtualised network functions can share a single physical machine, enabling the use of computational power that would be otherwise lost in proprietary middleboxes [3].

Despite the provision of a higher return on both capital and operational expenditures over traditional middleboxes, the virtualisation of network functions introduces new challenges to network operators. In the context of NFV, activities such as deciding which VNFs should be employed, where they should be physically located and how they are interconnected are achieved using algorithms developed by network operators. These activities are referred to as *selection*, *placement* and *chaining*, respectively, and are the main challenges of this architecture [4], [5]. The software component responsible for performing these activities is called an *NFV orchestrator*.

978-1-5386-3416-5/18/$31.00

Although VNFs can be easily added and removed of a network function virtualisation infrastructure point-of-presence (NFVI-PoP), it is not straightforward to decide which NFVI-PoP is the most appropriate to host a VNF. Resource allocation of NFVI-PoPs and proximity to other VNFs are key to determine the ideal physical location of a virtualised middlebox. Previous work has tackled the placement and chaining problems from different perspectives, such as genetic algorithms [6], [7], [8], [9] and greedy heuristics [10], [11], [12], [13]. Although these approaches have made progress towards an automated NFV orchestrator, they do not provide a solution for placement and chaining in *dynamic* scenarios and neither address the selection problem. Traditionally, the selection problem is manually dealt with by a network operator, who selects the most appropriate set of VNFs and uses this as input for placement and chaining automated approaches.

As opposed to existing work, which assumes that the selection problem is outside the boundaries of an NFV orchestrator, our approach addresses the three aforementioned problems. Our orchestrator is composed of cognitive software components, which interact in such a way that a solution to specific network requirements emerges. Our components, or agents, are structured using the belief-desire-intention (BDI) architecture [14], a widely used cognitive architecture that implements practical reasoning. This idea was introduced in our previous work [15], which integrated BDI agents, capable of bidding on the allocation of VNFs, to the NFV architecture of ETSI [5]. We now build on our previous work by extending the proposed reverse auction mechanism used to orchestrate VNFs. Moreover, we propose a bidding heuristic that takes into account many factors that influence the selection, placement and chaining problems. The resulting bid is used to make decisions regarding what task should be performed by which VNF and where. We evaluate our approach using a distributed denial of service (DDoS) scenario, in which BDI agents provided with our solution collaborate to mitigate the attack by managing the selection and deployment of VNFs.

In summary, our contributions in this paper are: (i) an extension of a reverse auction protocol for NFV management using BDI agents; (ii) a bidding heuristic, which is used

to evaluate whether a particular task must be executed by a particular VNF and where this VNF should be placed; and (iii) a novel testbed that integrates a network emulator, `Mininet` [16], and a BDI agent platform, `BDI4JADE` [17].

II. Background

In this section, we provide the background needed to understand our work.

A. NFV Orchestration Problems

As introduced, there are three main problems associated with NFV orchestration. This first is the *selection* problem, which consists of logically selecting and connecting virtual network functions (VNFs) in an ordered manner to provide some service [18]. The solution to the selection problem is a virtualised network function forwarding graph (VNF-FG), informing which VNFs should be used and their connections. Most of the existing NFV orchestration approaches assume that the VNF-FG is provided as an input to their solutions.

The second, most frequently addressed, problem is *placement*, involving the allocation of the VNFs of a VNF-FG into a physical network. Placing VNFs in NFVI-PoPs is bounded by the availability of the hardware resources provided by the NFVI-PoP [19], [20]. Typically, the placement algorithm receives as input a physical network, a candidate VNF and an allocation goal. Alternatively, the input can be multiple VNFs, with the aim of allocating them all at once. In the latter case, the result is a mapping that indicates in which physical machine each VNF should be placed. The placement algorithm uses the allocation goal to help decide where to allocate the VNF, representing a general preference given by a network operator. Examples of goals are the minimisation of used NFVI-PoPs or the reduction of network usage.

Lastly, the *chaining problem* consists of physically defining the path between VNFs as specified by the forwarding graph [21], [22]. The chaining solution creates virtual paths between VNFs ensuring that communication constraints are met (*e.g.*, minimum latency and required bandwidth). Moreover, this solution must be compatible with the allocation goal, *e.g.*, to reduce power usage or distribute workload. Algorithms that provide such solutions receive as input a physical network, a VNF-FG and location to place each VNF, and an allocation goal. Differently from the VNF and virtual link requirements, which are particular to each single element of the VNF-FG, the allocation goal defines requirements for the entire VNF-FG.

These described problems are often subject to specific constraints, *e.g.*, hardware limits for placement and network constraints for chaining. However, the allocation goal is directly related to both placement and chaining problems. One can have multiple allocation goals such as to minimise the number of deployed VNFs and reduce the number of active NFVI-PoPs. Focusing on such shared characteristics, algorithms that solve together the placement and chaining problems have been proposed [21], [23], [24].

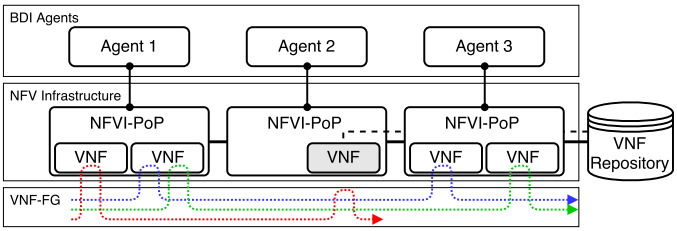

Fig. 1. An NFV architecture based on BDI agents. The blue and green dotted lines represent VNF-FGs that traverse multiple VNFs, while the red line indicates a new VNF-FG, which will be created after the VNF highlighted in grey is downloaded from the repository.

B. BDI-based NFV Orchestration

Our previous work [15], which is used as the basis for our present proposal, is an extension of the ETSI's NFV architecture [2], in which we replaced a centralised orchestrator [5] and VNF manager for BDI agents. BDI agents are cognitive components structured with an architecture [14] composed of: (i) *beliefs*, which represent the agent's knowledge about itself and its environment; (ii) *desires*, which are goals to be achieved; and (iii) *intentions*, which represent the commitment to achieve a goal using a plan (a set of actions). A reasoning cycle controls agent behaviour using these BDI components.

In our decentralised orchestrator, each agent controls an NFVI-PoP and VNFs to be deployed, as illustrated in Figure 1. When a goal must be achieved—*e.g.*, to deal with a significant increase in the traffic on a link—an auction takes place to decide which NFVI-PoP and VNF will achieve this goal. Each agent, representing a candidate NFVI-PoP, places a bid corresponding to the cost of achieving the goal in that NFVI-PoP. Such a cost is the combination of deploying the VNF (if needed) and consuming computational and network resources. The agent that placed the lowest bid wins the auction and will be responsible for achieving the goal. We assume that in an initial network configuration there is at least one proactive agent that triggers the first auction in the face of a challenge, *e.g.*, an agent controlling an NFVI-PoP with a deployed link monitor. These are *reverse* auctions [25] because agents are auctioneers wanting to *buy a resource* rather than sell.

Because a goal can be achieved by different VNFs, the auction is a means of selecting which VNF will achieve a goal (selection problem). Furthermore, given that bids can be placed by agents representing different NFVI-PoPs using the same VNF, the auction is also a means of selecting the location of a VNF (placement problem). Finally, our solution assumes that bidders assess the path between them and the auctioneer when placing a bid (chaining problem).

Although our previous work evidenced the potential of our approach, it relied on a preliminary solution for placing bids. Given that the bid is fundamental to make orchestration decisions and achieve the expected emergent behaviour, in this paper we present a sophisticated auction mechanism and a bidding heuristic that take into account a wide range of factors, including network latency and available memory and bandwidth, in order to adequately coordinate VNFs.

TABLE I
NOTATION FOR RESOURCES AND NETWORK MEASUREMENTS.

Symbol	Set	Description
U	R_C	Number of CPU cores.
M	R_C	Amount of RAM memory in GB.
D	R_C	Amount of disk memory in GB.
B	R_N	Bandwidth of a link in MB/s.
L	NM	Latency of a link in ms.
P	NM	Percentage of packet loss of a link.
H	NM	Number of hops in a path.

III. REVERSE-AUCTION-BASED NFV ORCHESTRATION

Before detailing our auction mechanism and bidding heuristic, we first formalise, in the next section, key concepts that are used in later sections. We then introduce the *NFV-A* protocol, which specifies messages exchanged in auctions. Next, we describe the algorithm used by bidder agents to place bids and how an auctioneer evaluates bids to select a winner.

A. Notation and Definitions

In a computer network, nodes have a set of their available *resources*, $r_i \in R$. Each r_i can be either a computational resource in R_C or a network resource in R_N. Therefore, $R = R_C \cup R_N$ and $R_C \cap R_N = \emptyset$. Each type of computational resource and network resource are represented by r_C and r_N, respectively. In this work, we assume that there are four types of resources, from which three are computational resources and one is a network resource. The amount of these resources are represented using the notation introduced in Table I (first four rows), which also indicates to which set they belong. Moreover, there are measurements that can be collected from the network. The last rows of Table I correspond to the network measurements, $nm \in NM$, associated either with a network link or a path.

In the context of NFV, a node corresponds to a point-of-presence that has NFV-enabled infrastructure (NFVI-PoP), *i.e.*, hardware infrastructure that can host VNFs. For simplicity, we assume that their hardware resources are homogeneous in specification (*e.g.*, processor frequency and memory latency) but not in size (*e.g.*, amount of CPU and memory). Each NFVI-PoP p is characterised by a set of functions that give its specification and current state, detailed below, where NGB_p is the set of NFVI-PoP that are neighbours of p.

- $total_p : R_C \to \mathbb{R}_{>0}$: gives the total amount of a particular computational resource of p.
- $free_p : R_C \to \mathbb{R}_{\geq 0}$: gives the current available amount of a particular computational resource of p.
- $value_p : NPoP \times R_N \cup NM \nrightarrow \mathbb{R}_{>0}$: gives the value of a particular network resource or measurement of a link between p and a given NFVI-PoP $p' \in NPoP$, neighbour of p, that is, $\mathrm{dom}\, value_p = NGB_p \times R_N$.

NFVI-PoPs provide different amounts of resources, which are allocated and consumed by VNFs when instantiated. While NFVI-PoPs have an specification of *available* resources, VNFs are associated with *required* resources. The requirements of a VNF v are given by the following function.

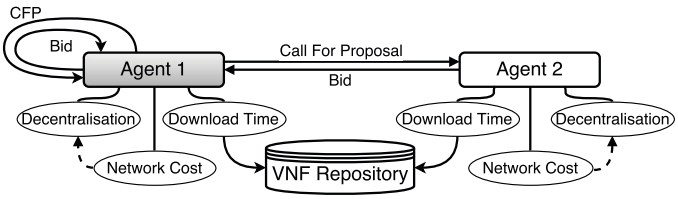

Fig. 2. Overview of the auction process.

- $req_v : R \to \mathbb{R}_{>0}$: gives the amount of a particular resource required by v.

We use p, p', p'', ... to denote NFVI-PoPs and v, v', v'', ... to denote VNFs. Using this introduced notation we proceed to the description of our NFV auction protocol.

B. NFV-A Protocol

Our NFV orchestrator consists of a collection of BDI agents, and each of which controls an NFVI-PoP. The collaboration between agents and, correspondingly, VNFs occurs when an agent has a goal that cannot be achieved by this agent. It then becomes an auctioneer and starts the process of delegating this goal to other agents. Agents (bidders) receive a call for proposal (CFP) [15] from the auctioneer and evaluate the cost of achieving such a goal.

An overview of this process is presented in Figure 2, in which Agent 1 starts an auction by sending CFPs to candidate agents, including itself. Although Agent 1 cannot achieve the goal considering its currently deployed VNFs, it might conclude, as result of the auction process, that it is the most suitable NFVI-PoP to deploy a new VNF to achieve this goal.

In our earlier work, the auction target was simply a goal and bids were given as a single value. The lowest bid was the winner. Our extended protocol NFV-A enriches goals with constraints, which capture particularities of the functions to be performed. Moreover, bids are now composed of three components (shown in Figure 2), which provide the auctioneer with more information to reason, considering its preferences, about the most adequate agent to delegate a goal. We define constrained goal and the NFV-A protocol as follows.

Definition 1 (Constrained Goal): A constrained goal g_i^C is a tuple $\langle g_i, C \rangle$, where g_i is a goal to be achieved and C is a set of constraints that must be satisfied to achieve this goal. Each $c \in C$ is a tuple $\langle n, op, val \rangle$, where $n \in R_N \cup NM$ is a network resource or measurement, op is a comparison operator (*e.g.*, $>$ or $\leq$) and val is a value.

Example 1: Consider the goal $isTrafficMalicious$ to be achieved, generated as a consequence of an unusual spike of 500 MB/s on the incoming traffic of an agent. Consequently, a constraint to achieve this goal is to have bandwidth higher than or equal to 500 MB/s. This is complemented by a design-time-specified constraint regarding packet loss, which should be lower than 20%. The constrained goal is, therefore, $\langle isTrafficMalicious, \{\langle B, \geq, 500 \rangle, \langle P, <, 20\% \rangle\}\rangle$.

Definition 2 (NFV-A): Let $\mathcal{G} = \{g_1^C, ..., g_n^C\}$ be a set of constrained goals to be auctioned, an atomic bid β is a tuple $\langle g_i^C, \mathcal{T}, \mathcal{C}, \mathcal{D} \rangle$ where $g_i^C \in \mathcal{G}$, $\mathcal{T}$ is the time to download a

VNF to be deployed, $\mathcal{C}$ is the computational and network cost to achieve the goal, and $\mathcal{D}$ is a metric capturing the network decentralisation if g_i^C is achieved by the agent that placed the bid. The agent offering the best combination of $\mathcal{T}$, $\mathcal{C}$, and $\mathcal{D}$ considering the auctioneer's preferences wins the auction and is assigned to achieve g_i^C.

Considering the NFV-A protocol, we next describe how agents place bids and how the auctioneer selects the winner.

C. Bidding Heuristic

When an agent receives a CFP, it first evaluates if it has a plan (executed by a VNF) that can achieve the auctioned goal. If multiple plans can achieve the goal, the agent uses a plan selection function, *e.g.*, [26] (out of the scope of this paper). A simple approach is to prioritise plans that demand less time to execute the VNF, as implemented in our evaluation. Then, the bidder performs the following tasks: (i) evaluation if the NFVI-PoP meets computational requirements of the VNF that will execute the plan; and (ii) calculation of each bid component. Network hard constraints—in the VNF requirements and specified in the goal constraints—are evaluated while calculating the computational and network cost. If any hard constraint is unsatisfied, the agent informs the auctioneer that it refuses to participate in the auction. Otherwise, it sends its bid. Currently, we assume that agents belong to the same provider. Therefore, there is no need for incentivization for them to participate in auctions because they always evaluate CFPs and send a bid if constraints are satisfied. If there are other candidate plans available in case of failure in meeting VNF requirements and goal constraints, these other plans are evaluated before sending a refusal to the auctioneer. We describe the tasks of calculating a bid as follows.

1) Evaluation of Computational Requirements: Computational requirements are related to the instantiation of a VNF. In order to instantiate a VNF v in a NFVI-PoP p controlled by a bidder agent, p must have available computational resources that meet v's requirements. Formally, v's requirements are satisfied when $sat_{req}(p, v)$ holds, as shown below.

$$sat_{req}(p, v) := \forall r_C \in R_C, req_v(r_C) \leq free_p(r_C) \quad (1)$$

2) Bid Components: As introduced earlier, each bid has three components $\mathcal{T}$, $\mathcal{C}$, and $\mathcal{D}$, which correspond to the VNF download time, the computational and network cost, and network decentralisation, respectively. We next detail how each component is calculated.

a) VNF Download Time: In our solution, we currently assume that a VNF associated with an agent cannot be responsible for achieving multiple goals in parallel. Therefore, if a VNF is executing, another VNF must be deployed. However, after achieving a delegated goal, the VNF becomes idle and, consequently, becomes eligible to achieve another goal. VNFs are made available to agents through the VNF repository, a server that provides the service of making VNFs available for download, which is known to all agents.

The VNF download time is: (i) 0, if the VNF is already deployed and does not need to be downloaded; or (ii) the estimated time to download, from the VNF repository, the VNF that can execute the plan selected to achieve the auctioned goal. The estimated time considers the path with the highest available throughput to download the VNF in the shortest amount of time and the size of the VNF to be downloaded.

b) Computational and Network Cost: The computational and network cost is our main bidding component. It assesses the amount of computational and network resources employed to instantiate a VNF and achieve a goal as well as evaluates if the instantiation of a VNF in a particular NFVI-PoP would leave resources unusable, thus preventing the instantiation of other VNFs. Moreover, while calculating this bidding component we: (i) verify whether goal constraints are met; and (ii) choose the path between the auctioneer and bidder agents to be used, if the bidder wins the auction (chaining problem).

The main idea is to find the path with the lowest cost between the auctioneer and the bidder. In order to do so, we represent the network as a graph, in which nodes are connected by links with weights capturing the amount of resources being consumed due to the use of this link. The identification of the path with the lowest cost is done using a customised version of the widely known Dijkstra's algorithm, which finds the shortest path in a graph in polynomial time. The key issue is thus to specify meaningful weights to the network links.

In Algorithm 1, we show the Dijkstra's algorithm, customised to calculate accumulated network measurements to verify goal constraints and the link weights. Our customisations correspond to the highlighted lines in the algorithm. It receives as input a graph representing the network, a source (bidder) and a destination (auctioneer) nodes, a constrained goal and the VNF to be deployed.

We first detail how we evaluate whether goal constraints are met. In the variables initialised in lines 4–6, we store accumulated latency, packet loss and hops in paths. Therefore, when evaluating the use of a particular link in a path, we update these variables. This is done in lines 21–23 in Algorithm 1. Then, in line 24, we check if goal constraints, associated with network resources (bandwidth) and measurements (latency, packet loss and hops), are satisfied. This is done using the $sat_{cst}(var, value, C)$ function, shown below.

$$\begin{aligned} sat_{cst}(var, value, C) :=& \forall c \in C | var = c[n], \\ & c[op](value, c[val]) \end{aligned} \quad (2)$$

where var is the network resource or measurement to be evaluated, $value$ is its current value and C is the set of goal constraints. This given value is then checked against the value informed in goal constraints using the specified comparison operator. If any constraint is unsatisfied, the link is discarded.

We now focus on the link weight calculation. In order to evaluate consumed resources, we consider both the link and its source node and assess the utility of computational resources that remain left by going throught a node (NFVI-PoP) of the network. Typically, to instantiate a VNF in the future we need to use part of all computational resources. Consequently, if there is only a subset of resources available, the instantiation of new VNFs may be problematic. Our heuristic thus aims to incentivise the balanced use of resources. This balance in

Algorithm 1 $lowestCostPath(G, s, d, g_i^C, vnf)$

Require: Graph G
Require: Source node s
Require: Destination node d
Require: Constrained goal g_i^C
Require: VNF to be deployed vnf
1: **for all** $v \in V[G]$ **do**
2: $dist[v] \leftarrow +\infty$
3: $prev[v] \leftarrow 0$
4: $latency[v] \leftarrow 0$
5: $pktLoss[v] \leftarrow 0$
6: $hops[v] \leftarrow 0$
7: **end for**
8: $dist[s] \leftarrow 0$
9: $Q \leftarrow V[G]$
10: **while** Q is not an empty set **do**
11: $u \leftarrow \text{Extract}_{\text{Min}}(Q)$
12: **if** u is d **then**
13: $S \leftarrow$ empty set
14: **while** $prev[u]$ is not undefined **do**
15: $S \leftarrow S \cup \{(prev[u], u)\}$
16: $u \leftarrow prev[u]$
17: **end while**
18: **return** S
19: **end if**
20: **for all** edge (u, v) outgoing from u **do**
21: $latency[v] \leftarrow value_u(v, L) + latency[u]$
22: $pktLoss[v] \leftarrow pktLoss[u] + ((1 - pktLoss[u]) \times value_u(v, P))$
23: $hops[v] \leftarrow 1 + hops[u]$
24: **if** $sat_{cst}(L, latency[v], C) \wedge sat_{cst}(P, pktLoss[v], C) \wedge sat_{cst}(H, hops[v], C) \wedge sat_{cst}(B, value_u(v, B), C)$ **then**
25: $rscEval_u \leftarrow rscEval_{hop}(u)$
26: **if** $u = s$ **then**
27: $rscEval_u \leftarrow rscEval(u, vnf)$
28: **end if**
29: $weight \leftarrow value_u(v, B) \times (rscEval_u + rscEval_{hop}(v))$
30: **if** $dist[u] + weight < dist[v]$ **then**
31: $dist[v] \leftarrow dist[u] + weight$
32: $prev[v] \leftarrow u$
33: **end if**
34: **end if**
35: **end for**
36: **end while**
37: **return** undefined

the use of resources is captured by calculating the harmonic mean of the ratio of available computational resources of an NFVI-PoP. This is calculated by the functions $freeRscUtil(p)$ and $consRscUtil(p, v)$, shown in Table II, which calculate the utility of an NFVI-PoP's free resources—current resources and resources consumed by a VNF instantiation, respectively.

These introduced functions are used to evaluate a link weight in the following way. We first assess the cost of using the source and target nodes of the link. If the node is a hop, no computational resource is used, only network resources (links). Hence, if VNFs can be instantiated in the node due to available resources, the cost is higher because links surrounding this node should preferably remain less used. The cost is then given by $freeRscUtil(p)$. If no VNF can be instantiated (*i.e.*, the node is full), the cost is 0, because the link can be used without compromising the instantiation of VNFs in the future in that node. This is formalised in the equation below, which uses the $full(p)$ logic predicate introduced in Table II.

$$rscEval_{hop}(p) = \begin{cases} 0 & \text{, if } full(p) \\ freeRscUtil(p) & \text{, otherwise} \end{cases} \quad (3)$$

The evaluation of the source node (bidder) is different, because it requires consuming resources for instantiating the VNF. The idea is also to consider the instantiation of VNFs in the future. If, after instantiating the VNF, no other VNF can be instantiated in that node, the consumed resources are not only those consumed by the VNF but all resources ($freeRscUtil(p)$), as those remaining become useless. In the opposite case, the cost is lower, corresponding only to the consumed resources. This is also the case if, although no new VNFs can be deployed, there is no alternative VNF that can be instantiated in that node allowing the instantiation of other new VNFs, meaning that only one VNF can be deployed in the node anyway. This is formalised as follows, using the $canDeploy(p, v)$ predicate, also defined in Table II.

$$rscEval(p, v) = \begin{cases} consRscUtil(p, v) & \text{, if } canDeploy(p, v) \vee \\ & \neg\exists v' \, canDeploy(p, v') \\ freeRscUtil(p) & \text{, otherwise} \end{cases} \quad (4)$$

Finally, the link weight is its bandwidth (the higher the bandwidth, the higher the cost) multiplied by the sum of the cost evaluation of the nodes connected to the link, as shown in line 29 of Algorithm 1. Lines 25–28 check if the link source is the path source to treat it in a distinguished way. This completes the explanation regarding this bidding component and we then proceed to the last component.

c) Decentralisation: The decentralisation component corresponds to how much physically distant the auctioneer and the bidder are. Given that our input information does not capture geographic information about NFVI-PoPs, we use latency to estimate decentralisation, assuming that geographically distant NFVI-PoPs tend to have higher latency than NFVI-PoPs that are geographically closer. This component is thus the accumulated latency of the path that links the auctioneer and the bidder, obtained from the algorithm presented above.

D. Bid Evaluation

To choose a winner, auctioneers receive from each bidder the three bid components introduced in Section III-B. The values of these components are used to calculate a utility value considering the auctioneer's preferences. A preference pf is a real number in $[0, 1]$ corresponding to the importance of a bid component to achieve a particular constrained goal, where 0 is the lowest importance and 1 the highest. Moreover, $pf_{\mathcal{T}} + pf_{\mathcal{C}} + pf_{\mathcal{D}} = 1$. Preferences can be specified globally or tailored to specific goals.

Example 2: Assume that an auctioneer has preferences 0.5, 0, and 0.5 associated with $\mathcal{T}$, $\mathcal{C}$ and $\mathcal{D}$, respectively, to achieve the $isTrafficMalicious$ goal. This means that the amount of the network infrastructure usage is irrelevant (as long as it meets the computational and network constraints), and the time to start analysing suspicious traffic and the decentralisation of VNFs are equally important to achieve this goal.

Preferences are used as weights for each bid component, when they are combined in a weighted sum. Given that each bid component is in a different unit, we normalise them to a

TABLE II
DEFINITION OF MATHEMATICAL FUNCTIONS AND LOGIC PREDICATES.

Function/Predicate	Mathematical/Logic Expression	Description
$freeRscUtil(p)$	$\frac{\lvert R_C \rvert}{\sum_{r_C \in R_C} \left(\frac{free_p(r_C)}{total_p(r_C)} \right)^{-1}}$	Harmonic mean of the ratio of available computational resources of an NFVI-PoP p.
$consRscUtil(p, v)$	$\frac{\lvert R_C \rvert}{\sum_{r_C \in R_C} \left(\frac{req_v(r_C)}{total_p(r_C)} \right)^{-1}}$	Harmonic mean of the ratio of the computational resources required by VNF v from an NFVI-PoP p.
$full(p)$	$\forall v \exists r_C \in R_C, req_v(r_C) > free_p(r_C)$	$full(p)$ is true when there is no VNF that can be deployed in the NFVI-PoP p, that is, there is at least one VNF requirement unsatisfied, for all VNFs.
$canDeploy(p, v)$	$\exists v' \forall r_C \in R_C, req_{v'}(r_C) \leq free_p(r_C) - req_v(r_C)$	$canDeploy(p, v)$ is true when there is a VNF that can be deployed in an NFVI-PoP p after the deployment of the VNF v in p.

value in $[0, 1]$ considering the minimum and maximum values of received bids, for each component—note that the auctioneer receives bids only from agents that meet all hard constraints. Normalised values are denoted by $\hat{\beta_{\mathcal{T}}}$, $\hat{\beta_{\mathcal{C}}}$, and $\hat{\beta_{\mathcal{D}}}$. As result, the agent that placed the bid associated with the highest utility is the winner, as shown in the equation below.

$$\underset{\beta \in Bid}{\operatorname{argmin}} \quad pf_{\mathcal{T}} \times \hat{\beta_{\mathcal{T}}} + pf_{\mathcal{C}} \times \hat{\beta_{\mathcal{C}}} + pf_{\mathcal{D}} \times (1 - \hat{\beta_{\mathcal{D}}}) \qquad (5)$$

In this equation, we use the complement of $\hat{\beta_{\mathcal{D}}}$ because, as opposed to other components, the higher its value, the better.

IV. IMPLEMENTATION AND EVALUATION

In this section, we detail a DDoS case study to illustrate our auction-based approach for NFV orchestration.

A. Case Study

We show our solution for NFV orchestration in action on a DDoS attack scenario. This type of attack can compromise the availability of network services, putting the resilience of the network at risk. Malicious activities such as DDoS attacks can disrupt network operation by flooding the network with spurious requests. This attack is a common issue that can be solved in many different ways.

Our VNF repository has five VNFs that allow agents to tackle DDoS attacks: (i) *Link Monitor*, which monitors links and identifies spikes in traffic; (ii) *Rate Limiter*, which throttles the traffic of interfaces or flows by destination and/or source addresses; (iii) *Anomaly Detector*, which detects an attack and determines the victim address; (iv) *Classifier*, which records and classifies traffic flows to find out the source address of attackers; and (v) *Load Balancer*, which enables agents to split traffic into flows small enough to be handled by more VNFs, each consuming fewer resources.

Each VNF can accomplish tasks corresponding to plans that can achieve goals. Examples of goals are *linkRateLimited(link)* or *?anomalousUsage(link)*, which aim to achieve states in which a link has its traffic reduced and there is knowledge regarding the usage of a link (whether it is anomalous). The expected VNF orchestration in this scenario is that a link monitor (manually deployed, as part of the initial state of the network) detects a spike in traffic in one of the links, causing a rate limiter and anomaly detector VNFs to be deployed to limit the link traffic and analyse the traffic, respectively. Once the target of the attack is identified, only the traffic destined to the attack victim is reduced and a classifier VNF is instantiated to find out attackers. Lastly, the rate limiter VNF will block only the connections from attackers. The complete BDI agent model of this scenario can be seen elsewhere [27].

In our extended scenario, agents must now dynamically instantiate VNFs to achieve goals. Therefore, when goals must be achieved, auctions take place. Moreover, we added the *Load Balancer* VNF, which is a VNF associated with a plan that can achieve many goals by splitting the traffic and generating sub-goals with less strict constraints.

B. Prototype Implementation

To evaluate our approach, we developed an NFV testbed that integrates `BDI4JADE` [17], a BDI agent platform, and `containernet` [28], a fork of `Mininet` [16] that implements `Mininet` hosts as Docker containers [29]. We selected `containernet` because it provides operational system-level virtualisation and allows to fine tune the maximum resource usage of hosts (*e.g.*, memory, network and CPU). Differently from most BDI platforms, `BDI4JADE` is a multi-agent platform that implements agents in pure Java, which facilitates the integration with other applications. In our system, each VNF deployed to the virtualised network is represented by a host in `Mininet` (*i.e.*, a Docker container). Furthermore, the agent-to-VNF communication is implemented using `RESTful` [30] calls. We provide VNFs with a light Python script that sends and receives `RESTful` calls, allowing agents to easily obtain VNF-specific information. Our prototype integrating `BDI4JADE` and `containernet` is publicly available.[1]

At the beginning of the simulation, a configuration file specifying the network topology (resource specification of NFVI-PoPs as well as their links and VNF repository location) is read by the `BDI4JADE` platform, which launches the BDI agents and a Python application responsible for starting `Mininet`. They exchange information through `RESTful`

[1] https://sourceforge.net/projects/nfvauction/

TABLE III
VNF REQUIREMENTS AND SUPPORTED WORKLOADS.

VNF	CPU	Memory	Disk	Bandwidth
Link Monitor	1 core	1 GB	2 GB	0 MB/s
Rate Limiter	1 core	1 GB	2 GB	0 MB/s
Anomaly Detector	1 core	2 GB	10 GB	2 MB/s
Classifier	2 core	4 GB	10 GB	1 MB/s
Load Balancer	1 core	1 GB	2 GB	0 MB/s

calls and to coordinate network configuration and agent-to-PoP assignment. As result, agents are in control of NFVI-PoPs and pre-configured VNFs, which are those present in the initial state of the network, are deployed to specified agents according to a topology configuration file.

C. Simulation Setup

We created a server that answers HTTP requests through a Python's `SimpleHTTPServer` module [31] v. 2.7.12 and provides video streamming through VLC [32] v. 2.2.2.

Our VNFs were implemented as follows: (i) *Link Monitor* uses `ifstat` to monitor incoming and outgoing traffic; (ii) *Rate Limiter* communicates with the `OpenVSwitch` [33] controller v. 2.5.0 to apply and remove queue-based rate limiting rules to network interfaces and specific source/destination IPs; (iii) *Anomaly Detector* and (iv) *Classifier* use `Snort` [34] v. 2.9.2.2 to detect suspicious traffic; and (v) *Load Balancer* also communicates with `OpenVSwitch` to set load balancing rules. The `OpenVSwitch` switches are connected to a `POX` [35] controller.

The requirements of each VNF used in our simulation are listed in Table III. All VNFs have a download size of 500 MB, which is the amount of data transferred from the VNF repository to instantiate them. Agent preferences are tailored to goals. For example, the $linkRateLimited(link)$ goal (whose auction is described in detail in the next section) is associated with the following preferences: $pf_{\mathcal{T}} = 0.8$, $pf_{\mathcal{N}} = 0.1$, and $pf_{\mathcal{D}} = 0.1$. As goal constraints, we use maximum packet loss of 8% for all goals, and a varying latency constraint—for the $linkRateLimited(link)$ goal, the maximum latency is 50 ms.

Our network topology, which is composed of six NFVI-PoPs, is illustrated in Figure 3. In this topology, the VNF repository is connected to NFVI-PoP 5. The HTTP and video streaming server is directly connected to NFVI-PoP 6, and a *Link Monitor* VNF is also installed on this NFVI-PoP, monitoring the link that connects the switch to the HTTP and video streaming server.

D. Evaluation Results

By simulating the scenario described above, we obtained the configuration shown in Figure 3. The squares in gray show where VNFs have been allocated (selection and placement) and dotted lines represent the chaining of VNFs throughout the adopted DDoS attack mitigation strategy. They are numbered according to the time order that auctions occurred.

The background traffic of our simulation consisted of 50 hosts requesting to watch the video streaming in a time frame,

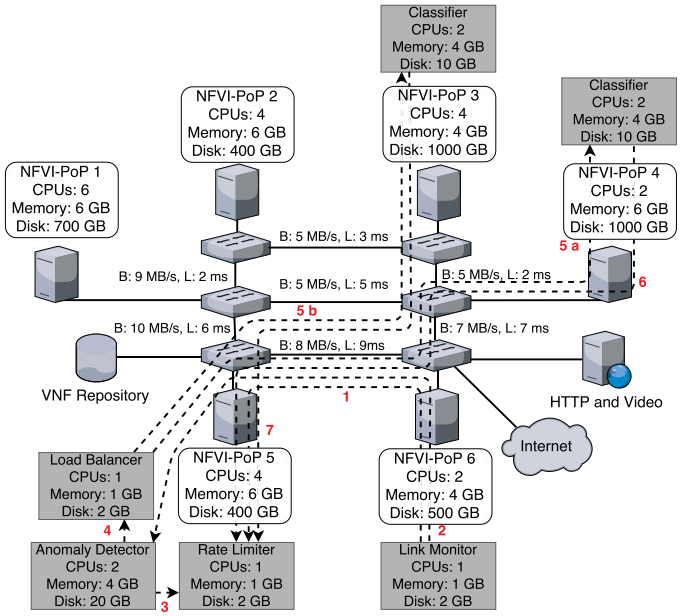

Fig. 3. Emergent selection and resource allocation.

TABLE IV
COSTS OF ACHIEVING GOAL *linkRateLimited(link)* AND BID UTILITY.

Agent	$\mathcal{N}$	$\mathcal{T}$	$\mathcal{D}$	Bid Utility
1	2.69	50.00	19.45	0.47
2	4.70	55.55	23.52	0.54
3	2.69	100.00	16.72	0.88
4	0.69	100.00	11.34	0.86
5	0.70	0.00	11.85	0.06
6	0.00	62.50	0.00	0.60

then they stop and make HTTP requests. As in [36], for each host requesting video streaming traffic, there are other five hosts requesting HTTP traffic. We used `scapy` [37] to generate realistic background as well as anomalous traffic. The attack consisted of 8 malicious hosts performing a SYN flood attack, in which packets targeted the two ports of the service server (8000 for HTTP and 8080 for video stream).

Figure 4 shows the amount of traffic from the switch connected to NFVI-PoP 6 to the HTTP and video server throughout the simulation. After a few seconds of simulation, the attack started and the outgoing traffic to the server reached 6 MB/s. Immediately, the *Link Monitor* detected the peak, triggering the auctions to instantiate VNFs. As result, the *Link Monitor* triggered the auction to achieve a *linkRateLimited(link)* goal, and agents selected the *Rate Limiter* VNF to achieve this goal. Next, agents executed our bidding heuristic to place bids and the auctioneer evaluated bids using the preferences detailed in the previous section. Table IV shows the values of each bid component from agents, along with the final bid utility evaluated by the auctioneer. Because the lowest bid was given by Agent 5, this agent was the winner and deployed the VNF to achieve the auctioned goal.

Due to space restrictions, we briefly detail results of other

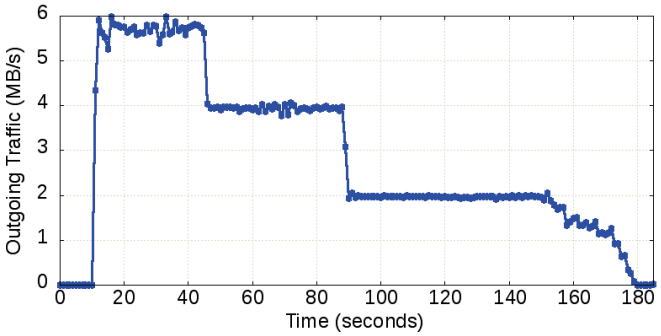

Fig. 4. Traffic from the switch towards the HTTP and video server.

auctions. A *flowRecord(IP)* goal was auctioned to deal with 2 MB/s of traffic (the *Rate Limiter* reduced the link to 4 MB/s and furthermore to 2 after the *Anomaly Detector* identified the target IP), requiring a *Classifier* with bandwidth of 10 MB/s. As it is not possible to achieve *Classifier*'s plans due to the goal's bandwidth constraint, the *Load Balancer*'s plan to split the traffic and generate sub-goals with less restricted constraints was selected by the agent. The traffic was then separated into two flows, resulting in two auctions for *flowRecord(IP)*, each with 1 MB/s, which led to the allocation of two *Classifiers* in the network. The resulting traffic reduction after the successive deployment of VNFs and the successful mitigation of the attack can be seen in Figure 4.

V. Related Work

The idea of using auctions in the context of NFVs was already proposed by Gu *et al.* [24]. However, they did not address the placement and chaining problems. Instead, they used auctions as a mechanism for service providers to sell network resources for users. The placement and chaining problems have been tackled in alternative solutions, which are briefly described as follows.

Kuroki *et al.* [38] implemented an orchestrator that takes as input a service description and an optimisation policy for each incoming network service request. Each request is submitted to an allocation algorithm based on its optimisation policy. Differently, instead of receiving as input the service requests, we solve the selection problem by embedding agents with knowledge that enables them to decide which VNFs are needed and build the VNF-FG dynamically.

Shen *et al.* [39] use external network controllers to handle low-level operations in a modular orchestrator, in which modules can be easily replaced by third-party implementations. They use a multi-objective resource scheduling algorithm, which is based on genetic algorithms and can handle multi-objective functions to allocate service chains. They provide a graphical interface that allows the creation of service chains and the monitoring of resources. Although we do not provide a graphical interface to monitor resources, our solution ensures that the deployment of service chains is fully automated and does not depend on the operator input.

Yoshida *et al.* [6] proposed a multi-objective resource scheduling algorithm that aims to optimize NFV infrastructure resources. They provide a genetic algorithm that finds pareto-optimal solutions to the placement problem considering conflicting objectives. Similarly, preferences are given to their system *a priori* and used to determine the most suitable solutions. Although their system deals with requests dynamically, it only solves the placement problem (statically) while we also deal with selection and chaining.

A formalisation to the placement and chaining problems was made by Luizelli *et al.* [21], who proposed both an Integer Linear Programming (ILP) model and a heuristic procedure. Similarly to previously discussed related work, they take as input the VNFs, their connections and know *a priori* the amount of work the system will have to handle. Distinctively, our auction system relies on the cognitive capacity of agents to dynamically decide the placement and chaining of VNFs based on the current state of the network. In addition, it solves the selection problem while in [21] the solution to the selection problem is taken as input.

VI. Conclusion

The NFV architecture provides an alternative to the cumbersome deployment, configuration and management of middleboxes, relieving network operators from vendor-specific technologies. This architecture leads to the challenges of selection, placement and chaining of VNFs, requiring solutions that solve these problems, ideally, at runtime and without human interference.

In this paper, we proposed an auction-based mechanism, in which agents bid on the allocation of VNFs. Cognitive agents are embedded with preferences from a domain specialist, allowing them to decide what action should be taken based on perceptions from the environment. While placing a bid, with our bidding heuristic, agents select a VNF to achieve a given goal and identify the path with the lowest cost to chain VNFs. The auction winner determines where to place the VNF. We evaluated our proposal in a testbed that integrated `BDI4JADE` and `Mininet`, using Docker containers to represent five distinct VNFs that work together to mitigate a DDoS attack. Results show that the emergent behaviour achieved by our agents can successfully mitigate the attack by selecting and instantiating the required VNFs on demand and autonomously. As future work, we will further explore the emerging behaviour of agents by implementing new VNFs with distinct abilities and evaluate them in different scenarios, analysing the optimality of the solution devised by agents. Moreover, we will address scenarios involving multiple providers, thus requiring the use of incentivization and reputation models.

Acknowledgment

The authors of this work would like to thank the financial support of CNPq for research grants ref. 442582/2014-5, 303232/2015-3, 311088/2015-5 and 407899/2016-2.

REFERENCES

[1] R. Guerzoni *et al.*, "Network functions virtualisation: an introduction, benefits, enablers, challenges and call for action, introductory white paper," in *SDN and OpenFlow World Congress*, 2012.

[2] G. ETSI, "Network functions virtualisation (nfv): Architectural framework," *ETsI Gs NFV*, vol. 2, no. 2, p. V1, 2013.

[3] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc)," *IEEE Network*, vol. 28, no. 6, pp. 18–26, 2014.

[4] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[5] G. ETSI, "Network functions virtualisation (nfv); management and orchestration," *ETSI*, 2014. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf

[6] M. Yoshida, W. Shen, T. Kawabata, K. Minato, and W. Imajuku, "Morsa: A multi-objective resource scheduling algorithm for nfv infrastructure," in *Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific*. IEEE, 2014, pp. 1–6.

[7] W. Rankothge, J. Ma, F. Le, A. Russo, and J. Lobo, "Towards making network function virtualization a cloud computing service," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 89–97.

[8] N. Bouten, J. Famaey, R. Mijumbi, B. Naudts, J. Serrat, S. Latré, and F. De Turck, "Towards nfv-based multimedia delivery," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 738–741.

[9] L. Qu, C. Assi, and K. Shaban, "Network function virtualization scheduling with transmission delay optimization," in *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*. IEEE, 2016, pp. 638–644.

[10] M. Bouet, J. Leguay, T. Combe, and V. Conan, "Cost-based placement of vdpi functions in nfv infrastructures," *International Journal of Network Management*, vol. 25, no. 6, pp. 490–506, 2015.

[11] R. Riggio, T. Rasheed, and R. Narayanan, "Virtual network functions orchestration in enterprise wlans," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 1220–1225.

[12] T. Wen, H. Yu, G. Sun, and L. Liu, "Network function consolidation in service function chaining orchestration," in *Communications (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[13] M.-T. Thai, Y.-D. Lin, and Y.-C. Lai, "A joint network and server load balancing algorithm for chaining virtualized network functions," in *Communications (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[14] A. S. Rao, M. P. Georgeff *et al.*, "Bdi agents: From theory to practice." in *ICMAS*, vol. 95, 1995, pp. 312–319.

[15] F. Schardong, I. Nunes, and A. Schaeffer-Filho, "A distributed nfv orchestrator based on bdi reasoning," in *Integrated Network Management (IM), 2017 IFIP/IEEE International Symposium on*. IEEE, 2017, pp. 107–115.

[16] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

[17] I. Nunes, C. Lucena, and M. Luck, "Bdi4jade: a bdi layer on top of jade," in *Proc. of the Workshop on Programming Multiagent Systems*, 2011, pp. 88–103.

[18] A. M. Medhat, G. Carella, C. Lück, M.-I. Corici, and T. Magedanz, "Near optimal service function path instantiation in a multi-datacenter environment," in *Network and Service Management (CNSM), 2015 11th International Conference on*. IEEE, 2015, pp. 336–341.

[19] H. Moens and F. De Turck, "Vnf-p: A model for efficient placement of virtualized network functions," in *Network and Service Management (CNSM), 2014 10th International Conference on*. IEEE, 2014, pp. 418–423.

[20] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*. IEEE, 2015, pp. 171–177.

[21] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 98–106.

[22] J. G. Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.

[23] Q. Sun, P. Lu, W. Lu, and Z. Zhu, "Forecast-assisted nfv service chain deployment based on affiliation-aware vnf placement," in *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE, 2016, pp. 1–6.

[24] S. Gu, Z. Li, C. Wu, and C. Huang, "An efficient auction mechanism for service chains in the nfv market," in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 2016, pp. 1–9.

[25] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *The knowledge engineering review*, vol. 10, no. 02, pp. 115–152, 1995.

[26] J. Faccin and I. Nunes, "Bdi-agent plan selection based on prediction of plan outcomes," in *Proceedings of the 2015 IEEE / WIC / ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT) - Volume 01*, ser. WI-IAT '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 166–173. [Online]. Available: http://dx.doi.org/10.1109/WI-IAT.2015.58

[27] I. Nunes, F. Schardong, and A. Schaeffer-Filho, "Bdi2dos: An application using collaborating bdi agents to combat ddos attacks," *J. Netw. Comput. Appl.*, vol. 84, no. C, pp. 14–24, Apr. 2017. [Online]. Available: https://doi.org/10.1016/j.jnca.2017.01.035

[28] "Github - containernet/containernet: Mininet fork that adds container (e.g. docker) support," https://github.com/containernet/containernet, accessed: 2017-09-29.

[29] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[30] L. Richardson and S. Ruby, *RESTful web services*. " O'Reilly Media, Inc.", 2008.

[31] "20.19. simplehttpserver — simple http request handler — python 2.7.14 documentation," view-source:https://docs.python.org/2/library/simplehttpserver.html, accessed: 2017-09-29.

[32] C. Müller and C. Timmerer, "A vlc media player plugin enabling dynamic adaptive streaming over http," in *Proceedings of the 19th ACM international conference on Multimedia*. ACM, 2011, pp. 723–726.

[33] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending networking into the virtualization layer." in *Hotnets*, 2009.

[34] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks." in *LISA*, vol. 99, no. 1, 1999, pp. 229–238.

[35] "Github - noxrepo/pox: The pox controller," https://github.com/noxrepo/pox, accessed: 2017-09-29.

[36] A. S. da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, "Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn," in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2016, pp. 27–35.

[37] "Scapy," http://www.secdev.org/projects/scapy/, accessed: 2017-09-29.

[38] K. Kuroki, M. Fukushima, and M. Hayashi, "Framework of network service orchestrator for responsive service lifecycle management," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 960–965.

[39] W. Shen, M. Yoshida, K. Minato, and W. Imajuku, "vconductor: An enabler for achieving virtual network integration as a service," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 116–124, 2015.