A Distributed NFV Orchestrator based on BDI Reasoning

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

***Abstract*—Network function virtualisation (NFV) decouples network functions from physical devices, simplifying the deployment of new services. As opposed to traditional middleboxes, VNFs can be dynamically deployed and reconfigured on demand, posing strict management challenges to networked systems. Selecting VNFs from a repository, defining where they will be placed in the virtualised network as well as chaining them to achieve the desired behaviour are problems that have to be tackled by an orchestrator. In this paper, we propose a distributed approach to NFV orchestration using belief-desire-intention (BDI) reasoning, addressing the selection, placement and chaining problems through the interaction among autonomous software agents, which collectively work in a distributed and decentralised manner. Agents are capable of bidding on the allocation of resources for new VNFs, as well as managing the chaining of VNFs. Further, we validate our theoretical model through a DDoS attack case study, in which we analyse the emergent behaviour of the autonomous agents.**

I. Introduction

Over the years, complexity and size of networks have drastically increased and so did the requirement for more flexible management. Physical network devices (also known as middleboxes) are proprietary highly-specialised products that require specific chaining, physical installation, and their functionality cannot be easily changed. Moreover, the increasing demand for more diverse and short-lived networks to handle high data rates, such as in infrastructure as a service (IaaS) and network as a service (NaaS) [1], requires network operators to deploy rapidly and operate complex network equipments, leading to high CAPEX and OPEX [2].

To address these issues, network function virtualisation (NFV) has been proposed as a way to decouple network services from physical devices through virtualisation [3], [4]. There is a broad set of services that have been traditionally performed by middleboxes—*e.g.*, firewalls, intrusion detection systems (IDS), intrusion prevention systems (IPS), traffic shaping, network address translation (NAT), traffic accelerators, caches and proxies—that can be virtualised into cheap and easily deployable virtual machines [5]. The evolution of networks into software-based functions and services are concrete steps towards future networks.

However, virtual network functions (VNFs) need to be managed and aggregated in meaningful ways such that the desired functionalities are achieved, in what is called NFV orchestration. NFV orchestration can be decomposed into three core problems: (i) automatic *selection* of VNFs; (ii) VNF *placement* in the virtualised network; and (iii) *chaining* of VNFs into service chains [1]. Typically, humans design the network forwarding graph [6] of service chains, that is, decide how VNFs are chained. However, as network complexity increases and service-level agreement (SLA) requirements over on-demand networks become more strict, guaranteeing the orchestration of virtual nodes in real-time becomes vital for carriers and service providers. Relying solely on humans to enforce SLAs is often impractical and, consequently, automating NFV orchestrators is crucial and can relieve network operators from the burden of manually ensuring SLA reinforcement. Nonetheless, NFV orchestration schemes proposed so far do not explore the benefits of decentralised autonomous components [7], [8], [9], [10], [11].

In this paper, we propose an approach to the orchestration problem by extending the NFV architecture of ETSI [6] with autonomous and distributed software components, referred to as *agents*, which are provided with the *belief-desire-intention* (BDI) reasoning [12]. We represent each VNF as a BDI agent, each with its capabilities and beliefs about the network. Such agents are capable of bidding on the allocation of resources for new VNFs either requested by the network manager or by other agents (selection of VNFs), as well as managing the forwarding graph of the NFV (chaining). Although the reasoning approach described in this paper focuses on NFV selection and chaining, we also present a preliminary solution for placement based on resource allocation. Our approach is thus *decentralised*, given that agents make individual autonomous decisions and, collectively, orchestrate VNFs. To evaluate our approach for NFV orchestration, we consider a DDoS attack scenario, in which we analyse the emergent behaviour of the autonomous agents. However, we emphasise that our distributed reasoning approach can be more generally applicable to other scenarios.

Section II details the BDI reasoning cycle and its components. Our NFV orchestration approach is described in Section III. Section IV details a prototype implementation and experimental results demonstrating NFV orchestration for the detection and mitigation of a DDoS attack. Related work is discussed in Section V, and Section VI concludes this paper.

II. BDI Reasoning Cycle and Architecture

Believe-desire-intention (BDI) is an architecture that includes a reasoning cycle, which provides agents with rational

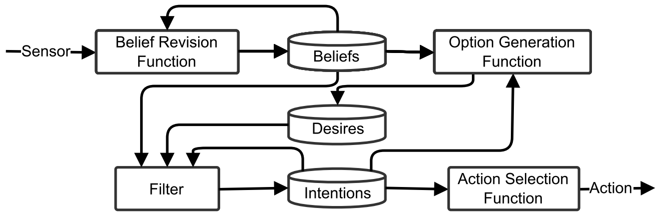


Fig. 1. Overview of the BDI architecture [13].

```
initialize-state();
repeat
  options := option-generator(event-queue);
  selected-options := deliberate(options);
  update-intentions(selected-options);
  execute();
  get-new-external-events();
  drop-successful-attitudes();
  drop-impossible-attitudes();
end repeat
```

Fig. 2. BDI reasoning cycle [12].

behaviour [12]. This architecture allows agents to deal with different scenarios in which flexible and intelligent behaviour is needed. In this section, we introduce the key concepts that comprise this architecture and briefly explain its limitations.

A. Overview

The BDI architecture has three main components. The first is *beliefs*, which represent the information that an agent has about itself and its environment. Beliefs do not necessarily correspond to an environment state that holds, because they capture an agent's world view, which is susceptible to noise. A set of beliefs, or belief base, can be implemented as a database, a set of logical expressions or other data structure. Beliefs are time-dependent, so they can be added, removed or changed over time. The second component corresponds to objectives to be achieved by an agent, named *desires* or *goals*, representing the motivational state of an agent. They can be explicitly added to an agent by the designer or generated by the agent through interactions with its surroundings or as result of agent actions. The deliberative state of an agent is called *intentions*, the third main BDI component. An intention is a goal that an agent is committed to achieve by the execution of a sequence of actions (or *plan*), in a structured manner intending to achieve one or more goals. That is, an agent is committed to achieve only what it believes that is feasible to be achieved.

These BDI components are connected through a reasoning cycle that is illustrated in Figure 1 [13]. A BDI agent, through sensors, perceives events from its environment, and updates its belief base using a *belief revision function*. Such beliefs, together with agent intentions, are used to update agent goals. This is performed by the *option generation function*. The selection of goals to be achieved, *i.e.*, those that will become intentions, is made by a *filtering function* (filter). Finally, an *action selection function* chooses an appropriate sequence of actions, or plan, to achieve intentions. This reasoning cycle is also detailed in the BDI interpreter (Figure 2), proposed by Rao and Georgeff [12]. It uses an alternative terminology and explicitly adds functions to drop achieved or impossible goals.

B. Limitations

The BDI architecture is a theoretical model associated with issues to be addressed when it is instantiated. Besides implementation issues, the BDI reasoning cycle includes many gaps to be fulfilled during the development of concrete BDI agents. An example of such a gap is the *action selection function* (often referred to as plan selection), which is responsible for selecting a suitable plan to achieve a goal. Often agents have multiple plans that are appropriate for achieving a goal; however, selecting one of them might be a challenging task due to unknown plan outcomes. Different approaches have tackled this problem from various perspectives. For example, Singh *et al.* [14] focused on learning when a plan might fail, while Faccin and Nunes [15] select the best plan in a given context considering agent preferences over secondary goals.

Another limitation of the original BDI architecture is the absence of abstractions to promote modular design and software reuse, which is important when considering the development of real world applications. The *capability* concept was introduced by Busetta *et al.* [16] as a way to build modular structures, promoting reuse of agent components. Capabilities include a subset of beliefs, goals and plans, which combined form an agent building block. Capabilities and other extensions and customisations of the BDI architecture make it suitable for modelling autonomous VNFs as BDI agents that together emerge into an NFV orchestrator capable of solving the selection, placement, and chaining problems.

III. NFV Orchestration based on BDI Reasoning

In this section, we detail our decentralised NFV orchestrator composed of autonomous BDI agents. It is summarised in Figure 3, showing our extension to the ETSI's NFV architecture [6], in which we replaced its centralised orchestrator and VNF manager, with BDI agents. Next, we first formalise the components of our orchestrator. Then, the solution to the placement, chaining, and selection problems is introduced based on the previous formalisation, using an auction process and internal agent reasoning. Finally, we present a case study based on a DDoS attack detection and mitigation, which illustrates how the theoretical model and the reasoning approach work.

A. Model Formalisation

To develop a decentralised orchestrator, we rely on the behaviour that emerges from the interaction among autonomous BDI agents. An agent is in charge of managing VNFs and is capable of monitoring the resources of physical machines and the network to make management decisions. Each agent is aware of the connections of its VNF instances as well as the memory and processor specifications. In order to modularise an agent and allow VNFs to be dynamically instantiated, we structure agents in terms of capabilities. Capabilities are reusable components that contain the elements required to

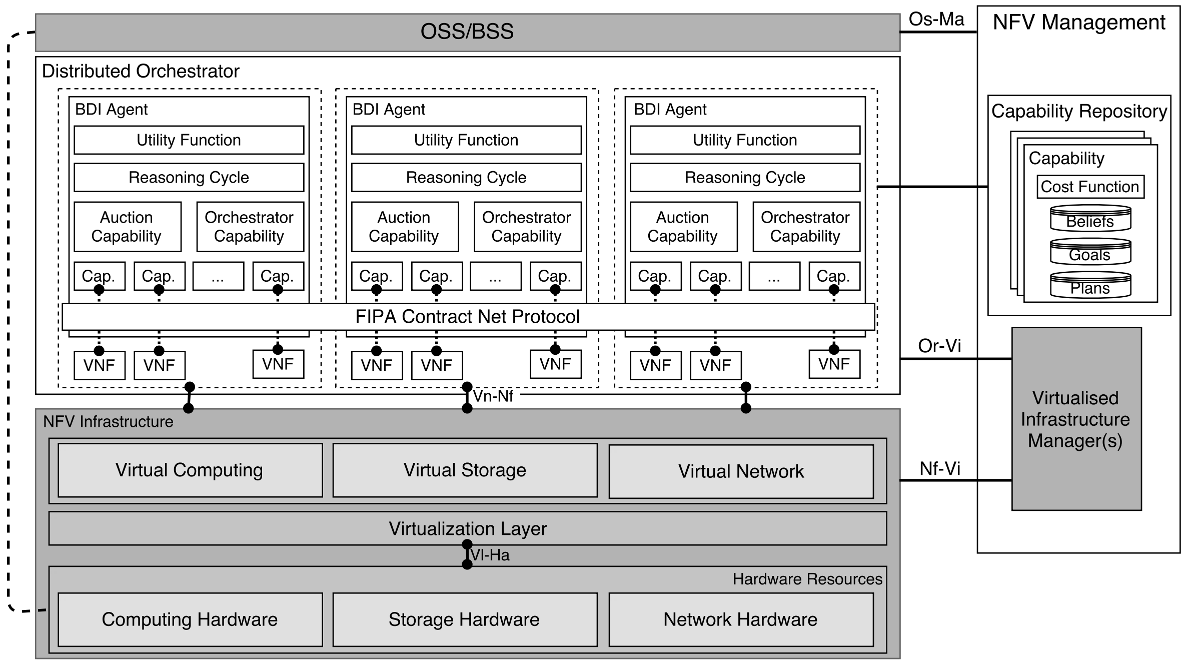


Fig. 3. Extended NFV Architecture: components in grey correspond to the ETSI's framework, while those in white are our NFV orchestrator.

coordinate VNFs. Therefore, each VNF is associated with a capability. As result, an agent is a component that aggregates a set of capabilities. Given this brief informal description of our model, we provide formal definitions as follows.

Definition 1 (Capability): A capability is a tuple $\langle B, G, P, \mathcal{B}, \mathcal{O}, \mathcal{F}, \mathcal{P}, \mathcal{C} \rangle$, where B is set beliefs, G is a set of goals, P is a set of plans, $\mathcal{B}$ is a belief revision function, $\mathcal{O}$ is an option generation function, $\mathcal{F}$ is a filtering function, $\mathcal{P}$ is a plan selection function, and $\mathcal{C}$ is a cost function.

Definition 2 (Agent): An agent is a tuple $\langle G, I, C, \mathcal{U} \rangle$, where G is a set of current agent goals, I is a set of plans that correspond to agent intentions, C is a set capabilities, and $\mathcal{U}$ is a utility function.

Definition 3 (Multiagent System): A multiagent system is a tuple $\langle A, Prot \rangle$, where A is a set of agents, and $Prot$ is a message protocol that each $a \in A$ is able to understand.

According to these definitions, it can be seen that capabilities specify parts of an agent. Each capability has a set of beliefs B, which collectively represent the agent belief base, *i.e.*, an agent's belief base is $\bigcup_{c \in C} B_c$. Goals of a capability consist of declarations of the goals that a capability can achieve (they represent the capability interface [17]), and plans consist of a sequence of actions to achieve goals. As said above, capabilities coordinate VNFs, and this is achieved by a set of five functions, which we detail below.

- $\mathcal{B} : \wp(E) \times \wp(B) \to \wp(B)$ — the belief revision function takes into account events E perceived by an agent, *e.g.*, messages received from other agents, and the current set of beliefs, and produces an updated set of beliefs.
- $\mathcal{O} : \wp(B) \times \wp(G) \to \wp(G)$ — the option generation function takes into account the current set of beliefs and goals, and produces an updated set of goals, *i.e.*, goals may be generated or dropped.
- $\mathcal{F} : \wp(G) \to \wp(G)$ — the filtering function takes into account a set of goals and produces a subset of goals, those that the agent will be committed to achieve.
- $\mathcal{P} : \wp(B) \times \wp(P) \to P$ — the plan selection function takes into account the current set of beliefs, *i.e.*, current context, and a set of plans that are candidates to achieve a goal, and selects one, among the candidates, to be executed.
- $\mathcal{C} : \wp(B) \to \mathcal{R}$ — the cost function is a function that takes into account the current context (beliefs) and produces a real number $[0, 1]$ that represents the cost required for an agent (or capability) to achieve a particular goal, *i.e.*, to perform a particular service. This function is further detailed in the next section.

Each capability is thus a part of an agent, and an agent has a set of capabilities. In addition, agents at runtime have a set of goals, which drives their behaviour. These are added as result of the functions described above or execution of plans. An agent may be committed to achieve not all of its goals, but a subset. Goals that an agent is committed to achieve are referred to as intentions, and the agent must have a plan to achieve them. Therefore, agent intentions are a set of plans, which are those that the agent is executing at the moment. In order to address the placement problem, agents are also provided with a utility function, detailed next.

- $\mathcal{U} : \wp(C) \to \mathcal{R}$ — the utility function is a function that takes into account the current set of capabilities (which are associated with VNFs) that are part of an agent and produces a real number $[0, 1]$ that represents the utility that having a capability active brings to an agent. This function is also further detailed in the next section.

Finally, the set of agents, with their capabilities that are responsible for coordinating all VNFs, comprise a multiagent system. This system is our decentralised NFV orchestrator. Given that all agents must communicate, message protocols are specified in this multiagent system, and all agents must be able to understand them.

B. BDI Agent Interaction

Given that our proposal consists of a decentralised orchestrator, in which independent and autonomous agents must collectively solve the selection, chaining and placement problems, we must provide means for agents to communicate and negotiate. Much research has been done to support agent communication [18], [19]. Therefore, we adopt an existing approach, namely the FIPA Agent Communication Language (ACL)[1], which specifies standards associated with message exchange. This specification is implemented in many of the existing agent and BDI platforms.

Agents negotiate in order to choose which services they are responsible for. Negotiation is performed by means of *auctions*. In our solution, we have two built-in capabilities, *auction* and *orchestrator*, which are added to every agent, allowing them to participate in auctions and decide whether to activate or deactivate other capabilities (*i.e.*, VNFs).

1) Auction Process: Auction algorithms are simple and well-known mechanisms used to reach agreements on the allocation of scarce resources, where "resource" is a general concept. In the orchestrator problems, we use a first-price, sealed-bid, reverse auction to negotiate the placement of VNFs and to make decisions regarding the routes of network forwarding graphs. In our reverse auction, agents secretly respond to the auctioneer agent, i.e. agents do not know each other's bids, with their cost to accomplish the auctioned goal. Given that we use a reverse auction, agents look for the lowest price to accomplish a certain task. Therefore, the lowest bidder wins the auction. Our reverse auction process is defined as follows.

Definition 4 (Reverse Auction): Let $\mathcal{G} = \{g_1, ..., g_n\}$ be a set of goals to be auctioned, an atomic bid β is a pair (g, p) where $g \in \mathcal{G}$ and $p \in \mathbb{R}_{\geq 0}$ is a positive real number that indicates the price of accomplishing goal g by an agent. The agent offering the lowest price p wins the auction and receives the goal g.

As said above, we use the FIPA ACL specification for supporting agent communication. For the auction process, we adopt the FIPA Contract Net interaction protocol[2]. In this protocol, an initiator agent sends a call for proposal (CFP), and each participant agent either replies with a proposal—*i.e.*, a bid $\beta = (g, p)$—or refuses to participate. Based on the replies, the initiator agent selects a bid (the lowest price wins the auction), and informs all participants of the result (accept proposal or reject proposal). The participant who had its proposal accepted, after performing what was requested, informs the result to the initiator.

In our model, when an agent is unable to accomplish a particular service (or achieve a particular goal), it initiates an auction by sending CFPs to candidate agents, *i.e.*, auction participants. Candidate agents are those that meet the requirements of the VNF to be allocated. They must evaluate whether they have the required capabilities to accomplish the service; if not, whether the capability should be added to the agent; and the associated price. When an agent has its proposal accepted, it is said that this agent *plays a role* in the multiagent system, and there is *commitment* between the initiator and participant agents. Next, we describe how agents place bids.

2) Bidding: When a CFP is received, an agent becomes aware that a service must be provided in the network (each service is associated with a role). In order to place a bid, an agent performs a two-step process. First, an agent must check whether it has the capabilities needed to play that role. Second, it must evaluate the cost of playing that role.

If the agent does not have the capabilities required to play the role, the agent must assess the utility of activating them. This is done using the utility function $\mathcal{U}$. Although this function focuses on the placement problem, it is a conflict solving mechanism that can be used by the chaining and selection solutions. To correctly inspect agents and select the best place to attach a new VNF, $\mathcal{U}$ is evaluated to specify the trade-off between the cost of activating a VNF and the benefits of having it activated. In its current form $\mathcal{U}$, defined as follows, verifies if the agent's virtual infrastructure satisfies the memory and processor requirements of the new VNF.

$$\mathcal{U}(C) = (c \times m)^{-1}$$

where c is the CPU usage and m is the memory usage, each having a normalised value in the range $[0, 1]$.

The capability associated with the role to be played assesses the cost of providing the service using the cost function $\mathcal{C}$. This evaluation—associated with the selection and chaining problems—is domain-dependent. That is, the knowledge of which VNFs should be selected in a given context and how to chain them with the VNFs already in use depends entirely on what is trying to be achieved, *e.g.*, improve resilience, increase performance, balance workload, etc.

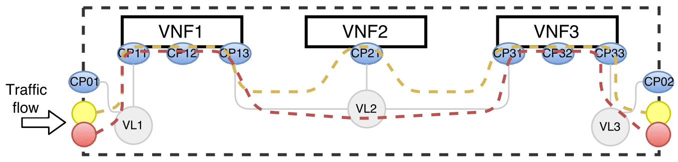
The final bid price p is a weighted sum using these two components, as shown next.

$$p = \omega_{\mathcal{C}} \times \mathcal{C}(B) + \omega_{\mathcal{U}} \times (1 - \mathcal{U}(C))$$

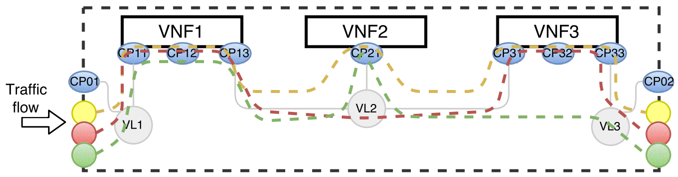
where ω_i are weights such that $0 \leq \omega_i \leq 1$ and $\sum_i \omega_i = 1$, so bids are in the range $[0, 1]$. We currently use equal weights $\omega_{\mathcal{C}} = \omega_{\mathcal{U}} = 0.5$ for each component. We use the complement of the utility function, because the highest the utility for the agent, the lowest the price. We provide the above cost and utility functions, together with the bidding process, as conflict solving tools that can be used by a domain specialist whose objective is to embed her knowledge into BDI agents, building a decentralised and autonomic NFV orchestrator.

To illustrate the auction process that decides which VNFs should handle specific traffic flows (chaining problem), we present the following example. Consider the network forwarding graph shown in Figure 4. Two different traffic flows enter the virtualised network through connection point 01 (CP01)

[1]http://www.fipa.org/repository/aclspecs.html
[2]http://www.fipa.org/specs/fipa00029/SC00029H.html



(a) Established routes for two traffic flows.



(b) Inclusion of a third traffic flow.

Fig. 4. Network forwarding graph for three distinct traffic flows.

and are chained differently by the decentralised orchestrator. Traffic flows may represent different protocols, distinct sessions or any other rule embed in the agents. In Figure 4a, the yellow traffic (top) goes to VNF1, VNF2 and then VNF3, while the red traffic (bottom) is redirected to VNF1 and then to VNF3. Suppose a distinct traffic flow is detected by the agent that controls virtual link 1 (VL1), which has to go through VNF1 and then either VNF2 or VNF3, as both can handle it. To decide to which VNF the virtual link will send this new flow, an auction is performed. Considering that VNF3 is already busy with the red and yellow traffic (assuming they are resource-consuming), the lowest price offered comes from the agent that represents VNF2. The chaining of the new incoming traffic flow is shown in green (bottom) in Figure 4b.

C. Case study: DDoS Resilience Strategy

Network resilience can be defined as the ability that a network has to recover from an incident [20], such as malicious attacks. We evaluate our model on a DDoS attack scenario, because it is a common issue, with a broad variety of solutions. In this section, we detail the network functions used in our scenario as well as how they are combined into a network resiliency strategy.

We designed roles, each associated with a capability, that represent commonly used network functionality and observed their collective behaviour emerging into a meaningful resilience strategy. Four different roles were implemented with BDI capabilities [21]: (i) *Link Monitor*: responsible for detecting any link overused in the network; (ii) *Rate Limiter*: performs traffic throttling to a specific link, to a specific IP address, and to specific flows (where both source and destination are known); (iii) *Anomaly Detection*: detects the target of a DDoS attack; and (iv) *Classifier*: analyses and identifies malicious flows, among all traffic to a specific host.

Initially, an analyst designs agents and indicates the initial state of the network, *i.e.*, which services should be provided and how. As an example, we show possible belief revision and goal generation functions of the *Link Monitor* capability in Table I. After indicating the virtualised infrastructure available and deploying the multiagent system, agents negotiate

TABLE I
BELIEF REVISION AND GOAL GENERATION FUNCTIONS OF THE *Link Monitor* CAPABILITY.

Belief Revision
$event(overUsage(link)) \rightarrow belief(overUsage(link))$ $\wedge belief(not\ anomalousUsage(link))$
$event(\neg overUsage(link)) \rightarrow belief(not\ overUsage(link))$
Goal Generation
$overUsage(link) \wedge not\ anomalousUsage(link) \rightarrow$ $goal(?anomalousUsage(link))$
$overUsage(link) \wedge \neg attackPrevented(link) \wedge$ $(anomalousUsage(link) \vee not\ anomalousUsage(link))$ $\rightarrow goal(attackPrevented(link))$

their distribution in the network using the *orchestrator* and *auction* capabilities, thus solving the initial placement problem imposed by the network startup. Next, agents negotiate on the path that information will take. Assume that a *Link Monitor* was added by the network operator during the network design and monitors a link of this network. The agent playing the *Link Monitor* role has the *Link Monitor* capability active and controls the VNF associated with it.

When the *Link Monitor* detects a link being overused, the event *event(overUsage(link))* is triggered, adding the beliefs *overUsage(link)* and *not anomalousUsage(link)* to the beliefs base in the belief revision step. An overused link is due to either a peak in legitimate traffic or an ongoing attack. Two goals are generated because the belief *overUsage(link)* is present: (i) *attackPrevented(link)* is responsible for restricting the traffic in the overused link; and (ii) *?anomalousUsage(link)* to determine if it is a malicious attack or not. Those goals are generated together in order to guarantee that the network remains functional while overcoming a challenge. Currently, only a throttling strategy is implemented. In the *Link Monitor* capability, there are no plans to achieve the goals *linkRateLimited(link)* and *?anomalousUsage(link)* within the *Link Monitor* capability. The first goal can be fulfilled by the *Rate Limiter* capability, while the second by the *Anomaly Detection* capability. In our scenario, each agent is composed of only one role. Thus, agents communicate about their unfulfilled goals, requesting other agents if they can achieve those goals.

The agent playing the *Link Monitor* role then starts two auctions, one for each unfulfilled goal. Candidate agents in the simulation evaluate their utility and cost functions for each auction created by the *Link Monitor*. Considering the auction to fulfil the *linkRateLimited(link)* goal, an agent that already has the *Rate Limiter* capability active, *i.e.*, it is playing the *Rate Limiter* role and is underused, will likely win this auction. However, if there is no agent currently playing this role or the current *Rate Limiter* is overloaded, an agent with no active *Rate Limiter* capability may win the auction. In the latter case, the winning agent activates the *Rate Limiter* capability, which now controls a new VNF—the Virtualised Infrastructure Manager performs the process of handling virtual resources. Consequently, this agent now plays the *Rate Limiter* role in the multiagent system. The same happens with the *?anomalousUsage(link)* goal, which results in an agent playing the *Anomaly Detection* role.

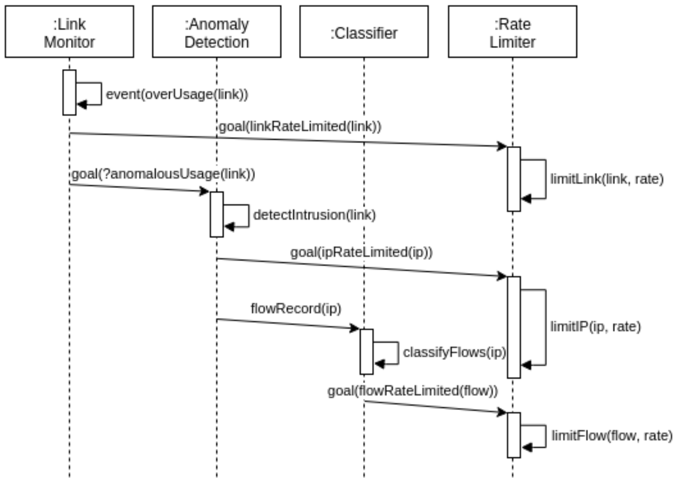


Fig. 5. DDoS resilience strategy.

The *Anomaly Detection* starts processing packets coming through, with the objective of finding out which IPs are being targeted by the DDoS attack. When potential IPs being targeted by the attackers are identified, it starts an auction for limiting only the traffic destined to the targeted IPs by a certain threshold. After reducing the malicious traffic, the link limit previously established is removed, allowing legitimate traffic while the attack is mitigated. Lastly, when the *Anomaly Detection* message reaches the *Classifier* with the targeted IPs, the latter can start recording the traffic to be analysed. If the attack continues and the *Classifier* has enough information to determine the source IPs responsible for the malicious traffic, then a message is sent to the *Rate Limiter*. Once the *Rate Limiter* starts limiting specific malicious flows, then legitimate traffic to the IPs being attacked is allowed without restrictions. Note that all these interactions occur through auctions. As result of the interaction between agents, the emerged behaviour that can be observed in our multiagent system is shown in Figure 5, in which we omit auction processes for simplicity.

IV. Implementation and Evaluation

In this section, we describe the implemented prototype and an experimental evaluation of our decentralised NFV orchestrator based on BDI reasoning.

A. Prototype Implementation

Our NFV orchestrator was built using `BDI4JADE` [22] and `ESCAPE` [23]. `BDI4JADE` was used for implementing the autonomous components. Although other BDI platforms are available, `BDI4JADE` allows agents to be implemented in pure Java, which facilitates the implementation of mechanisms that integrate autonomous agents with the NFV testbed [22]. We chose `ESCAPE` for the NFV testbed because it is a highly scalable prototyping tool that combines SDN software with a simple Python interface [24]. `ESCAPE` provides a graphical user interface that integrates `Mininet` and an NFV environment along with a simple orchestrator that can be easily replaced [23]. Furthermore, `ESCAPE` does not use a pool of

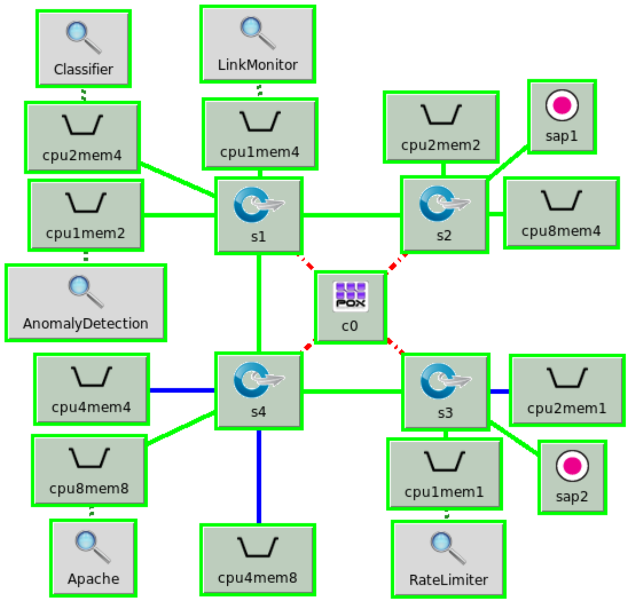


Fig. 6. Network topology and placement of VNFs in `ESCAPE`.[3]

CPU and memory to allocate resources dynamically to VNFs; instead, CPU and memory have to be manually allocated into containers at design time. Each VNF is associated with a capability of a BDI agent, and each capability implements the behaviour needed for an agent to play a role.

When a simulation starts, the configuration of each VNF is taken into account by the built-in placement algorithm, which is aware of all the container specifications as well as VNFs. We replaced the original placement algorithm by our decentralised solution. After the orchestration algorithm assigns VNFs to containers, BDI agents running in the `BDI4JADE` platform (which in turn runs on top of the `JADE` platform) start to communicate with the VNFs in `ESCAPE` through TCP sockets, enabling agents to control the VNFs.

B. Experiment Setup

In our evaluation testbed, we adopted an `Apache` server as the service-provider VNF. The `Apache` server version 2.2.22 with `PHP` module version 5.4.9 was configured to respond with either an informative `PHP` page[4] or a video stream. In addition, four VNFs that support the provision of security to the network are added to the simulation: (i) *Link Monitor* was implemented using `ifstat`[5] tool; (ii) *Rate Limiter* communicates with a `OpenvSwitch` [25] switch to apply queue-based rate limiting rules to network interfaces and specific source/destination IPs; (iii) *Classifier* as well as (iv) *Anomaly Detection* use `Snort` [26] version 2.9.2.2 to perform their activities.

[3]In `ESCAPE`, blue links represent inactive links (no VNFs assigned); red links represent control links between switches and the SDN controller; and, green links represent active links. Green dotted lines indicate which VNF is attached to a container.

[4]http://php.net/phpinfo

[5]http://gael.roualland.free.fr/ifstat/

Figure 6 contains a screenshot of the network topology. Our simulation is composed of four `OpenvSwitch` switches connected to a `POX`[6] controller. Each switch has two or three containers connected to them. Each container was named after its virtual allocation of resources. For example, *cpu4mem4* has four processors and 4GB of memory. We do not take into account storage and network interfaces in this simulation. Also, this figure depicts the placement of VNFs through a preliminary placement implementation, due to ESCAPE limitations (it does not support placement changes at runtime). VNFs were allocated according to their configuration requirements, *e.g.*, the *Apache* VNF requires eight processors and 8GB of memory. The security VNFs require only one processor and 1GB of memory, except for the *Classifier*, which requires two processors. The placement algorithm implemented for this experiment is limited. It does not support the allocation of VNFs during the simulation. It is one of our goals to implement a robust and complete placement algorithm, providing all the functionality proposed in Section III-B.

C. Experimental Evaluation

We first perform a functional evaluation of the prototype to demonstrate the emergent behaviour of the agents. In our experiments, the *Link Monitor* VNF monitored the interface that connects switch `s4` to the *Apache* VNF. User behaviour is simulated through 30 hosts that randomly choose to either request HTTP pages or a video stream. From a group of 6 hosts, there is 1 host that requests to watch the video stream while the other 5 request an HTTP page, similarly to the work of Silva *et al.* [27]. Network anomalies were generated with `scapy` [28], which allows the generation of realistic anomalous traffic. In this simulation we perform a SYN flood attack, where 10 malicious hosts send TCP SYN packets with a random TCP source port to either the HTTP or the video streaming ports (8080 for the video streaming and 8000 for the HTTP service). As a result, the server allocates resources to handle each request and responds with an ACK to the malicious host, which does not reply, thus leaving the server with an open TCP connection. Both benign and malicious traffic entered the network through service access points 1 and 2 (`sap1` and `sap2` in Figure 6).

The traffic volume from switch `s4` towards the *Apache* VNF is shown in Figure 7. After around 10 seconds, the DDoS attack started with all malicious hosts making requests at the same time. When the attack starts, the traffic reaches 600 KB/s (1). Shortly after that, the *Link Monitor* detects a peak and informs the *Rate Limiter* agent to apply a new limit to the switch in which `Apache` is connected. This change successfully reduces traffic volume to around 300 KB/s. Subsequently, the *Anomaly Detection* analyses the traffic and finds out that it is targeting the `Apache` server. Having that information sent to the *Rate Limiter*, it decides to apply a rule that reduces all the traffic to the *Apache* VNF to 150 KB/s (2). Furthermore, the *Classifier* is warned and, at around

[6]http://www.noxrepo.org/pox/about-pox/

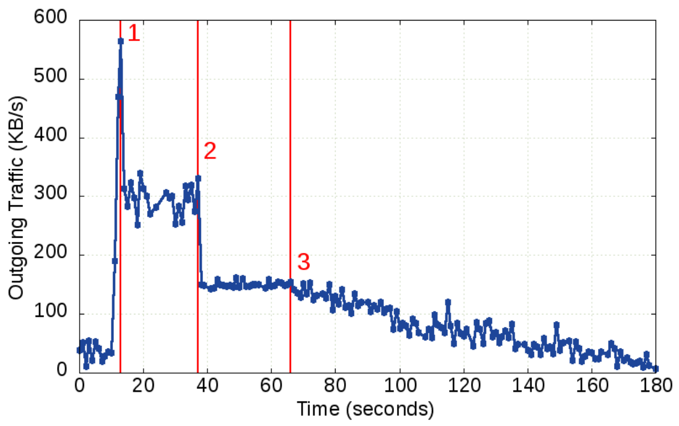


Fig. 7. Traffic volume from switch `s4` towards the `Apache` server.

65 seconds into the simulation it starts reporting the specific IPs flooding the `Apache` server (3). The *Classifier* sends a message to the *Rate Limiter* every time a new malicious IP is identified, which then creates a rule to block the traffic based on the source IP address of the malicious host. Consequently, the traffic slowly descends into acceptable levels.

With respect to the scalability of our solution, Figure 8 shows the average memory consumption and the average number of messages exchanged, in scenarios with an increasing number of agents. Memory consumption only accounts for `BDI4JADE` agents, as we do not consider the memory used by `Mininet`. Further, we also evaluate the performance of the solution considering scenarios in which agents may have a large number of goals unrelated to the resilience strategy (*i.e.*, agents that perform multiple roles). To do this, we generated an increasing number of synthetic goals, which simulate the impact of real plans by taking a random amount of time to complete (between 0 and 100 milliseconds). Figure 9 shows how long the *Rate Limiter* took, on average, to apply the throttling strategy to the anomalous link, considering an increasing number of other unrelated goals. The linear growth in both messages exchanged, memory usage and goal achievement time indicate that our system can scale to large networks. Note that we did not use goal filtering and prioritisation, which in a real scenario would be adopted to improve scalability, reducing the number of plans running at the same time.

V. Related Work

NFV has changed the way network functions are deployed and managed, posing new challenges to network operators. Management systems will have to cope with short-lived networks driven by per service demands, guaranteeing that all required functions are instantiated in an orderly and on-demand basis. This section briefly describes previous work related to the automation of the orchestration of VNFs.

Giotis *et al.* [7] proposed a policy-based approach, creating a centralised orchestrator based on Ponder2 [29]. They represent VNFs as an extension of Managed Objects (MO), allowing VNFs to interact with management policies and provide

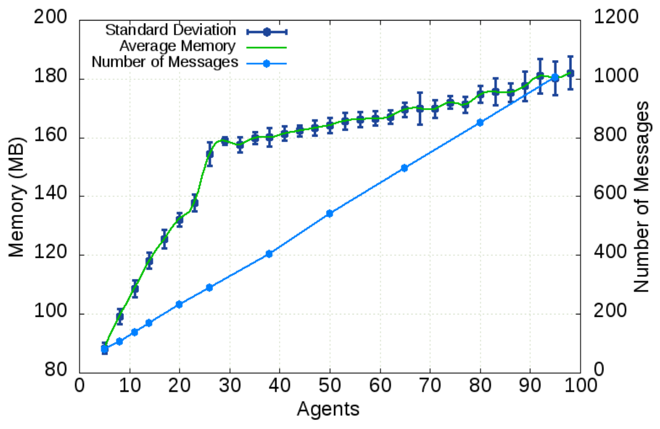


Fig. 8. Memory consumption and number of messages exchanged.

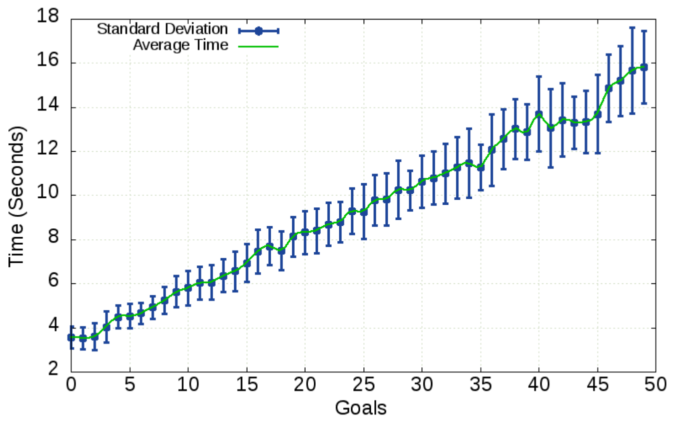


Fig. 9. Time taken by the *Rate Limiter* to apply the throttling strategy.

the desired NFV Service. The orchestrator is a collection of Event-Condition-Action (ECA) policies, authorisation and role assignment of resources. Differently from our work, however, ECA policies in [7] have to be manually specified by a human operator, whereas in our work the behaviour of VNFs emerges through an automated reasoning process.

Focusing on security, the orchestration of a Deep Packet Inspection (DPI) system [8] was proposed to detect and prevent DoS and DDoS attacks in SDN. It was tested on the SAVI testbed [30], which uses a centralised module aware of every network route, using this information to attach an Intrusion Detection System (IDS) where needed. We propose, instead, a decentralised approach where the interaction between autonomous components orchestrate the network behaviour.

Clayman *et al.* [9] developed three distinct placement algorithms that run on a global controller, which is aware of VNFs through local controllers running on physical hosts. Local controllers report their status to the global controller, which then decides whether new virtual routers will be added or not. The *Least Used Host* algorithm selects the physical host with the least number of virtual routers to receive a new virtual router. *Least Busy Host* algorithm selects host with the least amount of traffic on its virtual routers. Finally, *N at a time in a Host* allocates up to N virtual routers on a host. Other hosts can be set to a power saving state. In our work, instead of relying on a monitoring system, we propose the usage of a utility function and an auction process, providing agents with decision-making mechanisms.

Donadio *et al.* [10] proposed a PCE-based [31] orchestrator, which is composed of a traffic analyser that controls computational resources and log messages to perform virtual machine orchestration in real-time. Their orchestrator can be executed in multiple physical hosts simultaneously, where each orchestrator manages its resources. In addition, there is a centralised alternative in which one orchestrator is responsible for orchestrating resources in different network domains. Their work mainly focuses on minimising energy consumption of the IT infrastructure through the provision of orchestration algorithms that are oriented towards energy efficiency. Although Donadio *et al.* [10] propose to use multiple orchestrators, they are still centralised components that control part of the virtualised network, while we propose that all virtualised components work together towards a decentralised NFV orchestrator.

VI. Conclusion

NFV orchestration allows the composition of specific VNFs into aggregate service chains. The orchestration problem can be typically decomposed into the VNF selection, placement and chaining subproblems. Effectively solving these problems has the potential to improve the usage of resources, reducing energy consumption and also CAPEX and OPEX.

Although different approaches have been proposed to solve the orchestration problem, previous work relies either on manual or on centralised approaches. Instead, in this work we propose a distributed and decentralised NFV orchestrator based on BDI reasoning. The orchestrator can be used to coordinate the selection, placement, and chaining of VNFs. The emergent behaviour achieved by our approach is arguably more robust, since it can adapt to different situations and topologies, or in case of a component failure.

We evaluated our theoretical model through simulation using the `BDI4JADE` platform along with the `ESCAPE` testbed. In our simulation, we explored the automatic orchestration of VNFs on a DDoS attack scenario and demonstrated that the traditional monolithic orchestrator can be implemented as distributed autonomous components. We focused on proposing tools needed for the development of decentralised orchestrators, encompassing solutions and/or tools for the three problems (selection, placement, and chaining). Future work will address how these can be combined in broader scenarios.

Acknowledgment

This work receives financial support of CNPq, project ref. 442582/2014-5. Ingrid Nunes thanks for research grants CNPq ref. 303232/2015-3, CAPES ref. 7619-15-4, and Alexander von Humboldt, ref. BRA 1184533 HFSTCAPES-P. Alberto Schaeffer-Filho would like to thank CNPq for research grant ref. 311088/2015-5.

References

[1] 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 and Tutorials*, 2015.

[2] S. Verbrugge, D. Colle, M. Pickavet, P. Demeester, S. Pasqualini, A. Iselt, A. Kirstädter, R. Hülsermann, F.-J. Westphal, and M. Jäger, "Methodology and input availability parameters for calculating opex and capex costs for realistic network scenarios," *Journal of Optical Networking*, vol. 5, no. 6, pp. 509–520, 2006.

[3] 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.

[4] ETSI, "Network functions virtualisation (nfv): Architectural framework," *ETSI*, 2013. [Online]. Available: http://portal.etsi.org/portal/server.pt/community/NFV/367

[5] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, 2014, pp. 459–473.

[6] 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

[7] K. Giotis, Y. Kryftis, and V. Maglaris, "Policy-based orchestration of nfv services in software-defined networks," in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*. IEEE, 2015, pp. 1–5.

[8] P. Yasrebi, S. Monfared, H. Bannazadeh, and A. Leon-Garcia, "Security function virtualization in software defined infrastructure," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 778–781.

[9] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, "The dynamic placement of virtual network functions," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–9.

[10] P. Donadio, G. B. Fioccola, R. Canonico, and G. Ventre, "A pce-based architecture for the management of virtualized infrastructures," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. IEEE, 2014, pp. 223–228.

[11] 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 *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 98–106.

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

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

[14] D. Singh, S. Sardina, L. Padgham, and S. Airiau, "Learning context conditions for bdi plan selection," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 325–332.

[15] J. Faccin and I. Nunes, "Bdi-agent plan selection based on prediction of plan outcomes," in *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, vol. 2, 2015, pp. 166–173.

[16] P. Busetta, N. Howden, R. Rönnquist, and A. Hodgson, "Structuring bdi agents in functional clusters," in *International Workshop on Agent Theories, Architectures, and Languages*. Springer, 1999, pp. 277–289.

[17] I. Nunes, *Improving the Design and Modularity of BDI Agents with Capability Relationships*. Cham: Springer International Publishing, 2014, pp. 58–80. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-14484-9_4

[18] T. Finin, R. Fritzson, D. McKay, and R. McEntire, "Kqml as an agent communication language," in *Proceedings of the third international conference on Information and knowledge management*. ACM, 1994, pp. 456–463.

[19] Y. Labrou and T. Finin, "Semantics for an agent communication language," in *International Workshop on Agent Theories, Architectures, and Languages*. Springer, 1997, pp. 209–214.

[20] J. P. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Computer Networks*, vol. 54, no. 8, pp. 1245–1265, 2010.

[21] I. Nunes and A. Schaeffer-Filho, "Reengineering network resilience strategies using a bdi architecture," *AutoSoft*, p. 25, 2014.

[22] 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.

[23] A. Csoma, B. Sonkoly, L. Csikor, F. Németh, A. Gulyás, W. Tavernier, and S. Sahhaf, "Escape: Extensible service chain prototyping environment using mininet, click, netconf and pox," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 125–126, 2015.

[24] 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.

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

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

[27] 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.

[28] P. Biondi, "Scapy," *see http://www. secdev. org/projects/scapy*, 2011.

[29] K. Twidle, E. Lupu, N. Dulay, and M. Sloman, "Ponder2-a policy environment for autonomous pervasive systems," in *Policies for Distributed Systems and Networks, 2008. POLICY 2008. IEEE Workshop on*. IEEE, 2008, pp. 245–246.

[30] J.-M. Kang, H. Bannazadeh, and A. Leon-Garcia, "Savi testbed: Control and management of converged virtual ict resources," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 664–667.

[31] A. Farrel, J.-P. Vasseur, and J. Ash, "A path computation element (pce)-based architecture," RFC 4655, August, Tech. Rep., 2006.