# A framework for enabling security services collaboration across multiple domains

Daniel Migault*, Marcos A. Simplicio Jr.†, Bruno M. Barros†, Makan Pourzandi*,
Thiago R. M. Almeida†, Ewerton R. Andrade† and Tereza C. M. B. Carvalho†
*Ericsson Security Research, Montreal, Canada
Email: {daniel.migault,makan.pourzandi}@ericsson.com
†Escola Politécnica, Universidade de São Paulo, São Paulo, Brazil
Email: {mjunior,bbarros,talmeida,eandrade,carvalho}@larc.usp.br

***Abstract*—Collaboration among Security Service Functions (SSF) is expected to become as essential to SECaaS (SECurity as a Service) systems as elasticity is to IaaS (Infrastructure as a Service). The virtualization opens new era in network security as new security appliances can be created on demand in appropriate places in the network. At the same time, the increasing size and diversity of attacks make it necessary to come up with new approaches for more efficient and more resilient security mechanisms. In this paper, we propose a new framework leveraging SDN (Software Defined Networking) and SFC (Service Function Chaining) to enhance the collaboration among different SSFs to mitigate large scale attacks. We describe a framework that allows SSFs from different domains to negotiate and dynamically control the amount of resources allocated for collaboration, in what we call a "best-effort" collaboration mode. This SSF collaboration framework creates a distributed mitigation system for handling large scale attacks in a dynamic and scalable manner. The efficiency and feasibility of this framework is experimentally assessed, showing that our approach incurs low overhead, increases the amount of traffic treated by SSFs and reduces the dropped traffic due to the lack of resources from the security mechanisms.**

## I. Introduction

In the context of Network Function Virtualization (NFV), Security Service Functions (SSFs), a special type of Service Functions (SFs), usually take the form of on-path services instantiated across different administrative domains for detection and mitigation of security threats (e.g., a firewall providing traffic filtering) [51]. To achieve their goals, SSF must be able to dynamically scale in response to attacks (e.g., Distributed Denial of Service – DDoS), which typically requires some underlying orchestration mechanism. For example, as illustrated in Fig. 1, suppose that a web-server is instantiated in a cloud computing environment, within a virtual data center (VDC). The traffic from the end-users is conveyed by an Internet Service Provider (ISP) to the Cloud, then to the VDC by the Cloud Provider, and finally to the end server. In this scenario, each domain is likely to instantiate SSFs intended to address similar threats, considering only locally-available information, and also to over provision those SSF so they can handle eventual traffic fluctuations. These issues, typical of environments with multiple administrative domains, are becoming more common with the trend of moving computation to the networks' edge, as observed in fog computing [6], [47] and in mobile networks in general [37].

Enabling the SSF in this scenario to collaborate is expected to provide several advantages. First, since the VDC receives all traffic and runs the end server, it is better placed for detecting application- or service-specific attacks, as well as DDoS; the VDC can then inform other domains of the ongoing attack, leading to *better detection*. Then, for *better mitigation*, the VDC can define a response strategy and implement it, either locally or by (partially) outsourcing this task to other SSFs on the attack's path. If this strategy is conceived in such a manner that the SSF in each domain concentrate their efforts on tasks that they perform better, avoiding redundancies, a natural result is *better resource usage*: for example, the Cloud could instantiate SSF focused on attack detection, since it can inspect all incoming traffic (even if encrypted) and run application-specific tools; the ISP, on its turn, can deploy SSF for filtering malicious traffic closer to its sources. Even when the collaboration scope is a single domain, resource usage optimization is also possible, as an overloaded virtual machine (VM) can offload some tasks to another VM with idle resources, avoiding the creation of new instances. Finally, one main advantage of SSF collaboration is that it creates a *highly dynamic and scalable* architecture for attack mitigation, enabling the robust and cost-effective Security-as-a-Service (SecaaS) solutions [17]. For example, in DDoS and similarly distributed attacks, the aggregation of resources on the attackers' side usually overcomes the resources available at any single point on the target's side. A common strategy for dealing with this issue is to redirect the traffic to a scrubbing center, which is supposed to have enough resources to handle the attack. With collaboration, however, the defense is also distributed, combining the capabilities of several SSF to overcome the attacker's resources and reducing the system's dependency on scrubbing centers.

The literature has many examples of how collaboration can improve security (for a survey, see [35]). These benefits led to many initiatives based on collaboration protocols, most of which are focused on signaling the detection of DDoS attacks [11] or mitigating them by outsourcing tasks toward SSF closer to the attack's sources [30], [32]. Unfortunately, however, collaboration has seen limited deployment in practice. In

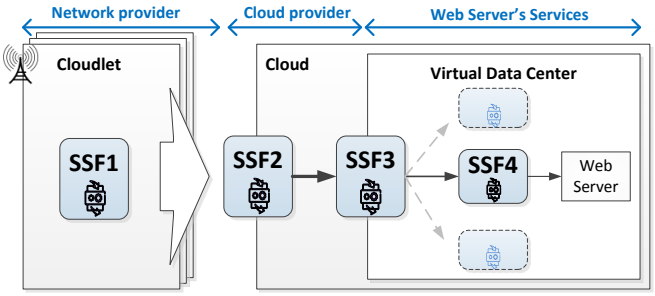



Fig. 1. Collaboration Scenario

part, this is due to the cost of adding such capabilities in appliance routers and to the limited mitigation capabilities made available by vendors. Nevertheless, today this issue can be addressed by Software Defined Networks (SDNs) [26], [44], which can both reduce management costs and increase the possible interactions between domains. Indeed, the interest in enabling and orchestrating collaboration among domains is today the focus of Internet working groups such as the Interface to Network Security Functions (I2NSF) [14] and the DDoS Open Threat Signaling (DOTS) [11], which shows the interest of major industry players. Despite those advances, a challenge for making collaboration feasible is that, even though it is expected to benefit all, collaboration requires one domain to allocate resources for another domain, which may not seem beneficial at first sight. To encourage such practice, each domain must be able to dynamically control the amount of resources allocated for a collaboration, negotiating it in real time and in a flexible manner. For example, a domain should be able to agree on handling only a fraction of the traffic, considering its current load, avoiding undesirable situations such as resource exhaustion or resource hijacking. Collaboration agreements established for this purpose can be significantly simpler than the formal agreements usually established with scrubbing centers, as they can be temporary and consider the current capabilities of each domain.

Aiming to provide such flexibility to multi-domain collaboration, we propose and discuss a collaboration framework that, when compared with previous work, brings the following contributions. First, it enables collaboration among SSF within or among different administrative domains, as well as the provisioning of new SSF instances on demand if deemed necessary. Second, these instances may be associated to: (1) different SSF types, so the collaboration consists in requesting a functionality; or (2) to a same SSF type, in which case the collaboration consists in outsourcing part of the SSF's load. The collaboration can, thus, be seen as an agreement on dedicating resources for a given SSF, allowing a temporary cooperation to be established among autonomous systems in what we call a "best-effort" mode: whenever the amount of resources required for performing the outsourced task reaches the agreed threshold, the task is not performed by the SSF that engaged in the collaboration, and untreated packets are marked so they can be treated later.

The rest of this paper is as follows. Sec. II discusses a motivational example, showing how collaboration can be useful when dealing with a DDoS attack. Sec. III then describes the related work on collaborative security. The elements and inner working of the proposed framework are presented in Section IV and Section V, respectively, assuming an orchestrated approach for the collaboration among domains. To evaluate our proposal, Sec. VI shows the results of our preliminary experiments using the OpenDaylight SDN Controller [34] and the Service Function Chaining (SFC) architecture [21]. Sec. VII presents our final considerations.

## II. Collaboration requirements: a DDoS motivational example

Before describing our proposal, we discuss a motivation scenario from which its requirements were built. The collaboration solution proposed in this work is mainly focused on volumetric infrastructure-layer DDoS attacks, which currently represent more than 97% of the attacks registered by leading content-delivery network (CDN) and cloud service Providers such as Akamai [2]. Nevertheless, the collaboration architecture hereby described can easily address the requirements for the mitigation of application-layer DDoS attacks from the deployment of appropriate application-layer SSFs. The motivational scenario is similar to the one analyzed in [54]: a possible implementation of a web server in a cloud infrastructure, aiming to leverage its elasticity to mitigate DDoS attacks. The example, illustrated in Fig. 1, considers a web server protected by a chain of 4 SSF: $SSF_4$, a Web Application Firewall (WAF) hosted at the web server; $SSF_3$, a gateway firewall at the VDC's entry point; $SSF_2$, a firewall placed at the cloud's border ; and several instances of $SSF_1$, cloudlet firewalls at the ISPs and, thus, closer to the web server's clients. The goals of collaboration in this scenario are: (I) mitigate the DDoS attacks as close as possible to the source, so their overall impact (e.g., in terms of latency and drop rate for legitimate packets) remains minimal; (II) reduce the infrastructure costs, by avoiding the conveying of packets that will be dropped later, so each SSF can be dimensioned accordingly; and (III) make detection and mitigation tasks more efficient by enabling different domains to share resources for such activities.

To work with a more concrete example, suppose for simplicity that all firewalls are dimensioned at $100\sigma$, where $\sigma$ is a generic measure of computational resources. Suppose also that, at some point in time, their resource usage considering their own security tasks is $50\sigma$. Then, due to a DDoS attack, the resource usage at $SSF_1$ and $SSF_4$ rise both to, say, $80\sigma$; for the same reason, if $SSF_2$ and $SSF_3$ keep processing the same tasks, their resource usage would become respectively $55\sigma$ and $105\sigma$, with $SSF_3$ filtering most of the DDoS traffic as currently configured. In this case, the queues in $SSF_3$ would continuously grow, increasing latency, until its buffers are full and packets start being dropped. One could deal with this issue by creating a second instance of $SSF_3$ and attaching both instances to a load balancer; it would be better, however,

if $SSF_3$ could simply *offload* some tasks (say, taking $25\sigma$) to $SSF_2$, so both would end up with a resource usage of $80\sigma$, preserving Quality of Service (QoS) without any extra SSF. With a multi-domain collaborative solution as hereby proposed, that can be done to deal with the attack.

Suppose that, some time after the attack is already under control due to the collaboration, the DDoS pattern changes: it starts including packets that are only filtered by rules installed at $SSF_4$. This raises the resource usage of the corresponding filtering task $\tau$ from $0\sigma$ to $30\sigma$ (and, thus, $SSF_4$'s resource usage goes from $80\sigma$to $110\sigma$). Unfortunately, $SSF_4$ cannot offload the whole task to any other firewall, as that would just relocate the problem. Nevertheless, $SSF_4$ can collaborate with other SSF for sharing the corresponding load, in a "best effort" manner. For example, if $\tau$ would take the same amount of resources wherever executed, $SSF_4$ could ask $SSF_3$ and $SSF_2$ to invest $10\sigma$ each on this task, so the load on $SSF_4$, $SSF_3$ and $SSF_2$ would be $90\sigma$. It is possible, however, that running $\tau$ with only $10\sigma$ in $SSF_2$ will not be as effective as in $SSF_4$, since the volume of traffic at $SSF_2$ is higher; after all, in $SSF_4$ task $\tau$ runs only on packets that have not been filtered by $SSF_3$, whereas $SSF_2$ potentially sees some packets that are yet to be filtered by $SSF_3$. Therefore, some negotiation is likely to be necessary to decide a reasonable workload division that ensures no single firewall gets overloaded.

Finally, there may be situations in which neither offloading or best-effort load-sharing is feasible, so a new SSF needs to be instantiated to avoid vertically scaling existing SSF. Even in this case, a multi-domain collaborative approach provides more flexibility than a simple load balancing: since the new instance added to the chain does not need to be an exact copy of some existing SSF, optimizations are possible. For example, the new SSF could be placed at the beginning of the chain and receive tasks that can filter many packets (preferably with little processing), thus reducing the network's overall load. In addition, or alternatively, the new instance could be optimized for the tasks it will perform (e.g., making it more processing- or memory-oriented), creating cost optimization opportunities based on the cloud's pricing model [45].

From this analysis, we can draw the following requirements for the proposed solution's design: (1) SSF should be able to offload tasks to any other capable SSF, even it they are in different domains; (2) SSF should be able to engage in temporary and "best-effort" collaborations, treating only a fraction of the traffic to which some offloaded task applies, and for an agreed upon period; (3) the framework should allow SSF to be instantiated on demand, whenever necessary. As it should be clear from those requirements, we focus on collaboration aspects rather than addressing DDoS attack detection or mitigation mechanisms themselves. The motivation for this approach is that, as highlighted in recent surveys on this subject (e.g., [56], [53]), whereas many detection and/or mitigation solutions exist, enabling collaboration among mechanisms, thus increasing detection accuracy and mitigation efficiency while optimizing resource consumption, remains as an open challenge. It is also out of our scope to discuss specific algorithms for optimizing the distribution of tasks among SSF, as each domain may prefer to employ different mechanisms for this purpose (e.g., like those described in [7], [57]).

## III. Related Work

Most collaborative security solutions in the literature focus on DDoS, aiming to create an environment for their detection, analysis and mitigation. The usual approach for doing so, adopted in solutions such as DiCoDefense [55], CoDef [25], Drawbridge [26], BGP Flowspec [32] and Sahay *et al.*'s architecture described in [44], is to rely on a central controller with a global view of the network. This controller is responsible for coordinating the collaboration among security modules, deploying the required SSF along the network, and/or re-routing flows to a path with enough resources and needed functions. Controllers from different domains may also cooperate to define a strategy against DDoS attacks considering alerts locally raised. For example, one can define a separate datapath (e.g., with VLAN tags) for a suspicious flow, including in this path all SSFs required for its analysis [44]. Our proposal behaves similarly, but it also aims to take into account the load on the SSFs in a path, provisioning new ones if necessary. In addition, the collaboration between elements in the mentioned solutions is usually of the type "all-or-nothing", as all traffic matching a given security policy must be handled by the node that accepts to perform a given task. As discussed in Sec. II, this is not necessarily optimal in terms of resource usage when compared to the "best-effort" approach hereby proposed.

Solutions with no central controller also exist, such as DefCOM [38] and Pushback [30]. In this case, a node has autonomy to send SSF requests to its upstream neighbors (i.e., closer to the packets' source) when congestion is detected, and assume that they are able to fulfill the request (e.g., apply rate-limiting policies). There are also hybrid approaches such as NetSecu [8], in which nodes in a path are autonomous but can be dynamically reorganized by a central controller for better filtering the traffic. Such non-centralized approach is also supported in our proposal, which may either rely on a central orchestrator or have SSFs collaborating directly; as discussed in Sec. V. However, the former has some advantages in terms of privacy and easiness of deployment.

Resource negotiation among nodes is another important aspect of collaborative systems, as it is usually necessary to ensure that a service can be reliably outsourced (even if temporarily) before actually doing so. DefCOM, for example, allows nodes to refuse a service deployment request based on their resource availability, while in CoDef the node initiating the collaboration sends a resource allocation request to the central controller before the route deployment request. The negotiation approach hereby presented provides the initiating node with the ability to outsource security services for a limited period of time, and in a best effort manner.

Finally, the general problem of sharing resource among domains has already been studied in by grid computing solutions [19], [15], [20]. In this context, the issue is usually addressed by means of the Service Oriented Architec-

ture (SOA), as originally standardized by the Open Grid Service Architecture (OGSA) [16], [18] and Infrastructure (OGSI) [48], and then superseded by Web Service Resource Framework (WSRF) [10]. More specifically, Agreement Based SOA (ABSOA) enables the association of Web Services (WS) to resources and Service Level Agreements (SLA) by means of a so-called WS-Agreement [3], [29], [28]. Even though SOA is quite flexible, it has been almost exclusively defined for Web Services (WS). Therefore, its integration in the context of NFV for defining collaboration agreements among SSF seems suboptimal. For example, SSFs are location-dependent and need to be on the path of the customer's traffic, which is not exactly compatible with WS mechanisms such as WS discovery. WS assumes a network topology and location agnostic approach, as the main target of WS was the web services through out the Internet. In contrast, our approach places SSFs on the communication path between different services as we consider NFV. Therefore, our approach depends on the network topology and the location for different services part of the collaboration. SSF orchestration is also very dynamic, with different SSF being instantiated and configured on demand in response to non-deterministic traffic variations, making the interaction among SSFs and network controllers more complex than the typical case with WS. This motivates the adoption of protocols such as NetConf [12] and RESTConf [4] for enabling collaboration among SSF, especially considering their capabilities to configure, rollback, test push network configurations. Indeed, these are the types of technologies considered in the collaboration architecture hereby proposed.

## IV. Enabling collaboration among SSF

In this section, we discuss the elements involved in the envisioned collaboration models, as well as their architectural organization. We start by defining classes of SSF that may collaborate and the services that are likely to be provided by each class. Then, we describe the set of attributes that should be included in the agreement among SSF to enable different forms of collaboration, including the best effort mode.

### A. Classes of SSF

Although the term "SSF" is used to designate generic security elements, the SSF hereby considered can be grouped in the following classes:

- *On-path SSF*: services that can be performed at various places on the path, like firewalling, email attachment analysis, or DDoS mitigation [46], [50]. For example, firewalls are typically located at the boundaries of various domains, as well as within a domain. In some cases, however, such SSF may have to be performed at the end point (e.g., when end-to-end encryption is in place).
- *Endpoint SSF*: services that are performed at the network's endpoints. SSFs are classified as Endpoint SSFs if their tasks, when outsourced, have to be handled entirely by the target SSF instance. Typical examples include tokenisation and Identity Management.
- *Location-dependent SSF*: services that must be performed at specific locations. One example is encrypters, that must be placed at the end of a tunnel, or edge firewalls protecting the borders of domains. Note that this category does not take into account situations in which the location is imposed by physical constraints (e.g., availability of a specific hardware in the network).

### B. Collaboration Agreement (CA)

This section provides an overview of the Collaboration Agreement (CA) agreed through the CA Protocol (CAP) [36]. Although this section assumes the CA is agreed between two SSFs, as detailed in Sec. V-B and Sec. V-C one or more orchestrators may be involved. The CA is expressed via a YANG model with a similar high level structure as agreements defined in WS-Agreement [3]. The CA is composed of a *context* that identifies the CA, a *service description* that defines the SSF and a *guarantee terms* that lists the agreed SLAs.

More specifically, the *context* is composed of a CA identifier, an expiration time, an identification of the involved parties as well as their respective roles. The SSF initiating the CAP is designated as the *Initiator*, while the SSF accepting the collaboration is designated as the *Provider*. In addition, it also contains a description of the traffic including its direction as well as the collaboration mode. There are two possible collaboration modes. The *resilient* mode indicates a full offload to the provider, while the *best effort* mode indicates the provider only treats the traffic up to the agreed resources indicated as an SLA. For the non treated traffic, the best effort mode also includes a description of the *Alternate Path* where non treated traffic is being steered as opposed to the *Standard Path* where the traffic is treated. For example, when the collaboration involves two independent administrative domains, those paths could be represented by different VXLAN [31] or GRE [13] tunnels configured with particular keys, or two distinct IPsec tunnels with different security parameter indexes (SPIs) [23]. Consequently, the Initiator and the Provider are expected to agree on the type of tunnel and their associated IDs and/or keys. In contrast, such tunnels are not required when the collaboration remains within a single administrative domain. In this case, the standard and alternate paths can be represented by paths with different IDs or via encapsulation, as further discussed in Sec. V-C.

The *service description* contains a description of the service capabilities, and the supported YANG models [5] used to configure the SSF with Netconf or Restconf. Currently capabilities are associated to a single SSF, but it is expected that the service description could include a composition of different SSFs. In addition, the description also includes information to reach the SSF such as the IP addresses, the authentication credentials and the supported configuration protocols (Netconf or Restconf).

*Guaranteed terms* list the agreed SLAs composed of a resource type, the associated guaranteed level, the notification level that triggers a notification to the Initiator as well as associated penalties. Note that the notifications type is already

defined by YANG [9]. The resources in a CA can be expressed in specific ways, such as a combination of various computational resources, such as CPU, memory, I/O and bandwidth; more specific attributes may be detailed if necessary, but these parameters are commonly used by Cloud Service Providers such as Amazon EC2 as the basic unit of a computational [54]. The manner by which such resources are controlled is left for the Provider's implementation; one possible approach is to leverage containers and micro services technologies [49].

In order to reach a consensus, the YANG model includes specific Remote Procedure Calls (RPC), as well as agreement specific modules used for the agreement. In order to improve the flexibility, CAP is a three way exchange. The Initiator proposes a set of CA, the provider responds with a set of proposals. The reason for not selecting an agreed CA is to provide the opportunity for the provider to send alternate proposals that may not part of those of the Initiator. The proposals from the Initiator should be considered as expressing the intended CA, the response contains the corresponding proposals by the provider. The derivation of the response is policy driven. In case the responder does not want to proceed to a collaboration it can abort the negotiation. Finally, the Initiator confirms the agreed CA. In this case the agreed CA must be included in the provider's proposals. If none of the proposals is acceptable, the Initiator can also ends the negotiation. A CA proposal consists in a list of structures of type *context-proposed*, *service-description-proposed* and *guarantee-term-proposed*. Accepting a proposal consists in selecting one instance for each type. The same rule also applies recursively, in order to avoid the enumeration of the possibilities as well as to ease the establishment of policies. Once the CA has been agreed, it can be updated using the regular Netconf / RESTconf operations to update each parameter. Note that these parameters can be updated in both ways, which means Initiator and provider implement both a client and a server component.

## V. Proposed SSF collaboration framework

This section describes a framework to enable the collaboration among SSF instances from clouds and cloudlets. We assume an orchestrated architecture, meaning that security orchestrators from each domain involved in the collaboration are responsible for putting in place the CA established by the SSF Initiator and SSF Provider. The motivations for an orchestrated approach are as follows. First, some global understanding of the logic behind the service chains already deployed, as well as a global view of those chains, are required when defining where in the chain the Provider needs to be inserted and the potential need of instantiating an SSF for this task; such vision is only expected to be available to an orchestrator from the corresponding administrative domain. Second, setting the collaboration between domains requires some administrative privileges, which only need to be given to an orchestrator. Finally, having the different domains negotiate via their respective orchestrators avoids exposing the SSF instances and other internal information from one domain to the other, facilitating management tasks and reducing the system's attack surface.

In such orchestrated scenarios, we start by discussing intra- and inter-domain collaborations in Sec. V-A. These two types of collaborations are actually complementary, as the purpose of inter-domain collaboration is basically to supplement intra-domain collaborations whenever that domain does not have sufficient information and/or resources to handle an attack; in such cases, one domain may delegate tasks to another domain that agrees with such collaboration. We then give an overview of the inter-domain collaborative architecture in Section V-B, and describe in Section V-C how an inter-domain collaboration can be accomplished by configuring each domain. For a more focused discussion, we limit the scope of the intra-domain collaborations to a scenario based on the Service Function Chaining (SFC) architecture [21]. The reason is that, although alternatives exist (e.g., a purely SDN-based approach as done in [44]), SFC already provides many mechanisms that facilitate the construction and management of paths with different security services inside a domain.

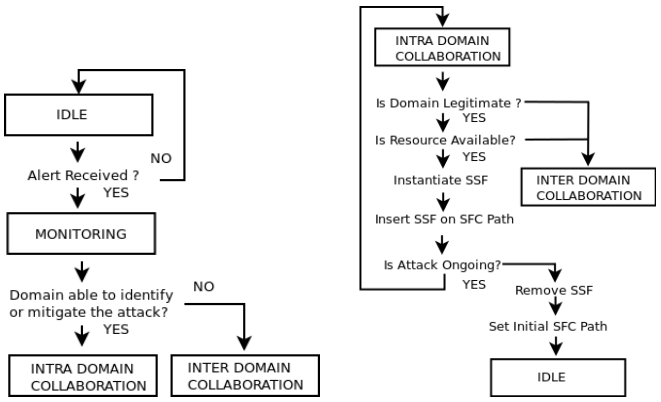
Along the discussion, we give special attention to how a best effort mode collaboration can be set between a Provider and Initiator, since this is a key novelty of the proposed architecture. More specifically, we describe different alternatives for inserting an SSF Provider into an SFC path in a best effort setting, which can be done by replacing the previous path or reconfiguring it. We also discuss how to set an alternate and a standard path between collaborators, showing how SFC can be used for this purpose inside each independent domain, and the impact of doing so among domains.

### A. Intra- and inter-domain collaboration relationship

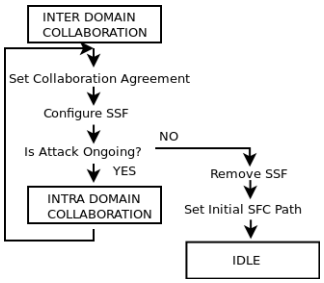
Fig. 2 depicts how intra- and inter-domain collaboration are expected to coexist. A domain that is not under attack has its security orchestrator in an idle state. As depicted in Fig. 2a, upon receiving an alert from one of the elements, the orchestrator analyzes the attack aiming to identify its characteristics and to evaluate whether or not the domain is capable of mitigating it [11]. If the domain can handle the attack on its own, it performs an intra-domain collaboration, while otherwise it goes into an inter-domain collaboration state. One possible approach for inter-domain collaboration is to request the service of a third-party scrubbing center specialized in this task; however, this usually involves a pre-agreed contract and high costs. Alternatively, the domain under attack may prefer to play an active role, collaborating with other domains to mitigate the attack to avoid the need of (or in complement to) resorting to a scrubbing center.

We assume that the domain is able to identify the attack as well as the necessary SSFs to mitigate it. After evaluating the presence of necessary conditions for a successful mitigation (e.g. sufficient resources, compliance with internal and external policies), the domain proceeds to the instantiation of an SSF in its own domain, updating the chain of on-path SSF accordingly. In this case (see Fig. 2b), no inter-domain collaboration is needed. Conversely, this task may instead be

exported to other domains for a distributed attack mitigation, in which case the domain that detected the attacks sets a collaboration with another domain as shown in Fig. 2c. In any case, when the attack subsides, the collaboration settings (e.g., additional SSFs instantiated and the network configurations employed for directing the traffic to them) can be removed, so the system goes back to its initial state.




(a) Idle to Collaborative State (b) Intra-collaborative State




(c) Inter-collaborative State

Fig. 2. Collaboration State Diagrams

## B. Inter-domain Collaboration

Fig. 3 illustrates the orchestrated architecture for two independent domains: a cloud and a cloudlet, each with its own orchestrator. Such inter-domain collaboration may be set using a RESTful service API implemented by an OpenDaylight control layer application [34], or any other SDN controller communication strategy based on westbound APIs [22]. For sake of simplicity, Fig. 3 does not represent all SSF nor other non-security SF in the chain, but only the SSF Initiator and Provider.

As illustrated in Fig. 3, suppose the Initiator is located in the cloud and sends its orchestrator a request for collaboration with a given SSF type. The collaboration request is triggered when the Initiator identifies, for example, that its own resources are nearly exhausted. The cloud orchestrator evaluates if the Provider should be instantiated inside its own domain or further downstream. Fig. 3 assumes the latter case, so a CA between cloud and cloudlet is created and: (1) an SSF instance from the cloudlet is selected as Provider; (2) the cloud and the cloudlet are interconnected with a standard and a alternate paths; and (3) the cloud proceeds to internal configurations for ensuring that only the traffic coming from the alternate path goes through the Initiator (which is further discussed in Sec. V-C).

The inter-domain alternate path and its corresponding intra-domain path inside the cloud and cloudlet are represented with dashed lines in Fig. 3. In practice, as discussed in Sec. IV-B, such interconnection between the cloud and the cloudlet will typically be implemented with different tunnels (e.g., VXLAN, GRE or IPsec). One of the tunnels is for the standard path, which conveys packets already treated by the Provider, while the other is for the alternate path, with untreated traffic; hence, at its entry point, the cloud can easily differentiate between the two types of traffic without relying on any internal information from the cloudlet's SFC configuration. In other words, each SFC is responsible for performing the appropriate binding between the inter-domain's standard (resp. alternate) path and the corresponding intra-domain's standard (resp. alternate) SFC path. In Fig. 3, for example, the final forwarder in the cloudlet is responsible for steering the traffic from the cloudlet's standard (resp. alternate) path to the inter-domain's standard (resp. alternate) path, interconnecting the regular (resp. dashed) lines depicted in this figure. Similarly, the *Classifier* in the cloud is responsible for steering the traffic from the inter-domain's standard and alternate paths to the appropriated SFC path inside the cloud. We note that this definition of an alternate path only involves the collaborating SSF instances, i.e., Initiator and Provider. Hence, if the traffic is steered through SFs other than the Initiator and the Provider, then the traffic from standard and alternate paths passes through them as usual.

During step (1), the cloudlet's orchestrator may decide to re-use an existing SSF instance or to create a new one, depending on the cloud's collaboration needs and on the cloudlet's currently available resources. If a new SSF instance needs to be created, it is not enough to have privileges to dynamically update the network configuration and the allocate resources. In fact, this action also requires an understanding of the chain of services applied to the traffic, as each class of SSF may have different requirements (as discussed in Sec. IV-A). For example, if the chain contains an encryption SSF as well as a firewall, the firewall is expected to be placed before the encryption occurs. The proposed solution does not address this issue directly, though, assuming that such decisions are made by the orchestrators of the each collaborating domain.

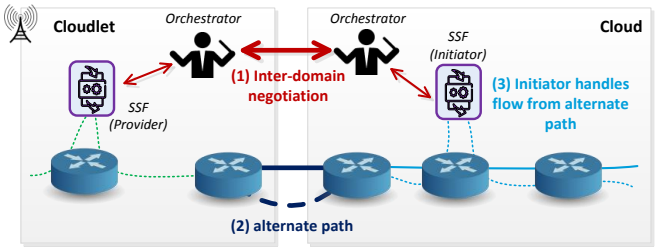



Fig. 3. Inter-domain orchestrated architecture: collaboration overview

### C. Intra-domain collaboration

As previously mentioned, in what follows we assume that the SSF in each domain are managed by the Service Function Chaining (SFC) architecture [21]. The resulting SFC-based architecture for steering traffic through various SSF instances within a domain is illustrated in Fig. 4. As shown in this figure, upon receiving some traffic, a so-called *Classifier* node selects the SFC path the traffic has to be steered to.

SFC uses a specific SFC encapsulation, named a Network Service Header (NSH) [42], to carry SFC-related information. Some examples of information conveyed by the NSH are the path's ID, the position of the packet within the SFC chain (also known as Service Index – SI), as well as potentially some metadata intended for specific SSF instances along the path. The packet is iteratively forwarded to the intended SSF by the so-called Service Function Forwarders (SFF), which determine the next SSF or forwarder according to the SFC ID and the position indicated in the NSH. When an SSF receives a packet, besides processing it accordingly (e.g., applying filtering rules, in the case of a firewall), it may also add some contextual information to the NSH. This is done by adding specific metadata intended for subsequent forwarders or SSF in the chain. After traversing the whole chain, but before leaving the domain, the final forwarder is responsible for handling the NSH packet, removing any contextual SFC information.

The SFC Control Plane enables the orchestrator to dynamically update the SFC architecture. For example, interface C1 is used by the orchestrator to steer the traffic to a different path. C2 is used to modify the next hop to which the traffic is steered or, on the final forwarder, to manage how to strip the NSH and forward the packet to another domain. Finally, C3 is employed to update the SSF itself.

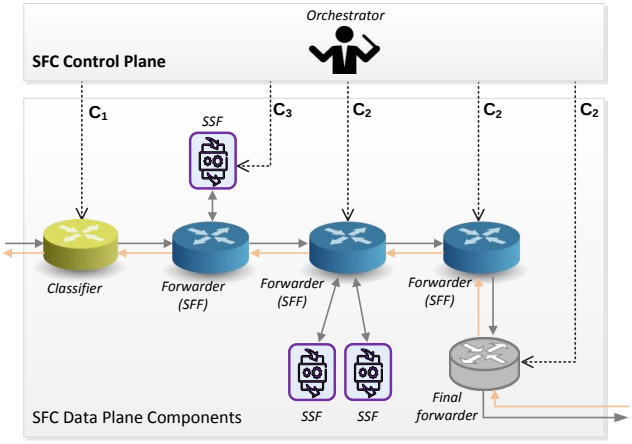
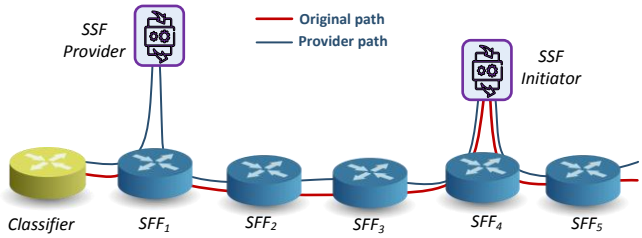


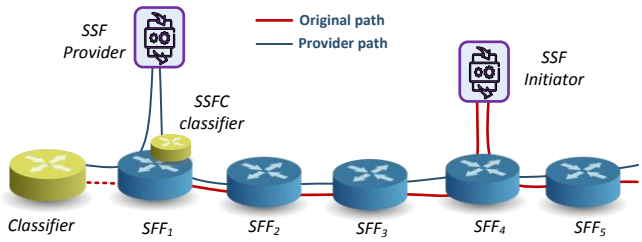

Fig. 4. Intra-domain orchestrated architecture

For enabling the collaboration to occur after the CA is agreed upon, the orchestrator on the cloudlet's side is expected to proceed to the following steps: (1) insert the (possibly newly instantiated) Provider into the existing SFC path; (2) set an internal signaling to distinguish the alternate path from the standard path, using NSH's context metadata; and (3) update the final forwarder so the outgoing packets are appropriately redirected to the inter-domain alternate path or the inter-domain standard path.

Fig. 5 illustrates different ways to insert a new SSF Provider into the SFC path within a domain. In Fig. 5a, a new SFC path (called a "Provider path") containing the Provider and all SSF and SF of the former SFC path – including the Initiator – is created; in this scenario, the Classifier uses this new path instead of the old one while the collaboration remains active. As an alternative, Fig. 5b illustrates a "branching" approach. In this case, a Provider path containing all SSF of the original path but the Initiator is created, and all traffic of the chain is directed to this new path by the chain's Classifier. The original path remains active, however, as a classifier embedded in the forwarder responsible for the SSF Provider ($SFF_1$ in Fig. 5b) steers packets from the Provider path to the original path whenever the corresponding traffic is not treated by the Provider, thus making the original path an "alternate path".




(a) Creating a new SFC Provider Path




(b) SFC Provider Branching

Fig. 5. Different approaches for adding an SSF Provider into an SFC path

The first approach is the simplest, as it is uses the SFC architecture in a quite straightforward manner, requiring only that the collaboration mechanism is provided sufficient privileges to create a new SFC path; one disadvantage, though, is that in this case the Initiator still receives all traffic and needs to evaluate whether or not it has been previously treated. In comparison, the second approach is expected to reduce the traffic going through the Initiator, making it a more efficient strategy. In all cases, the Provider needs to indicate whether the traffic has or not been treated, thus allowing the packets to be redirected to the "alternate path". This is done by setting some specific metadata embedded into the NSH header. In the first approach, this information is intended for the Initiator, which can act accordingly, treating or simply forwarding the packet. In the second, the information is intended to the classifier embedded into the forwarder that is responsible for steering

the traffic coming from the Provider to the appropriated path. Note that, with the first approach, the alternate and standard paths are multiplexed into the same SFC path, whereas in the second the alternate and standard paths are segregated.

The cloud's orchestrator, in its turn, is expected to do the following as a result of agreeing to the CA: (1) update its classifier to take into account the inter-domain alternate path and the inter-domain standard path; and (2) internally handle the alternate and standard traffic.

Updating the classifier in this case consists basically of configuring it to handle the two different tunnels for treated and untreated traffic. To distinguish between them, it is recommended to have two distinct paths, with different IDs. The reason is that, although the cloudlet needs to steer all inbound traffic to the Provider, so it can decide whether or not to treat it, the cloud does not have to steer all the traffic to the Initiator. The creation of two SFC paths avoids, thus, the need of conveying some alternate path information in the form of SFC metadata to the Initiator, which would then have to read and analyze this data, leading to an unnecessary overhead.

## VI. Experimental results with OpenDaylight

To validate the feasibility and efficiency of the proposed framework, we built an experimental setup that emulates the insertion of SSF instances in a chain, and then measured the benefits of the SSFs' collaboration. For this, we used the SFC OpenDaylight (ODL) project in its Beryllium release [39], which provides the infrastructure needed for the ODL controller [27] to provision and control a service chain.

The resulting testbed, illustrated in Fig. 6, emulates the behavior of a service network composed of two network domains, representing a cloudlet and a cloud. Such domains are implemented in two physical servers, both displaying the following specification: Intel Xeon E5-2430 processor (15M Cache, 2.20 GHz), 48GB (6x8GB) of DDR3 RAM memory, and Intel Gigabit ET2 Quad Port Server, running Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-24-generic x86_64). Each server runs VirtualBox [40] with a similar set of VMs: an SFC controller (`odl`) that acts as an orchestrator for the SFC infrastructure; two SSF, both able to emulate Deep Packet Inspection services by performing some CPU-intensive task, besides handling the packets' NSH so they can be correctly treated by the underlying forwarders; and four forwarders (numbered `sff1` to `sff4`), the first acting as a classifier at the beginning of the chain, two forwarding the traffic to the SSF in the middle of the chain, and the last one handling the standard and alternate paths at the chain's end, using different virtual network interfaces (VNI) for connecting to the cloud's Classifier. This simple scenario corresponds to a statically configured collaboration agreement between cloud and cloudlet, which may result, for example, in the instantiation of a new SSF on the latter's domain for running some task (in Fig. 6, this is the case for `ssf1` on the cloudlet's side).

To generate traffic for our experiments, we use a JMeter client in the cloudlet to send increasing amounts of HTTP requests toward a web server in the cloud. In all experiments, we assume the branching approach depicted in Fig. 5b.

### A. Adding a new SSF to the service chain

To evaluate the overhead introduced by the path modification mechanisms when using the branching approach, we measured the time necessary for the insertion of a new SSF into the chain (namely, `ssf1`, as shown in Fig. 6). Even though many possible deployment scenarios are possible, we consider the time of insertion for a new SSF to be the base level for the insertion overhead and therefore a good indication of the overall overhead for insertions in our framework. This task was separated in two steps: (1) the time taken for instantiating a VM image; and (2) the time taken for creating a new SFC chain to include that SSF after it is already instantiated. The result is that the first and second steps take, respectively, 64.9 $\pm$ 0.3 and 69.5 $\pm$ 0.1 seconds. In practice, however, the first step would be executed only if the VM was instantiated on demand after the negotiation between orchestrators is finished; hence, such overhead could be neglected with the preemptive instantiation of a suitable VM as part of the collaboration protocol. Even in the worst case scenario, when VMs need to be instantiated on demand, the overhead is less than two minutes, which is reasonable when compared to the hours-long lifespan of typical DDoS attacks, as indicated in security reports provided by Akamai and others [1], [2], [33], [41]. The path modification overhead can also be mitigated through the preventive creation of alternate paths inside cloudlet and cloud domains. The creation of preventive paths may consider DDoS occurrence and collaboration records from multiple domains, or even remain from previous collaboration schemes. Preventive paths and corresponding VMs may remain idle until security incidents are detected, when the preventive path becomes active and start to operate as alternate path. In this case, the path creation overhead will be reduced to the time necessary for waking up the corresponding VMs plus the time necessary to modify the classification rules inside the participant domains. The time necessary to activate the preventive path will vary according to the number and configuration of the Providers, and can be of the order of 10s for each VM [24]. Other alternative is to maintain the preventive path active even before a security incident to be detected, reducing the path creation overhead to the time for modifying the classification rules. From the data plan perspective, this time is of the order of 1ms [43]. Alternatively, container-based virtualization can be applied to reduce the instantiation time of SSFs and SFFs, with associated implications on the isolation and the manageability of these elements [52].

### B. Measuring SFC's latency overhead

In a second experiment, we evaluated the average overhead introduced by the adoption of the SFC itself as basis for an orchestrated implementation of the proposed framework. More precisely, we measured how much latency is added by the presence of forwarders along the chain, in comparison with having a direct connection between the SSF. For this

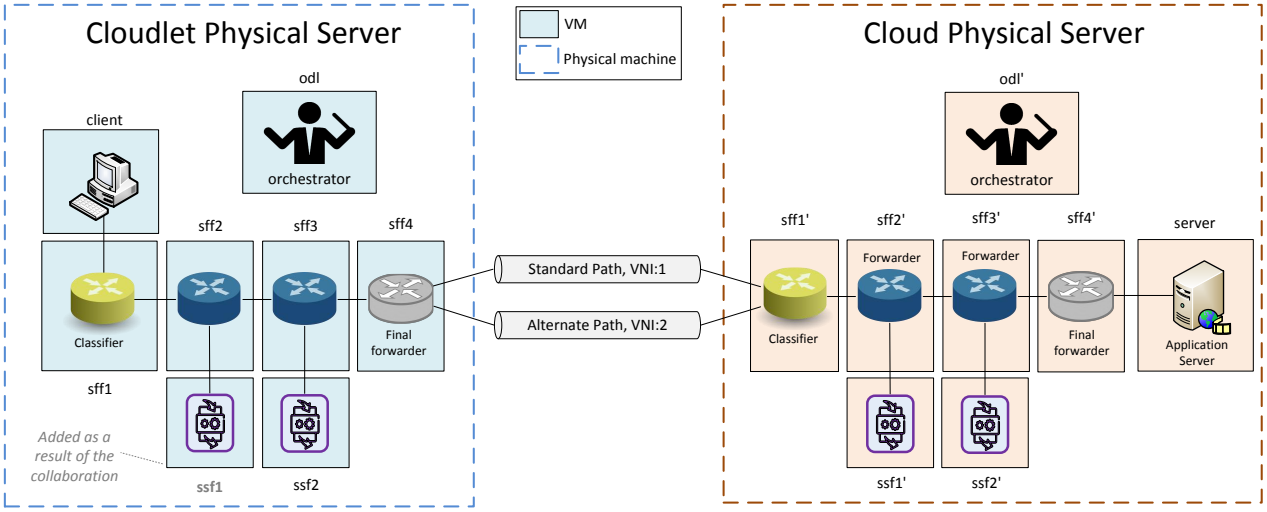



Fig. 6. Overall SFC testbed employed in our experiments: chains from two domains connected via two VNIs.

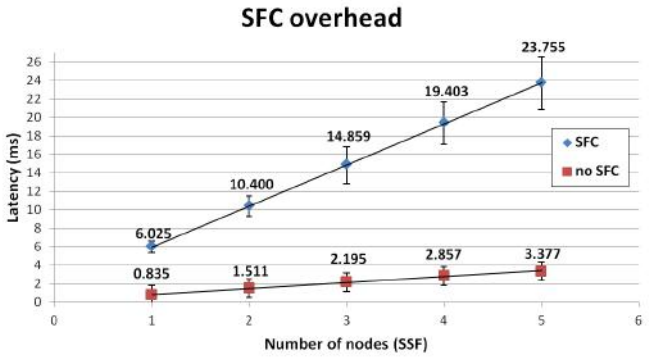



Fig. 7. Latency incurred by SFC architecture, with one forwarder per SSF.

task, we employed the same SSF provided by default in the OpenDaylight SFC demo used as basis for the testbed, which do not perform any actual service, but simply forwards all traffic received after evaluating and updating the packets' headers (i.e., the DPI-like task is deactivated). We then connected the network interfaces of those SSF (1) via forwarders, one per SSF, analogously to the scenario shown in Fig. 6, or (2) directly, so no overhead would result from the SFC architecture itself. Fig. 7 depicts the results obtained considering a total number of SSF ranging from 1 to 5, showing a linear growth that is also observable with addition SSF; the numbers shown correspond to 1000 repetitions of the tests. The analysis of this figure reveals that each SSF and corresponding forwarder contribute to an average latency of approximately 4.5 ms, which means a total latency overhead of approximately 3.8 ms per SSF compared to the scenario where the SSF are connected directly.

### C. Evaluating the benefits of best-effort collaboration

As a final experiment, we analyzed the best-effort collaboration approach, aiming to identify how it affects the communication's QoS in terms of latency, jitter and packet drop rate. To avoid perturbations from other SSFs, which could add noise to our measurements, we removed `ssf1` and `ssf2` from our testbed's chain and left only `ssf1'` and `ssf2'` as the collaborations' Provider and Initiator, respectively. Both were then pre-configured (1) to perform a CPU-intensive task on the traffic coming from the cloudlet, thus emulating a DPI service for each packet it decides to treat, and (2) with different collaboration percentages, to represent distinct mitigation scenarios. For example, for a 30% collaboration percentage, the Provider treats 30% of the incoming packets and sends them through the Provider path (which does not include the Initiator), whereas the other 70% of packets are simply reclassified and forwarded to the Original path to be treated by the Initiator; this applies only to the traffic coming from the cloudlet, as the cloud's response packets are simply forwarded back to the client without any additional treatment. We then use JMeter to send an increasing amounts of traffic to the cloud's web server, following a linear growth rate.

Fig. 8 depicts the percentage of packets that are dropped in this experiment for similar traffic loads, both in terms of total number of packets and of growth rate. As observed, collaboration provides some interesting optimization opportunities: if the Provider is configured to treat packets until it starts being unable to do so without dropping packets, the network can considerably reduce the packet drop ratio. Specifically in the case of our experiment, a direct result of collaboration is that the drop ratio goes from 31% the original scenario, in which the Initiator was handling all traffic, to 2.7% when 40% of the traffic is treated by the Provider. When the Provider handles more than 40% of traffic, it becomes itself overloaded, and starts dropping packets. In the worst case, when the Provider handles 100% of the traffic on behalf of the Initiator, the former ends up dropping approximately as many packets as before the collaboration was set (namely, 27%), showing that fully offloading the Initiator's tasks is far from being an optimal solution. Therefore, we can conclude that the proposed

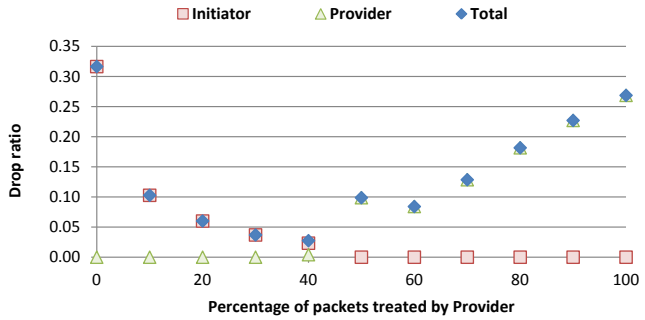



Fig. 8. Packet drop ratio for different collaboration percentages.

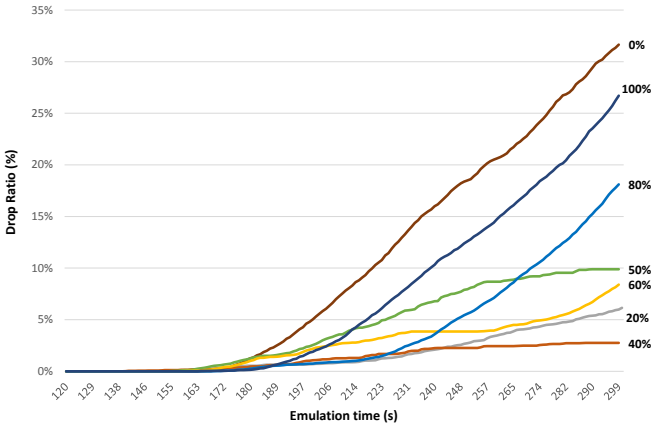



Fig. 9. Packet drop ratio curves during execution time, as traffic increases, for different percentages of traffic being handled by the Provider.

framework successfully extends the overall security resources to the sum of available resources in the SSFs participating in the collaboration.

We also assess the behavior of the packet dropping ratio rate over time (i.e., as traffic grows). This is shown in Fig. 9. For better visibility, this figure only includes some selected collaboration percentages (namely, 0% to 100% in steps of 20%, besides the 50% inflection point), and starts at 120 seconds of simulation time (since, before that, no packet is dropped for any collaboration percentage). The resulting graphs once again indicate that, with a suitable configuration for the best-effort collaboration (optimally, at 40% in our setup), packet dropping occurs after a longer period of time. This means that the traffic loss happens for much higher traffic, i.e., that the framework can significantly improve the system's resilience against packets losses during traffic bursts.

We also note that collaboration approaches placing a higher load on the Provider tend to be less effective than those concentrating the burden on the Initiator. One probable reason is that even a small amount of packet treatment by the Provider serves to alleviate the burden on the Initiator, as the latter does not receive some packets. In comparison, the Provider always receives the full load of packets, independently of the subsequent treatment by the Initiator, so the overall result of overloading it is more dropped packets and less efficiency.

Another benefit of best-effort collaboration refers to latency

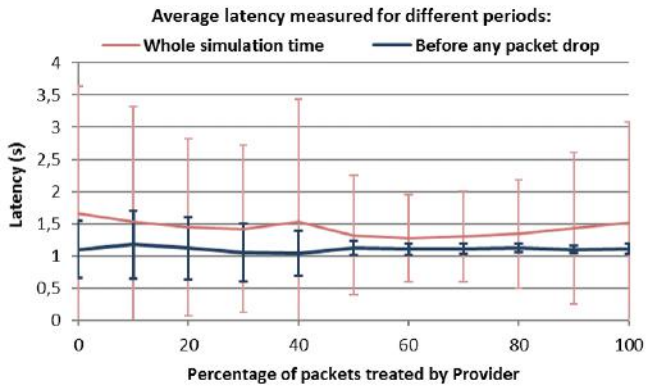



Fig. 10. Latency and jitter for different collaboration percentages.

and jitter, which are shown in Fig. 10. This figure compares the average latency considering the whole experiment with the one obtained before any packet is dropped, for different collaboration settings. It shows that the latency is on average 30% higher when we consider the period of time in which the network is overloaded and packets keep being dropped, while its standard deviation (and, thus, the network's jitter) is about 7 times in this situation. These results indicate that, since the proposed framework delays such overload scenario, it also delays a rise in packet latency and jitter, making the system more resilient to traffic bursts. It is also interesting to notice that placing a higher burden on the Provider potentially leads to a lower jitter when the traffic is still under control (i.e., while no packets are being lost). The reason is that, in this case, a lower amount of traffic passes through both SSFs and, thus, the packets are on average less prone to uncertainties caused by queuing on those elements.

## VII. Conclusions

In SECaaS (SECurity as a Service) systems, the effective collaboration among security service functions (SSF) is an important feature for scalability. In this paper, we present a flexible approach for enabling collaboration among multiple administrative domains. The proposed collaboration approach considers that the SSF can negotiate not only the tasks to be offloaded among them, but also the duration of the collaboration and the amount of resources to be dedicated on each task, thus enabling a "best-effort" cooperation mode.

The experiments based on the Service Function Chaining (SFC) architecture and OpenDaylight show that the proposed approach is feasible, incurs a low overhead, and enables an extension of the available resources for the security mechanisms to the sum of the resources available on the collaborating SSF instances. As future work, we plan to extend our current implementation to include different services, including packet filtering, and also a larger number of SSF to evaluate scalability issues.

**Acknowledgment:** This work was supported by the Inovation Center, Ericsson Telecomunicações S.A, Brazil, and also by the Brazilian National Council for Scientific and Technological Development (CNPq) under grants 473916/2013-4 and 305350/2013-7.

## REFERENCES


[1] Akamai. State of the internet / security – q1 2016 executive review. Technical report, Akamai, 2016. www.akamai.com/StateOfTheInternet.

[2] Akamai. State of the internet / security q3 2016 report. Technical report, Akamai, 2016. www.akamai.com/StateOfTheInternet.

[3] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web services agreement specification (WS-Agreement). In *Open Grid Forum*, volume 128, page 216, 2007.

[4] A. Bierman, M. Bjorklund, and K. Watsen. RESTCONF Protocol. Internet-Draft draft-ietf-netconf-restconf-18, Internet Engineering Task Force, Oct. 2016. Work in Progress.

[5] M. Bjorklund. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020 (Proposed Standard), Oct. 2010.

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things, 2012.

[7] F. Chen, B. Bruhadeshwar, and A. X. Liu. A cross-domain privacy-preserving protocol for cooperative firewall optimization. In *INFOCOM, 2011 Proceedings IEEE*, pages 2903–2911, 2011.

[8] X. Chen, B. Mu, and Z. Chen. NetSecu: A collaborative network security platform for in-network security. In *3rd International Conference on Communications and Mobile Computing (CMC)*, pages 59–64. IEEE, 2011.

[9] S. Chisholm and H. Trevino. NETCONF Event Notifications. RFC 5277 (Proposed Standard), July 2008.

[10] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke. From open grid services infrastructure to ws-resource framework: Refactoring & evolution. Technical report, Fujitsu, IBM and The University of Chicago, March 2004. Available: http://toolkit.globus.org/wsrf/specs/ogsi_to_wsrf_1.0.pdf.

[11] R. Danyliw and T. Gondrom. DDoS open threat signaling (DOTS). datatracker.ietf.org/wg/dots/charter/, 2015.

[12] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman. Network Configuration Protocol (NETCONF). RFC 6241 (Proposed Standard), June 2011. Updated by RFC 7803.

[13] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. Generic routing encapsulation (GRE) – (RFC 2784). https://tools.ietf.org/html/rfc2784, 2000.

[14] A. Farrel and L. Dunbar. Interface to network security functions. datatracker.ietf.org/wg/i2nsf/charter/, 2015.

[15] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[16] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siedbenlist, R. Subramaniam, J. Treadwell, and J. Von Reich. The Open Grid Services Architecture, version 1.5. https://www.ogf.org/documents/GFD.80.pdf, July 2006.

[17] A. Furfaro, A. Garro, and A. Tundis. Towards Security as a Service (SecaaS): On the modeling of security services for cloud computing. In *International Carnahan Conference on Security Technology (ICCST'14)*, pages 1–6, 2014.

[18] D. Gannon, K. Chiu, M. Govindaraju, and A. Slominski. An analysis of the open grid services architecture. Technical report, UK E-Sciences Core Program, 2002. Available: http://www.nesc.ac.uk/esi/events/gog2/OGSAanalysis3.pdf.

[19] A. S. Grimshaw, M. A. Humphrey, and A. Natrajan. A philosophical and technical comparison of Legion and Globus. *j-IBM-JRD*, 48(2):233–254, 2004.

[20] A. S. Grimshaw, A. Natrajan, M. A. Humphrey, M. J. Lewis, A. Nguyen-Tuong, J. F. Karpovich, M. M. Morgan, and A. J. Ferrari. From legion to avaki: The persistence of vision. *Grid Computing: making the global infrastructure a reality*, pages 265–298, 2003.

[21] J. Halpern and C. Pignataro. Service function chaining (SFC) architecture (RFC 7665). https://tools.ietf.org/html/rfc7665, oct 2015.

[22] M. Jarschel, T. Zinner, T. Hossfeld, P. Tran-Gia, and W. Kellerer. Interfaces, attributes, and use cases: A compass for SDN. *IEEE Communications Magazine*, 52(6):210–217, June 2014.

[23] S. Kent and K. Seo. Security architecture for the Internet Protocol (RFC 4301). https://tools.ietf.org/html/rfc4301, 2005.

[24] T. Knauth and C. Fetzer. Fast virtual machine resume for agile cloud services. In *2013 International Conference on Cloud and Green Computing*, pages 127–134, Sept 2013.

[25] S. B. Lee, M. S. Kang, and V. D. Gligor. CoDef: collaborative defense against large-scale link-flooding attacks. In *Proc. of the 9th ACM conference on Emerging networking experiments and technologies*, pages 417–428. ACM, 2013.

[26] J. Li, S. Berg, M. Zhang, P. Reiher, and T. Wei. Drawbridge: Software-defined DDoS-resistant traffic engineering. *SIGCOMM Comput. Commun. Rev.*, 44(4):591–592, 2014.

[27] Linux Foundation. OpenDaylight, a Linux Foundation Collaborative Project, 2015. http://www.opendaylight.org/. Accessed: 2016-05-01.

[28] H. Ludwig. WS-Agreement concepts and use–agreement-based Service-Oriented Architectures. *IBM Research Division, Technical Report*, 2006.

[29] H. Ludwig, T. Nakata, O. Wäldrich, P. Wieder, and W. Ziegler. Reliable orchestration of resources using WS-Agreement. In *International Conference on High Performance Computing and Communications*, pages 753–762. Springer, 2006.

[30] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *ACM SIGCOMM Computer Communication Review*, 32(3):62–73, 2002.

[31] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. VXLAN: A framework for overlaying virtualized layer 2 networks over layer 3 networks (RFC 7348). www.ietf.org/id/draft-mahalingam-dutt-dcops-vxlan-06.txt, 2013.

[32] P. Marques, N. Sheth, R. Raszuk, B. Greene, J. Mauch, and D. McPherson. Dissemination of flow specification rules (RFC 5575). tools.ietf.org/html/rfc5575, aug 2009.

[33] T. Matthews. Incapsula survey: What DDoS attacks really cost businesses. Technical report, Incapsula, 2014. http://lp.incapsula.com/ddos-impact-report.html.

[34] J. Medved, R. Varga, A. Tkacik, and K. Gray. OpenDaylight: Towards a model-driven SDN controller architecture. In *IEEE 15th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–6. IEEE, 2014. https://www.opendaylight.org/.

[35] G. Meng, Y. Liu, J. Zhang, A. Pokluda, and R. Boutaba. Collaborative security: A survey and taxonomy. *ACM Comput. Surv.*, 48(1):1:1–1:42, July 2015.

[36] D. Migault and A. Ranjbar. Collaboration Agreement for Security Service Function. Internet-Draft draft-mglt-i2nsf-ssf-collaboration-00, Internet Engineering Task Force, June 2016. Work in Progress.

[37] Y. H. Milan Patel, P. Hédé, J. Joubert, C. Thornton, B. Naughton, J. R. Ramos, C. Chan, V. Young, S. J. Tan, D. Lynch, N. Sprecher, T. Musiol, C. Manzanares, U. Rauschenbach, S. Abeta, L. Chen, K. Shimizu, A. Neal, P. Cosimini, A. Pollard, and G. Klas. Mobile-Edge Computing Introductory Technical White Paper, sep 2014.

[38] G. Oikonomou, J. Mirkovic, P. Reiher, and M. Robinson. A framework for a collaborative DDoS defense. In *22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 33–42. IEEE, Dec 2006.

[39] OpenDaylight. Service Function Chaining, The OpenDaylight Project. https://wiki.opendaylight.org/view/Service_Function_Chaining:Main. Accessed: 2016-05-01, 2016.

[40] Oracle. VirtualBox, 2016. https://www.virtualbox.org/. Accessed: 2016-05-01.

[41] J. Pescatore. DDoS attacks advancing and enduring: A SANS survey. Technical report, SANS, 2014. https://www.sans.org/reading-room/whitepapers/analyst/ddos-attacks-advancing-enduring-survey-34700.

[42] P. Quinn and U. Elzur. Network Service Header (Internet-Draft). https://tools.ietf.org/html/draft-ietf-sfc-nsh-04, 2016.

[43] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore. *OFLOPS: An Open Framework for OpenFlow Switch Evaluation*, pages 85–95. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[44] R. Sahay, G. Blanc, Z. Zhang, and H. Debar. Towards autonomic DDoS mitigation using software defined networking. In *NDSS Workshop on Security of Emerging Networking Technologies (SENT'2015)*. Internet society, 2015.

[45] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh. A cost-aware elasticity provisioning system for the cloud. In *31st International Conference on Distributed Computing Systems (ICDCS'11)*, pages 559–570, 2011.

[46] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proc. of the 20th ACM Conference on Computer and Communications Security (CCS13)*, November 2013.

[47] K. Skala, D. Davidovic, E. Afgan, I. Sovic, and Z. Sojat. Scalable Distributed Computing Hierarchy: Cloud, Fog and Dew Computing. *Open Journal of Cloud Computing (OJCC)*, 2(1):16–24, 2015.

[48] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Sandholm, D. Snelling, et al. Open grid services infrastructure (OGSI). Technical report, Global Grid Forum, 2009.

[49] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at Google with Borg. In *Proc. of the European Conference on Computer Systems (EuroSys)*, 2015.

[50] P. Vixie. Rate-limiting state. *ACM Queue Magazine*, 12(2):10:10–10:15, Feb. 2014.

[51] E. Wang, K. Leung, J. Felix, and J. Iyer. Service Function Chaining use cases for network security (Internet-Draft). tools.ietf.org/html/draft-wang-sfc-ns-use-cases-00, sep 2015.

[52] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. D. Rose. Performance evaluation of container-based virtualization for high performance computing environments. In *21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 233–240, Feb 2013.

[53] Q. Yan, F. R. Yu, Q. Gong, and J. Li. Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges. *IEEE Communications Surveys Tutorials*, 18(1):602–622, 2016.

[54] S. Yu, Y. Tian, S. Guo, and D. O. Wu. Can we beat DDoS attacks in clouds? *IEEE Trans. Parallel Distrib. Syst.*, 25(9):2245–2254, 2014.

[55] S. T. Zargar and J. Joshi. DiCoDefense: distributed collaborative defense against ddos flooding attacks. In *34th IEEE Symposium on Security and Privacy (S&P'13) (Poster)*, 2013.

[56] S. T. Zargar, J. Joshi, and D. Tipper. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE Communications Surveys Tutorials*, 15(4):2046–2069, 2013.

[57] S. Zhang, F. Ivancic, C. Lumezanu, Y. Yuan, A. Gupta, and S. Malik. An adaptable rule placement for software-defined networks. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 88–99. IEEE, 2014.