Journal of Network and Systems Management (2020) 28:953–989
https://doi.org/10.1007/s10922-020-09559-4

Blockchain Signaling System (BloSS): Cooperative Signaling of Distributed Denial-of-Service Attacks

Bruno Rodrigues[1] · Eder Scheid[1] · Christian Killer[1] · Muriel Franco[1] · Burkhard Stiller[1]

Received: 9 May 2020 / Revised: 13 July 2020 / Accepted: 28 July 2020 / Published online: 24 August 2020

Abstract
Distributed Denial-of-Service (DDoS) attacks are one of the major causes of concerns for communication service providers. When an attack is highly sophisticated and no countermeasures are available directly, sharing hardware and defense capabilities become a compelling alternative. Future network and service management can base its operations on equally distributed systems to neutralize highly distributed DDoS attacks. A cooperative defense allows for the combination of detection and mitigation capabilities, the reduction of overhead at a single point, and the blockage of malicious traffic near its source. Main challenges impairing the widespread deployment of existing cooperative defense are: (a) high complexity of operation and coordination, (b) need for trusted and secure communications, (c) lack of incentives for service providers to cooperate, and (d) determination on how operations of these systems are affected by different legislation, regions, and countries. The cooperative Blockchain Signaling System (*BloSS*) defines an effective and alternative solution for security management, especially cooperative defenses, by exploiting Blockchains (BC) and Software-Defined Networks (SDN) for sharing attack information, an exchange of incentives, and tracking of reputation in a fully distributed and automated fashion. Therefore, *BloSS* was prototyped and evaluated through a global experiment, without the burden to maintain, design, and develop special registries and gossip protocols.

Keywords Distributed Denial-of-Service · Security · Blockchain · Software-Defined Networks · Security Management

✉ Bruno Rodrigues
rodrigues@ifi.uzh.ch

Extended author information available on the last page of the article

1 Introduction

The technological evolution has built a digitally networked society, in which the Internet is an indispensable basis for interactions. As the number of connected devices (mobile and stationary) increases, the complexity of systems providing content for these devices and their communication network grew in a similar proportion in support of the rocketing volume of traffic [1]. As a consequence, complex distributed systems are subject to several types of failures and threats that can compromise critical infrastructures of societies [2].

Although a Distributed Denial-of-Service (DDoS) is a widely known attack type, it remains one of the significant causes of concerns for service providers [5]. As observed (*cf*. Fig. 1, left) the number of IoT (Internet-of-Things) devices is surpassing the number of non-IoT devices, e.g., mobile phones, laptops, or computers. IoT devices, ranging from small sensors to home gateways, are a main target of malicious software exploiting their vulnerabilities to infect thousands [1, 6]. This software, termed *malware*, contains malicious code using resources of its host system to perform undesirable or malicious activities [7].

1.1 Botnets

Within a DDoS context, once a device is infected by malware, labelled as a *Bot*, it runs software, whose resources are unintentionally used to execute commands. A Botnet defines a system composed out of a cluster of Bots controlled by at least one attacker. Botnets take advantage of lacking security of IoT devices and are the primary basis for large-scale attacks (*cf*. Fig. 1, right). While in 2018 a DDoS attack was peaking 1.7 TBit/s in traffic volume on GitHub servers, the frequency of DDoS attacks also increased more than 29.7% times between 2015 and 2016 [8]. However, the 52% reduction in the number of large-scale attacks from

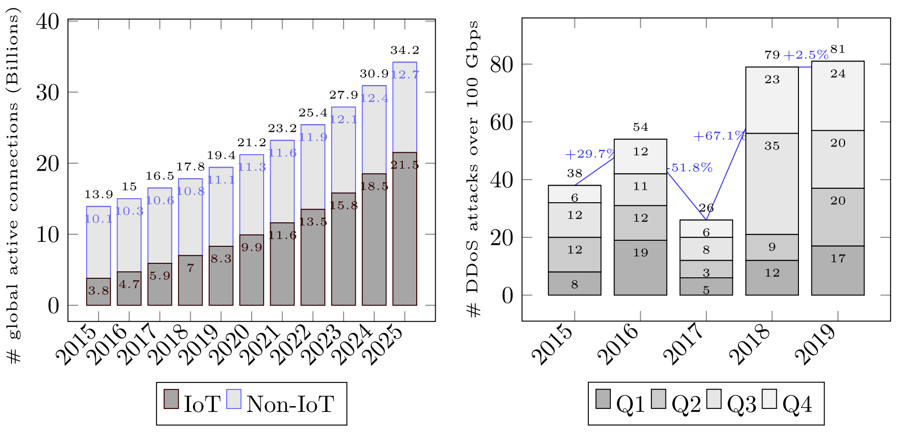

Fig. 1 Number of IoT-connected devices (per Year) (left), registered large-scale DDoS attacks (accumulated quarters/year) (right), based on [3, 4]

2016 to 2017 did not denote a reduction in attacks itself. Akamai reported that the majority of attacks ranged between 250 Mbit/s and 1.25 Gbit/s in traffic volume. However, in 2018 there was an alarming increase in the number of large-scale attacks by 67.1% due to different Mirai variations and reflection attacks based on Memcached [8].

The most prominent botnet example is the Mirai botnet, which exploits default and weak security credentials to take control of hosts and spreads itself to other devices. Mirai marked the transition toward an era of super attacks, in which the traffic volume at the target system often surpasses 1 TBit/s [1]. The first appearance of Mirai was in 2016, peaking 623 Gbit/s of traffic volume in an attack against Krebs Security. Th respective Web site's hosting company Akamai had to shut down this site, because the defense during three days became too costly. So many devices were used such that attackers did not have to use a sophisticated strategy. Iin October 2016 the attack on DynDNS peaked 1.2 TBit/s, resulting in the unavailability of significant Internet platforms and services, e.g., Twitter, GitHub, PayPal, and Spotify, due to the rendering of their Domain Name System (DNS) servers being unavailable [6].

1.2 Toward Cooperative Defenses

Taking these scenarios into account as well as the evolution of DDoS attacks, it is crucial that defense mechanisms have to evolve to become more powerful, too. For that, cooperative defenses for communication service providers—operators in short—have been proposed to reduce the overhead at a single point of mitigation. Although centralized defenses are effective in detecting and taking action against these attacks, in case of large-scale DDoS attacks, they are largely overwhelmed by the massive traffic. Hence, due to the highly distributed nature of such attacks, future network and service management, especially security management operations will benefit as follows:

1 Since DDoS attacks are largely distributed, security management approaches need to scale their defense capacity based on distributed approaches.
2 Operators can benefit from a cooperative decentralization, in which players coordinate themselves in pre-established trust alliances to exchange information required, as long as respective incentives exist to perform cooperative a DDoS attack mitigation.
3 Operations need to minimize hardware requirements such that cooperative defense mechanisms are based on state-of-the-art solutions, e.g., Software-defined Networking (SDN) and Network Function Virtualization (NFV), to manage network operations, security measures, and deploying service functions for the detection and mitigation of malicious traffic.
4 In addition, the use of Blockchains (BC) can increase trust among cooperative operators based on a transparent exchange of information, driven by incentives to deploy Virtual Network Functions (VNF) across SDNs.

Cooperative defenses allow for the combination of detection and mitigation capabilities of different domains, reduction of overhead at a single point, and blockage of malicious traffic near its source. However, since there is no widespread deployment of such a cooperative defense system yet main challenges [9, 10] remain: (a) the high complexity of operation and coordination, (b) the need for trusted and secure communications, (c) lack of incentives for the operators to cooperate, and (d) the understanding on how operations of these systems are affected by different legislation, regions, and countries.

1.3 Cooperative Blockchain Signaling System (BloSS)

The central goal of the cooperative Blockchain Signaling for DDoS (*BloSS*) is to provide a cooperative defense approach providing a technical answer for each of these categories combined into a single system. *BloSS* contributes to approaches and techniques to foster a collaborative defense between different Autonomous Systems (AS). while covering these four categories of challenges in an integrated manner. Furthermore, the analysis of legal, economic, and social requirements as well as the approach as an entire solution, embedded into a prototype, plays the key role to indicate the effectiveness of *BloSS* on a technical basis. Through an in-depth analysis, where related work provides data on functionality, performance dimensions (*cf.* Sect. 2), it is observed that none of these approaches adopted in practice cover these challenges.

BloSS relies on BCs and Smart Contracts (SC) [11]. Due to approaches, such as CoFence [12] and Bohatei [13], it is possible to enforce rules decentrally via BCs and SCs in a flexible and scalable fashion—at no additional hardware costs or restrictions to instantiate the cooperative defense. BCs do foster trusted cooperation, because they not only operate on the principle of decentralization, helping to eliminate third-party intermediaries, but they also provide incentives for stimulating the cooperative behavior among service providers. Similarly, BC capabilities allow for not only the mitigation of signaling requests, but also for an immutable platform for the exchange of mitigation services, where each participant can express their needs in forms of incentives.

While previous work by the authors presented the initial *BloSS* architecture of a collaborative defense based on BCs and SCs [14–16] being deployed as a local SDN cluster representing the underlying infrastructure (taking blacklisted address signaled and enforcing black-holing rules via OpenFlow), the extended *BloSS* design here includes mechanisms to ensure the provision of incentives for mitigation services via SCs, while now assessing the reputation of operators involved actively. In addition, an evaluation of the reputation system within BloSS was performed considering different behavior for each participant [17]. For instance, interactions were mapped based on honest, malicious, lazy, and selfish players (e.g., targets—the ones under attack—or mitigators—the ones offering collaborative mitigation). This assessment improved the final design of BloSS, including all on-chain steps for providing incentives and mapping reputation of members of such a collaborative defense, as well as front-end design presented in [18, 19].

Another aspect of cooperative defense is concerned with the impact of the size of blacklisted address lists on latency for signaling [20]. This work verified the benefit of the establishment of an off-chain P2P network based on IPFS (Inter Planetary File System) [21], in which nodes are exclusive participants in the collaborative defense as for BloSS. In this study the impact of confidentiality of data being exchanged off-chain was highlighted, impacting the design for a key exchange in the overall architecture (*cf*. security in Sect. 4.6).

This step is complemented in this work by the evaluation of performance aspects in a world-wide experiment, such as information propagation latency and protocol execution time. Therefore, in combination with the tracking and evaluation of reputation in a collaborative defense [17] and an off-chain transfer of blacklisted addresses [20] *BloSS* has reached herewith a practical and deployable approach for security management, especially in terms of DDoS mitigation in a cooperative, distributed manner.

The remainder of this article is organized as follows. While Sect. 2 presents background and related work, Sect. 3 outlines the assumptions and requirements taken into consideration. Section 4 describes the cooperative logic implemented in SCs and the decentralized application based on SDNs. While evaluations are contained in Sects. 5, 6 draws conclusions.

2 Related Work

As a response to the increasing number of DDoS attacks, research sees an increasing number of proposals to counter DDoS attacks based on both centralized and distributed (cooperative) perspectives. While centralized proposals target the optimization of detection and mitigation processes at a single domain, cooperative proposals broaden this scope by including mechanisms and protocols to perform the signaling of attacks between two or more domains, i.e., a gossip-based protocol and an architecture supporting its functioning.

As identified in [9, 10], main challenges of existing approaches are identified as: (a) the high complexity of operation and coordination; (b) the need for trusted and secure communication; (c) a lack of incentives for service providers to cooperate; and (d) the understanding on how operations of these systems are affected by different legislation, regions, and countries. They are categorized:

– Technical: The Internet is a heterogeneous environment whose underlying infrastructure is composed of many different protocols, systems, and networking equipment. The challenge is to abstract hardware/software differences of the underlying infrastructure or operate based on existing standards, avoiding to impose additional software or hardware requirements.
– Social: The public image of a service provider is often its most valuable asset. Thus, all the communication of such defense also needs a trusted channel to make sure that the attack information provided to all members is not only reliable, but also private to its members. Furthermore, trust needs to be established and reputation needs to be managed.

- Economic: Solely relying on voluntary contributions creates a favorable environment for free-riding (consuming resources without contributing). Incentives among the participating members need to be provided. Costs are in the form of CAPital EXpenditures (CAPEX) to configure and maintain the communication infrastructure as well as OPErating EXpenditures (OPEX) to cover resource utilization costs for the actual attack mitigation.
- Legal: It is necessary to understand and react upon the differences in the legal aspects of each region or country, which can influence the cooperation among members. For example, for legal reasons a member may be prevented from blocking traffic of a suspected host.

Secure Overlay Services (SOS) [22], COSSACK [23], and DefCOM [24] paved the way for cooperative defenses in the early 2000s. While SOS focused on identifying legitimate sources for time-sensitive networks (i.e., requiring peers to authenticate to the overlay network), COSSACK and DefCOM based their approach on detection and enforcement points in access networks. However, these approaches required changes in routers [23, 24] or required sources to be registered [22], thus, showing complexity of coordination and operation.

Another typical, although non-cooperative approach is to use cloud-based protection services. These serve as a proxy receiving, analyzing, and redirecting traffic to the target, which delegate detection and mitigation tasks to the protection provider (e.g., Akamai [1] or CloudFlare [25]). However, despite having dedicated resources to mitigate DDoS attacks and relying on incentives to perform this service, these are still centralized approaches and, therefore, vulnerable to large-scale attacks as observed in the DynDNS attack [26].

Guangsen and Manish [27] applies a gossip-based communication protocol to exchange attack information between independent detection points to aggregate information about observed attacks. The system is built as a peer-to-peer overlay network to disseminate attack information rapidly to other listening users or systems. A similar approach as of [28] formalizes a gossip-based protocol to exchange information in an overlay network using intermediate network routers. Also, [29] deploys a similar architecture, but uses an advertising protocol based on the FLEX (FLow-based Event eXchange) format, which is used to simplify the integration and deployment of the solution and facilitates communication process between domains involved.

More recent approaches, such as CoFence [12] and Bohatei [13], are based on relatively new technologies. For example, NFV and SDN can reduce the complexity of coordination and operation and based on essentially software approaches, decouple specific functions previously performed in hardware to a central point with a global view of the network as well as virtualize specific functions to be executed on servers. As a result, there is greater flexibility in the deployment and operation of these solutions, typically including the performance trade-off. Thus, the greater the level of abstraction or the generalization of the network function, the greater the loss of performance. In case of SDN, there is an issue of latency between the decision taken at the controller and the switches, and in case of VNF, the bottleneck can become the processing capacity of the server that typically accumulates various functions. Similarly, [30] proposes a collaborative framework that allows the customers to request DDoS mitigation from ASes. The proposal is based on an SDN controller implemented at customer side interfaced with the AS, which can change the label of the anomalous traffic and redirect them to security middle-boxes.

The implementation of Bohatei, for example, does not directly incorporate interdomain DDoS defense, which employing a scheme such as Pushback [31], would allow edge and access routers to relay traffic filtering to routers further upstream. Thus, expanding the approach for a multi-domain cooperative defense with the flexibility of SDN and NFV. Pushback relies on a Aggregate-based Congestion Control (ACC) concept, in which ACC imposes a traffic shaping on subsets of traffic (i.e., aggregations) defined by some characteristics such as specific destination port or source Internet Protocol (IP) address. It is a router-based solution that allows a router to request adjacent upstream routers to rate-limit the specified aggregates, and prevents upstream bandwidth (i.e., outbound traffic) from being wasted on packets that are only going to be dropped downstream.

Further, many proposals for a cooperative defense are not only limited to the signaling of attack information, offering a complete framework including attack detection, signaling, and mitigation. On the one hand, a complete solution has a positive side by offering the entire defense framework not only for the single domain but also in a cooperative fashion. On the other hand, this imposes hardware and software requirements that may restrict the widespread adoption of the solution. As an example, the IETF DOTS [32] proposal has a complex architecture that can make its widespread adoption an issue. However, it also has a high power of standardization, which can facilitate the adoption of the Protocol in standard networking hardware.

Related work (*cf.* Table 1) addresses a collaborative defense schemes facilitating communications among peers. This can be achieved either by using a novel technology (e.g., SDN and NFV) or a novel architecture, such as IETF DOTS [32] and DefCOM [24]. DOTS shows the major advantage of engaging the industrial community (through IETF) in forming a standard protocol for exchanging information about attacks. In this sense, the social aspect can be addressed in the creation of communities of mutually trustworthy entities, however, following a client-server model for the exchange of information.

However, challenges as of economic and social nature are not fully addressed. Thus, a technical solution based on BC has to avoid additional costs regarding hardware and software and needs to be simple to be deployed and operated. Therefore, *BloSS* encompasses the support for incentives based on BCs that can be safely and reliably distributed among participants and their legal/conformity options can be selected, too, e.g., restricting the operation to specific regions/countries or members.

Therefore, cooperative defenses can benefit from BCs in different dimensions. While BCs can (a) reduce the complexity of operations and coordination by using existing infrastructures to distribute rules without any specialized registries or protocols, they also can foster a (b) trusted cooperation due to their transparency and decentralized characteristics. Also, they can provide (c) financial incentives fostering the cooperative behavior among service providers [33]. Thus, BC capabilities can be leveraged for signaling mitigation requests across a BC network in a similar approach as for DefCOM [24] and BCs serve as an immutable platform

Table 1 Comparison of related work

Related work	Cooperative defense challenges				Capabilities
	Technical	Social	Economical	Legal	
DefCOM [24]	◐	✗	✗	◐	Signaling
SOS [22]	◐	✗	✗	◐	Signaling
COSSACK [23]	◐	✗	✗	◐	Signaling Mitigation
Zhang et al. [27]	◐	✗	✗	✗	Signaling
Pushback [31]	◐	✗	✗	◐	Signaling Mitigation
Steinberger et al. [29]	◐	✗	✗	◐	Signaling Mitigation
Sahay et al. [30]	●	✗	✗	◐	Signaling Mitigation
Velauthapillai et al. [28]	◐	✗	✗	◐	Signaling Mitigation
Bohatei [13]	●	✗	✗	◐	Signaling Mitigation
CoFence [12]	●	✗	✗	◐	Signaling Mitigation
IETF-DOTS [32]	◐	◐	✗	◐	Signaling
BloSS	●	●	●	●	Signaling

● = property provided; ◐ = property provided partially; ✗ = property not provided

for the exchange of mitigation services, where participants express their needs in forms of incentives.

3 BloSS Design Considerations

BloSS allows for the distribution of incentives to boost the cooperative behavior of participating entities and to track the reputation of operators involved. Thus, SCs are deployed in the underlying permissioned Ethereum [34] BC infrastructure [deploying a Proof-of-Authority (PoA) consensus mechanism] used to signal attacks over several domains, while at the same time managing incentives. E.g., a Computer Security Incident Response Team (CSIRT) within the same region is able to establish a private network, which is transparent only for its cooperative members (i.e., the CSIRT consortium). The consortium-based BC deployed provides trust by definition, i.e., assuming that shared information on an attacks signalled will not be exposed externally. However, trust is the key, as it involves sharing of data between CSIRTs [35]. Thus, a BC-based solution can provide through its natural transparency a higher degree of transparency and trust between collaborative instances (Table 2).

Table 2 Benefits and drawbacks of a BC-based collaborative platform in a cyber-security context

Dimension	Benefits	Drawbacks
Performance	Relatively simple to deploy and operate	Lack of performance in terms of transactions per second and storage capacity
Incentives	Platform to distribute incentives	Incentives may not be required by the CSIRT community
Trust	Enhancement of trust through full transparency and decentralization	Excess of transparency may impair confidentiality

3.1 Performance—Block Size and Delay

A limiting factor in a BC-based cooperative defense is performance, that involves the relation between the block size and the propagation delay, i.e., latency. The BC throughput is defined as [36]:

$$TP = \frac{blockSize}{propDelay} \tag{1}$$

Considering a *blockSize* of 1 MB and a *propDelay* of 10 s, the throughput would be of *t* 0.1 MB/s delivered in epochs defined by the block generation time. Similarly, a *blockSize* of 2 MB and a *propDelay* of 10 s would result in a twice better throughput *t* 0.2 MB/s. In addition, it should be noted that increasing block size can also negatively influence propagation time, the larger the file size being transferred, the greater the transfer time (*cf.* Fig. 2).

The effects of an increasing block size on the propagation delay is shown in Fig. 2. It has been shown a strong correlation between the size and its influence on

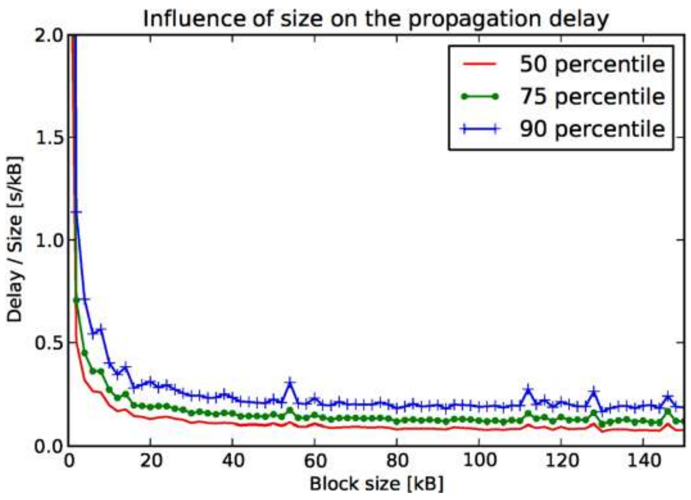

Fig. 2 Influence of block size on propagation delay [37]

the propagation time through the delay cost, which is defined by the authors as the time delay each kilobyte causes to the propagation of a transaction or block [37]. As reported, for sizes below 20 kB the round trip delay caused by the Protocol (Bitcoin used as example) causes a major influence on delay, whereas in blocks larger than 20 kB, each kilo byte represents an additional of 80 ms. The number of available blocks in a period of time can be determined by relation between the block generation time *blockGenTime* and the period of time *T*. Thus, considering a *blockGenTime* of 15 s based on the Ethereum, and a period of time in a day (in seconds) *T* = 86,400 s, the number of available blocks in a day is given by *T*/*blockGenTime*, resulting in this case in 5760 blocks. The maximum amount of information that can be Storage Available in the Period (SAP) is described by:

$$SAP = \frac{T}{blockGenTime} \times blockSize \tag{2}$$

In the case of a *blockSize* of 1 MB, the SAP considering a *blockGenTime* of 15 s would be 5.76 GB of storage available in a day, and 11.52 GB for a 2 MB *blockSize* for the same *blockGenTime*. Another negative aspect observed in a BC solution is the need for storage of BC history. This is to ensure that data previously entered into the BC is verified by all members, and also to prevent modifications being made for any malicious purpose. Alternatively, it is possible to define times (e.g., monthly or yearly), in which organizations of the alliance can save the BC state into a snapshot (which needs to be hashed and compared by all members) and start a new fork of the same BC [38]. However, such an approach does not reduce by itself the need for storage, requiring snapshot compression and storage at a more efficient media.

3.2 Financial Incentives

The CSIRT community currently shares information based on trusted contacts [39], i.e., the exchange of information between CSIRTs works at the confidence level that is defined by each CSIRT's individual relationship with other members in a region, group, or alliance. Currently, no exchange of financial incentives exists for shared information or cooperative mitigation actions.

The use of BC as a collaborative platform allows the creation of a marketplace to exchange mitigation services, financially rewarding actors involved in the mitigation of requests. While, to the best of the authors knowledge, there is no financial reward for these services (nor their needs explicitly evidenced by CSIRTs), there are proposals to formalize an incentive model associated with sharing, analyzing and delivering cybersecurity services [40]. There are arguments both for and against the use of financial incentives [39, 41]:

- Against: changing the current model of how information is shared could impact the trust model among CSIRTs.
- Favor: incentives could allow for greater engagement of collaborative organizations, working similarly to a bug bounty program.

The lack of financial incentives could discourage cooperation, which involves the use of resources and possible legal consequences of future mitigation acts in cases of false positives. The argument in favor is “a lot to lose and little to gain” [39] in effecting collaborative mitigation, and incentives are required to increase engagement. Naturally, by requiring resources from third parties, financial incentives are the most effective way to cover these costs.

3.3 Blockchain as an Enabler of Trust

Trust is the fundamental aspect of any cooperative environment and difficult to obtain, since it may rely on many non-technical aspects [42]. Also, the process of building trust between entities has no relation to a specific technology and several non-technical and specific aspects of each organization are required. BCs operate as a “trust-enabler”, providing transparency and trust between cooperative organizations. However, it is not possible to quantify the role of BCs as a trust enabler, since it is not possible to determine a “probability” in which the use of BC is a determining factor in ensuring trust between organizations. The role of BCs in building trust has been studied by [43], in which solutions are addressed on how these conflicting notions may be solved, while exploring the potential of BCs for dissolving the trust problem. According to [44, 45], the main characteristics of trust are defined as:

- Dynamic: as it applies only in a given time period and maybe change as time goes by. For example, a history of security data sharing between two or more companies does not guarantee that these companies will always share data at any time. Trust can only be built during a time-frame.
- Context-dependent: the degree of trust on different contexts is significantly different. E.g., organization A may share threat indicators, but may not disclose actual malware intelligence due to, e.g., legal issues. Thus, trust may exist between organizations A and B only for sharing a “threat indicators” context.
- Non-transitive: if A trusts B and B trusts C, A may not trust C. However, A may trust any organization that B trusts in a given context.
- Asymmetric: trust is a non-mutual reciprocal in nature. That means if entity A trusts B, the statement *entity B trusts entity A* is not always true.

Among the various (non-technical) facets of trust, in the cooperative platform it plays a crucial role. This has been demonstrated in different e-commerce studies [46, 47], where online shoppers must necessarily rely on the functioning mechanism of the online store to make the purchase (i.e., use the credit card in a potentially unknown online store). These studies suggest to measure trust as the belief that a platform is honest, reliable, and competent.

Mapping these dimensions to BCs, a permissioned deployment model with a consensus necessarily open to the participation of all members within the cooperative defense meets these requirements. The capability to create an immutable and publicly (within this context) available record of transactions is seen as an enabler of trust [43]. In addition, the definition of rules between participants through SCs does allow to verify the execution of the SC defining the cooperation. However, algorithmic trust is not limited to the correct functioning of the algorithm, but also includes a variety of socio-technical factors, such as its formal and legal correctness beyond the technical solution.

3.4 Truthfulness of Mitigation Proofs

After the acceptance of the service mitigation terms by both parties (*T* and *M*) and *T* send funds to be locked in the SC, a period begins in which *M* must send proof of service mitigation. At this stage, a technical challenge is the absence of guarantees that the service was performed as requested since it is performed outside the premises of T. This problem was detailed in a previous work [48] and this subsection presents an overview (*cf*. Fig. 3) on the challenge of verifying the quality of the mitigation service.

VNF Marketplace: Allows for a *T* to encapsulate mitigation actions as a software that can be directly deployed on commodity hardware running on *M*'s site. In such an approach, a marketplace for VNFs can be built for all entities involved in the cooperative defense. Then, a *M* loads the VNF certified by *T* directly from the marketplace to perform the mitigation service using a cookbook with negotiated on chain e.g., list of attacking addresses and a mitigation action. While this approach provides a high degree of isolation, it does not guarantee that *M* would not tamper with its execution environment. Solely deploying a VNF is not a reliable proof of

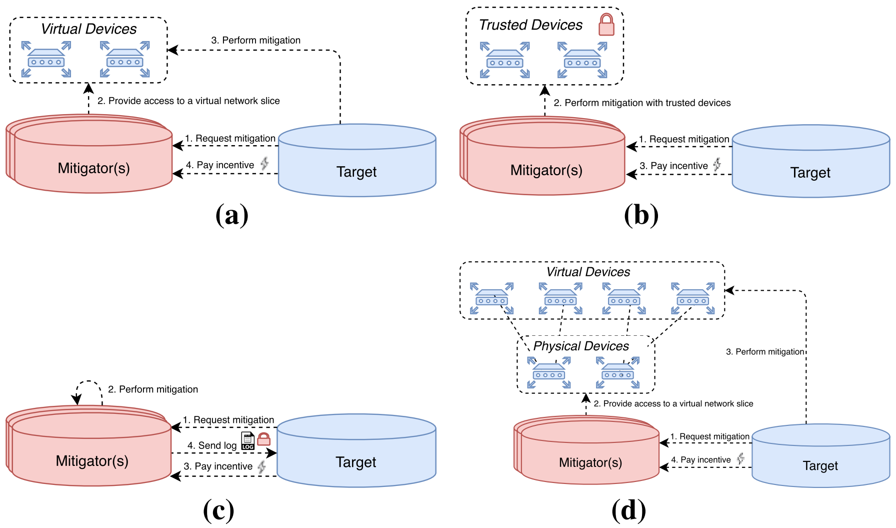

Fig. 3 Approaches toward a verifiable mitigation proof [48]: **a** VNF marketplace, **b** trusted platform, **c** secure logging, and **d** network slicing

mitigation and *T* still needs to trust that *M* will run untampered VNFs directly from the VNF marketplace.

Trusted Platform Module: A TPM allows to extend the chain of trust up to the VNF itself. In combination with a VNF marketplace, it is possible to provide a mitigation service in which VNF requests are always handled by known and trusted hardware. However, this approach imposes scalability concerns once TPM modules are a feature available only as a standalone chip or as a solution integrated into the motherboard, but it does not come pre-installed on networking equipment. Still, it would not be possible to ensure the truthfulness of the mitigation once a malicious *M* would be able to change the network flow to the system running the mitigation VNF and lead it to believe that it is seeing all the traffic while in reality, parts or all of the attack traffic have been rerouted and no mitigation seems to be required anymore.

Secure Logging: Is the production of a log outputting the effect of a mitigation action, which can be leveraged by previous approaches, where a VNF certified by *T* and running on a TPM module does store logs inside a BC to ensure immutable evidences. However, similarly to previous approaches, it is still not possible to guarantee the truthfulness of a mitigation test, since underlying traffic flows can be tampered before reaching VNFs required by *T*.

Network Slicing: An approach leveraged by SDN networks in which it is possible to virtualize network segments and allow *T* to install specific flows to perform the mitigation. Therefore, instead of allowing a virtual network function (whose source code may not be known by *M*), controlled access to the infrastructure (specified in the slice) is allowed to *T*. However, it is still possible that the slice provided by a malicious *M* is tampered with so that actions defined by *T* have no real effect on the underlying traffic.

4 BloSS Protocols Design and Application

The design of *BloSS* considers an on-chain part, where the cooperation logic is implemented in SCs, and an off-chain part, whose prototype was based on SDN to facilitate with the network management system signaling attacks and the implementation of mitigation actions. In addition, *BloSS* is based on am Ethereum Proof-of-Authority (PoA) consensus, i.e., access to the overlay network is restricted to the alliance of cooperative members in certain countries or regions.

4.1 Incentives and Fairness

While providing many benefits, a cooperative defense also poses many challenges e.g., why often competing organizations would help each other. In a competitive environment, trust needs to be established. Solely relying on a voluntary contribution (i.e.,

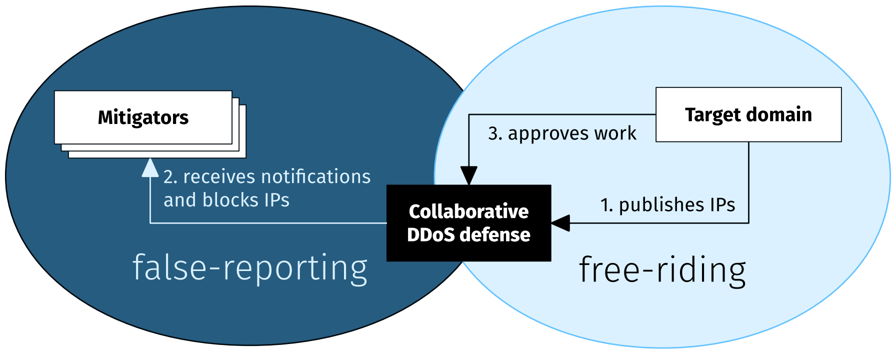

Fig. 4 Social dilemma of false-reporting and free-riding in collaborative DDoS defenses [17]

accepting defense requests) creates a favorable environment for free riding peers (consuming resources without contributing). This situation, and the social dilemma that the business partners find themselves in is illustrated in Fig. 4. First, the attack T publishes malicious IP addresses. Second, multiple Ms adjust the configuration of their network devices to filter and drop the malicious packets. In a third step, the attack T evaluates the effectiveness of the mitigation service.

A reputation scheme allows contributors and consumers of the network to rate entities that request protection in a cooperative defense. These systems have already been proven useful for e-commerce websites to incentivize peers to contribute with relevant information and establish fairness among peers. A basic scenario illustrating fairness problems in the DDoS mitigation process can be divided into three stages, as depicted in Table 3. Firstly, between the mitigation request by the DDoS attack target T and the actual blocking of the malicious IP addresses by a M domain M. Secondly, after the delivery of the mitigation service and the payment by domain T. Table 3 illustrates three possible mitigation histories and outcomes, depending on how T and M act.

Whenever a M submits a proof of mitigation, there is no automated way for other peers to build a consensus on the quality of the service delivered [48]. In other words, the receipt in a "payment-for-receipt" exchange process cannot be automatically issued by an SC, because in the worst case, any upload is accepted as successful delivery and the DDoS T domain would pay for a worthless receipt. Attack size and amount of payment are relevant factors that could result in severe financial losses for the attack T. In order to avoid this problem, the reputation process depends on the attack T to validate and rate the outcome of the mitigation service within an a priori agreed deadline.

Table 3 Cooperative defense scenarios with DDoS attack target T and mitigator M

Stage	1. Request	2. Service	3. Payment
Behavior	T requests	M blocks	T pays
	T requests	M blocks	T refuses
	T requests	M refuses	–

It is reasonable to assume that T's costs inflicted by the attack are higher than the costs of domain M, which is providing the mitigation service. A rational M would be better off not providing the costly mitigation service in the short term (i.e., betray). Also, T has strong incentives to refuse payment (i.e., betray). Since in a repeated game, the roles of T and M could be swapped, they are better cooperating in view of future attacks. A reputation and incentive scheme can incentivize the peers because it records and stores past behavior. This data can be analyzed by other peers before committing new transactions and can help them to make better decisions. The peers will preferably transact with reputable colleagues. This leads to an increase in successful transactions overall, i.e., it increases social welfare. The percentage of successful interactions can serve as a useful performance measure to evaluate a reputation and reward scheme [49].

Furthermore, reputation points used to rate T and M are preferably not the same, because a task owner with a good rating does not necessarily need to be a good M and vice-versa. Therefore, reputation earned as M is stored separately from the reputation earned as attack target or task owner. A simple metric gives the analyzing peers a clear understanding of a peers reputation. Added to this, a complex metric might lead to feedback loops. There are two types of reputation sources, subjective and objective. While the subjective reputation is composed of positive and negative ratings from other peers, the objective, historical metrics can be obtained from the public BC history. For example, the reputation system prototype in this thesis allows to compute the following metrics (among others):

- Customer age: The customer age can be derived from the block timestamp when the customer ID was created.
- Number of interactions: Completed tasks (negative, positive and unknown rating) and the number of interactions for a customer are obtained through the public task states.
- Average satisfaction: The average satisfaction with a peer (ratio of positive and negative ratings) signals the general satisfaction with this customer.
- Number of completed tasks: Observing the amount of completed tasks over time helps to compare historical with current customer performance.

4.2 *BloSS* Cooperative Protocol

Figures 5 depicts possible states and transitions depending on the message sender (i.e., caller) of the function, including the rating of both, the mitigation service performed by an M entity accepting a mitigation request and a T entity, the T of the attack.

After the deployment of the SC, the default state *Request* is set until the T requests defense from a M by initializing, which changes the state to *Approve*. The initialization contains important variables (e.g., network information, deadline interval, or minimal amount of funds), which are not changed during the following process until the SC is reused through initializing or re-initializing. During *Approve*, the chosen M may cooperatively accept or is uncooperative deny the request which either leads to the state *Funding* or the end-state *Abort*. The negotiation of SC parameters can

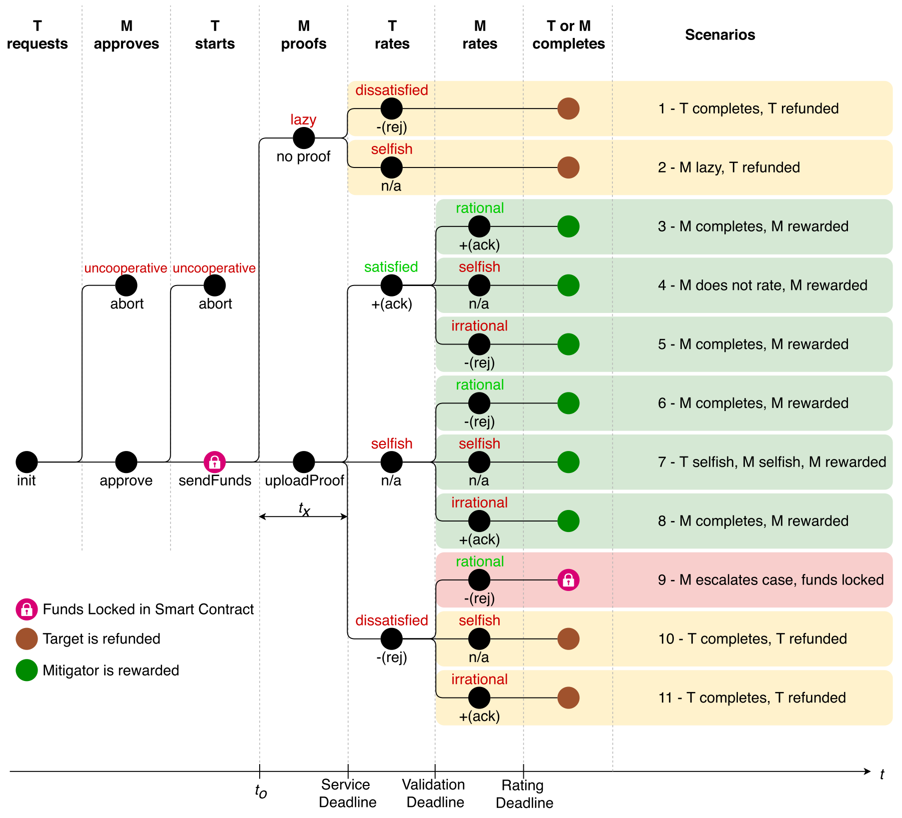

Fig. 5 *BloSS* SC for a collaborative, on-chain DDoS signaling

take place off-chain through a direct communication channel if needed. For example, the Whisper protocol (Ethereum) could be used to agree on these SC parameters. This negotiation phase could as well be designed two-ways, allowing *T* to submit an actively "request for service" to *M*. Once the offer reaches *T*, the domain under attack can draft a valid mitigation SC.

While in the *Abort* state, the *T* may choose to initialize with a different *M* or re-initialize with the same *M* to reach *Approve* again. However, if the *M* is cooperative, the *Funding* state begins, and after sending the required incentives, the state switches to *uploadProof* and the funds are locked into the SC until completion. *T* writes the attack information (e.g., IP address ranges, packet captures, request headers, notes, and other patterns [20]) to the external off-chain storage (e.g., IPFS [21]). During the service time window (t0), *M* is supposed to upload a proof of service. *M* will have an incentive to submit a proof (even a forged one) since time works against *M*. *T* is obliged to validate the effectiveness of the service and rate *M* during the validation time window. *T* will have the interest to vote and proceed in the mitigation process since the clock works against *T* during this time window.

The *M* can upload proof in the form of a report that is used as evidence for work that has been done by the *M* to mitigate the DDoS attack. In the best-case scenario, both parties keep their promises and deliver money and service on time. The offers and SC proposals can be made repeatedly by the two domains until they find an agreement. During this process, *M* and *T* also agree on the deadlines for service delivery, validation, payment, and rating. Playing against the rules will result in a financial loss for both parties, as depicted in Fig. 6. For example, if *T* misses to acknowledge or reject the proof in response to the delivered service, *M* will be rewarded.

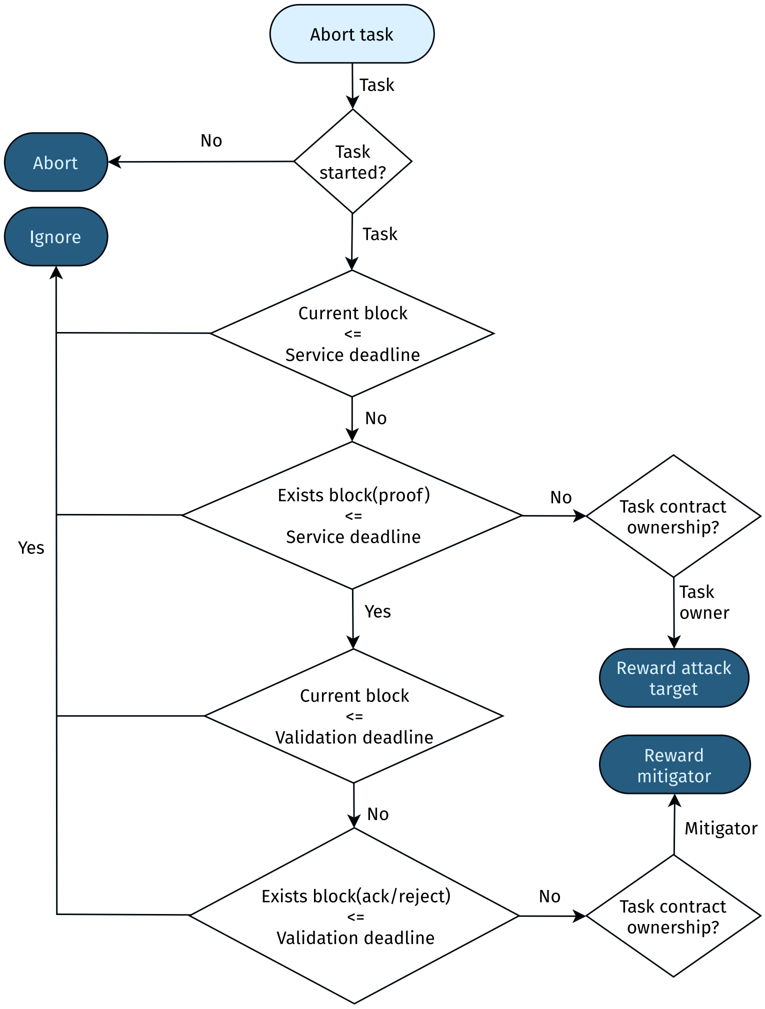

Fig. 6 Payout process: *T* is refunded, if no proof was uploaded and *M* is rewarded, if *T* does not validate in time

Therefore, *T* is penalized and has no chance to retrieve the payment. Both parties are free to abort the protocol, if the counter-party did not deliver the result expected during the agreed time-frame. The decision flow that shows who gets rewarded on abort is visible in Fig. 6. If *M* did not deliver the service, *T* will retrieve the payment. Similarly, if *T* did not respond during the validation time window, *M* will be reimbursed. *M* is not allowed to rate, if no proof was uploaded. This ensures that a user's reputation is only changed through valid interactions, which impedes bad-mouthing [50]. In order to bad-mouth a competing domain and deteriorate its reputation, the attacker needs to buy at least as many mitigation services from this competitor. This increases the competitors' profit and is irrational [50]. In contrast to *M*, the process allows *T* to rate, even if no proof was uploaded during the service window.

However, even if the BC preserves a transparent audit trail for all transactions, it cannot compensate for lack of ground-truth. This holds for the uploaded proof of service as well as for user-defined, subjective ratings, in which there is no automated way to fully determine the truthfulness of a proof or rating. If the defined the deadline interval is missed, or there is no upload of a proof, the *M* is marked as lazy, and the updated state is *Abort*. If a proof is uploaded, the rating process of the *T* and the *M* begins. During rating, only the case where both actors are dissatisfied leads to the *Escalate* end-state in which the actors themselves must manually find a solution to find consensus upon the service and incentive. All other combinations of the *T* or *M* rating satisfied, selfish or dissatisfied, lead to the end-state *Complete*. When *Complete* is reached, the locked funds from *BloSS* are released and transferred to either the *T* or the *M*, depending on missed deadlines and ratings.

4.3 Implementation

Storing participating *Targets (T)*, *Mitigators (M)*, and their respective addresses in a Register SC (i.e., a meeting point for participants of the collaborative defense), enables a search for *M*s and an efficient management of active processes. This registry type SC extension is important to facilitate the process of finding a known *M*, which is already registered, waiting for a *T* to interact.

When *M* is not be found in the *Register*, the default address value is received by *T* and further searching for a specific *M* can be done. However, in case the required *M* is found, *T* addresses can be observed by *M* in order to interact with the SC itself. At this point, an SC has stored the address of *T* and *M* in order to either allow or disallow access to functionality or a change of SC states.

```
function init(uint _DeadlineInterval,uint256 _OfferedFunds,
    string memory _ListOfAddresses) public {
        require(msg.sender==Target,"[init] sender is not required
            actor");
        require(Mitigator!=address(0),"[init] mitigator is not set.")
            ;
        require(CurrentState==Enums.State.REQUEST ||
          CurrentState==Enums.State.COMPLETE ||
          CurrentState==Enums.State.ABORT, "[init] State is not
              appropriate");
          Target = msg.sender;
          DeadlineInterval = _DeadlineInterval;
          OfferedFunds = _OfferedFunds;
          ListOfAddresses = _ListOfAddresses;
          CurrentState = Enums.State.APPROVE;
          emit ProcessCreated(msg.sender,address(this));
  }
function approve(bool descision) public{
  require(msg.sender==Mitigator,"[approve] sender is not required
      actor");
  require(CurrentState==Enums.State.APPROVE,"[approve] State is not
      appropriate");
    if(descision){
      CurrentState = Enums.State.FUNDING;
    }else{
      endProcess();
    }
}
function sendFunds() public payable {
  require(msg.sender==Target,"[sendFunds] sender is not required
      actor");
  require(CurrentState==Enums.State.FUNDING,"[sendFunds] State is not
        appropriate");
  if(msg.value >= OfferedFunds){
     CurrentState = Enums.State.PROOF;
     setNewDeadline();
  }else{
     endProcess();
  }
}
function uploadProof(string memory _Proof) public {
  require(CurrentState==Enums.State.PROOF,"[uploadProof] State is not
        appropriate");
  // when lazy
      if(now > Deadline){
          CurrentState = Enums.State.RATE_T;
          setNewDeadline();
          return;
      }
  require(msg.sender==Mitigator,"[uploadProof] sender is not required
        actor");
  Proof = _Proof;
      CurrentState = Enums.State.RATE_T;
      setNewDeadline();
}
```

Listing 1: First Interactions of T and M as Mapped in *BloSS*: Initiate SC, Its Acceptance, and an Upload of Proof by M

Retrieving an M (i.e., retrieving an M from the *Register SC*) requires to know its identifier by the Target T. If the identifier and the M address exist without having an SC address assigned to the same *struct*, the address of this SC is assigned and registered and that M address is returned to the SC instance. A protocol is created by associating addresses of T with M in a verifiable on-chain SC. Enabling both *Register* and *BloSS* SCs to interact, offers advantages, such as calling methods or retrieving information from each other. The logic in the SC can be separated from the registration process. Thus, a single deployment of a *Register* is needed whereas many SCs can be created, allowing every *T* to deploy *BloSS* and search for an *M* in the *Register*.

```
function ratingByTarget(uint _Rating) public{
  // require the number to be 0,1 or 2
  require(CurrentState==Enums.State.RATE_T,
  "[ratingByTarget] State is not appropriate");
  require(_Rating == 0 || _Rating == 1 || _Rating == 2,
  "[ratingByTarget] Rating must be 0, 1 or 2");
  if(now > Deadline){
          TargetRating = Enums.Rating.NOT_AVAILABLE;
          CurrentState = Enums.State.RATE_M;
          setNewDeadline();
          if(bytes(Proof).length==0){
              return endProcess();
          }
          return;
      }
      require(msg.sender==Target,
      "[ratingByTarget] sender is not required actor");
  TargetRating = Enums.Rating(_Rating);
  if(bytes(Proof).length==0){
    require(_Rating == 0 || _Rating == 1, "[ratingByTarget]
    Cannot rate Mitigator positive when no proof is uploaded.")
      ;
    return endProcess();
  }
      CurrentState = Enums.State.RATE_M;
      setNewDeadline();
}
function ratingByMitigator(uint _Rating) public{
  require(CurrentState==Enums.State.RATE_M,
  "[ratingByMitigator] State is not appropriate");
  require(_Rating == 0 || _Rating == 1 || _Rating == 2,
  "[ratingByMitigator] Rating must be 0, 1 or 2");
      if(now > Deadline){
          MitigatorRating = Enums.Rating.NOT_AVAILABLE;
          return endProcess();
      }
      require(msg.sender==Mitigator,
      "[ratingByMitigator] sender is not required actor");
  MitigatorRating = Enums.Rating(_Rating);
    return endProcess();
}
```

Listing 2: Rating by *T* and *M* in the *BloSS* SC

By storing these pairs of addresses in an efficiently verifiable data structure, a mapping allows for a usable client-side implementation, where only one instantiation of *BloSS* is needed for the interaction between the *T* and the *M* and one key with which a search on the map can be done. And because the instantiation of the protocol references the same SC on both sides, the storage, states, and fields are the same as well, and the interaction on the *BloSS* may proceed. In order for a client to access the *BloSS* instance the Application Binary Interface (ABI) and the address of the *BloSS* is required.

Listing 1 summarizes main functions in the *BloSS* SC, which maps initial steps of the SC initialization, such as setting funds and deadlines to complete the mitigation service (e.g., methods *init* and *approval*). Thus, the execution of the SC prevents one to return states before the SC reaches its final stage (completion). For instance, it is not possible for an *M* to send another mitigation proof after the proof rating by the *T* (e.g., each method verifies the previous state with the *require* statement). Another important aspect of the *BloSS* SC is the rating by *T* and *M* (*cf.* Listing 2).

Rating actions in Listing 2 consider both, rational and irrational behavior from *M* and *T*. Whereas a rational action means that one of the two parties acts as expected (best scenario) within the possible protocol alternatives, an irrational action means that either party loses a deadline or acts maliciously. The rating is the last step before completing the protocol, in which the call returns another function to complete the protocol, e.g., *endProcess()*.

4.4 Discussion of Scenarios

At scenarios *1* and *2* *T* is dissatisfied as a result of *M* not uploading a proof, leading to funds being refunded to *T*. Even without a reply from *T*, *T* is refunded (*cf.* scenario *2*). To remove the possibility of ballot stuffing, where *T* would rate satisfied in case no proof being uploaded, *T* is not allowed to rate positively here. However, if a proof is uploaded before the service deadline ends, *T* can rate satisfied, be selfish, or dissatisfied. *M* can always react and rate rational, be selfish, or irrational, which enables further possibilities, although only three ending states can be reached after locking the funds.

Scenario 3 shows *T* rating satisfied and *M* reacts rationally with a positive rating, leading to funds being transferred to *M*. Similarly, scenarios *4* to *8* of *T* rating satisfied or being selfish (i.e., missing the service deadline) lead to *M* being rewarded, regardless of the rating by *M*. Case *T* is dissatisfied with the proof uploaded by *M* and *M* reacts in a rational way by rating dissatisfied as well (*cf.* scenario *9*). The case escalates and further investigations are needed for resolving it. Scenarios *10* and *11* lead to *T* being refunded due to rating dissatisfied and *M* being selfish or rating irrationally satisfied.

4.5 Decentralized *BloSS* Application

An overview of the *BloSS* Decentralized Application (dAPP) is provided in Fig. 7 detailing connections between all its modules. The *BloSS* is the component where each service provider taking part in the cooperative defense, can post information about an ongoing attack to the Ethereum, i.e., the connector to the on-chain contracts. It uses a REST interface to facilitate the isolation of the *BloSS*module, encapsulating the entire module together with Pollen BC and Pollen data store as SDN applications and, possibly, as a VNF running on commodity hardware. The goal of this design is not to impose restrictions on the underlying networking hardware, further simplifying the interaction with the *BloSS* and its modules via REST interfaces.

Data exchange is accomplished with the "Pollen" set of modules, and the "Stalk" module handles network-related tasks. Pollen is divided into dedicated modules for the specific data exchange duties of the *BloSS*, which includes a BC module for access to the Ethereum, a data storage module managing information on the IPFS. Attack information posted to the BC is not directly stored on the BC due to limited block sizes and to maintain the information confidential. For this purpose, IPFS is used as a decentralized and highly scalable storage solution to hold attack information. Each service provider running the *BloSS* also maintains an IPFS node to enable

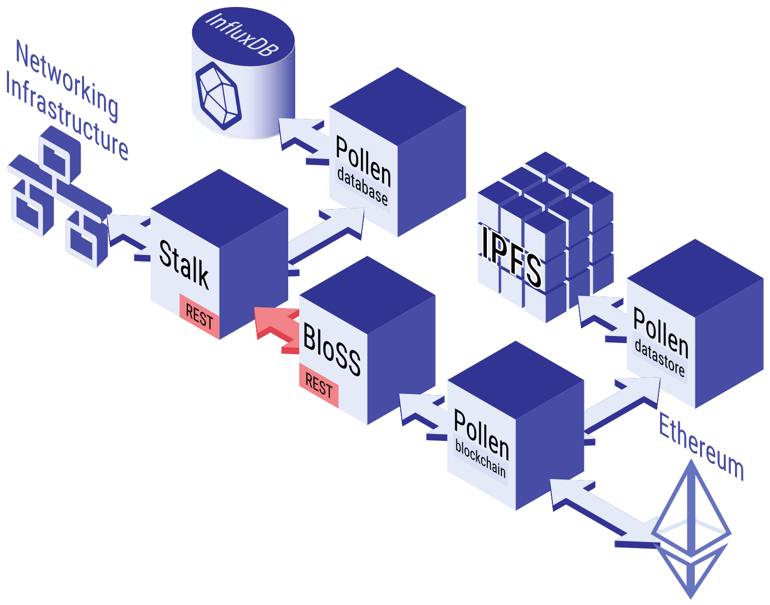

Fig. 7 Architecture of the blockchain signaling system (*BloSS*)

the decentralized storage. Whenever a new set of attack information is posted to the BC, the data is first stored in IPFS, and only the hash as a unique identifier of the storage location within IPFS is stored in a block on the Ethereum.

The Pollen data store also includes an encryption component. The encryption of attack information posted to IPFS ensures the confidentiality and the integrity of the attack information based on a per-message signature bundled with the attack information. Confidentiality is an essential attribute of the data exchange between service providers, since the attack information can be sensitive in regards to implicating individuals both as victims of an ongoing DDoS attack or as perpetrators of said attack.

Verifying the integrity of attack information allows for holding each service provider accountable for the information posted to the BC and makes forgery of attack information impossible. The integrity-check is enabled through a public key published by each service provider to the BC and, therefore, available to all providers participating in the *BloSS* defense alliance. Without this measure, forgery of attack information would allow a malevolent party to indicate specific IP addresses as being the source of an ongoing attack and to block flows from these addresses to the *T* address specified in the attack information.

4.6 Security Considerations

Figure 8 shows a prototypical defense scenario involving an *M* as well as *T*'s AS. Attack detection is outside the scope of the *BloSS* so the first step includes compiling

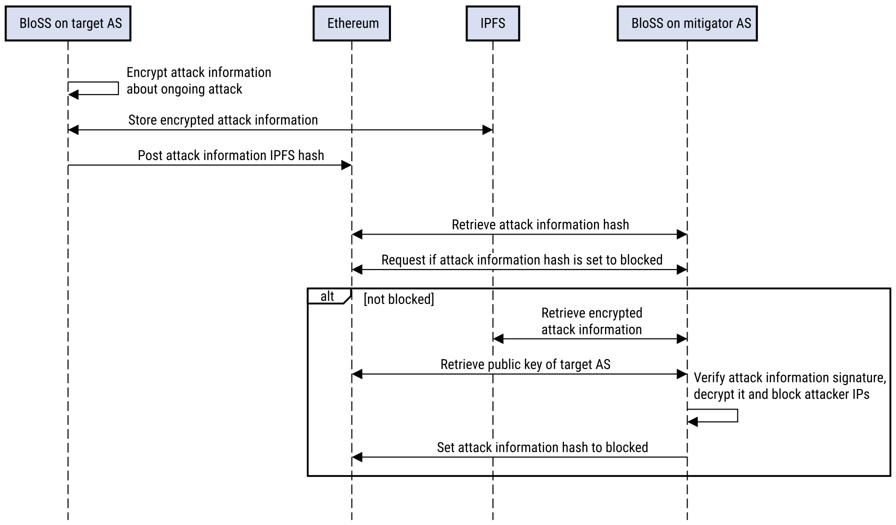
BloSS on target AS
Ethereum
IPFS
BloSS on mitigator AS
Encrypt attack information about ongoing attack
Store encrypted attack information
Post attack information IPFS hash
Retrieve attack information hash
Request if attack information hash is set to blocked
alt
[not blocked]
Retrieve encrypted attack information
Retrieve public key of target AS
Verify attack information signature, decrypt it and block attacker IPs
Set attack information hash to blocked

Fig. 8 *BloSS* defense scenario including a *T* and an *M* AS

the attack information and encrypting it to later store to IPFS and post the IPFS hash to the Ethereum. To minimize access to IPFS as well as Ethereum to access attack information and the public key of *T*'s AS, the attack information hash is connected to a Boolean indicating whether the information has already been accessed by *M*'s AS in order to block the attackers. However, incentive schemes necessary to realize a true Mitigation-as-a-Service (MaaS) offering as outlined in [48] are out of the scope of this implementation of *BloSS*.

Encryption and decryption is built with the Python Cryptography library [51] and the choices for cryptographic algorithms as well as key lengths and other cryptographic details are based on an article by Colin Percival [52]. PollenEncryption uses both asymmetric cryptography through RSA with 2048 bit keys and symmetric cryptography through Fernet, which is essentially the Advanced Encryption Standard (AES) block cipher in Cipher Block Chaining (CBC) mode using a 128 bit key [51].

Asymmetric encryption is used for two tasks in PollenEncryption: To encrypt the symmetric key as well as to cryptographically sign the unencrypted attack report with the private key of the sender. Signing the attack report with the private key allows the receiver to verify the authenticity of the attack report by using the public key available on the BC for cryptographic verification. Instead of directly encrypting the attack report through asymmetric encryption, the symmetric Fernet scheme is used. Asymmetric encryption is very useful since no secret key exchange has to occur, however it is not well suited to encrypt large amounts of data since the size of data to be encrypted cannot exceed the key size of 2048 bit [52] (Fig. 9).

After encrypting the attack report and symmetric key as well as signing the attack report, all three components, signature, encrypted symmetric key and encrypted attack report are stored in IPFS as a JavaScript Object Notation (JSON) object for

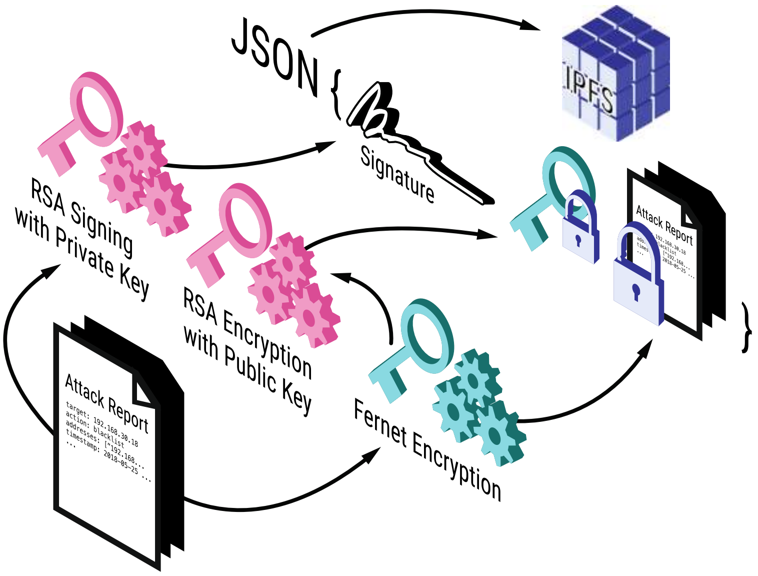

Fig. 9 Encryption Procedure for Off-chain Data Transfer via IPFS

easier handling through the Stalk and *BloSS* REST APIs. In addition to encrypting each set of attack information when posted to IPFS, all communication between the Pollen datastore module and IPFS is encrypted with the libp2p-secio [53] stream security transport, which is based on TLS 1.2. Transport encryption would not be strictly necessary since the data being transported is already encrypted, however this allows a certain degree of anonymization for the defense system users since it is not possible to ascertain which AS accessed which attack information when simply looking at the communication between IPFS and AS. This added anonymity can also be seen as an additional factor contributing towards increased confidentiality.

Communications between individual Ethereum nodes is also encrypted as detailed in the DEVp2p [34], which contributes to an increased confidentiality. However, due to the distributed ledger characteristics of Ethereum, transactions can be traced back to the party responsible for the chain. The last part of the communication chain with the REST interface between *BloSS* and Stalk is not encrypted. This is by design since the REST interface is designed to only be accessible on the same machine to allow for simple communication between *BloSS* and Stalk while enabling a high degree of encapsulation for the *BloSS* module in order to allow the implementation of Proof-of-Mitigation schemes.

5 Experimental Evaluations

The goal of an experimental evaluation is to measure quantifiable parameters, such as the *gas usage* and the *performance*. Based on those outcomes and within a global evaluation, it is possible to label *BloSS* a feasible approach. However, to achieve a global *BloSS* deployment, a simulation based on Truffle and Ganache Suite and a local deployment on a test net was performed. Since Truffle and Ganache are simulation environments, they allow for a verification of the correctness of SCs running on Ethereum. Since previous results from local deployments on hardware were published in [33], this evaluation discloses the global evaluation results.

Hardware Setup: *BloSS*' evaluation was based on Amazon Web Service (AWS) instances deployed in Ohio, Tokyo, and São Paulo. These EC2 *Amazon t2 medium* instances were configured with two threads on either an Intel Xeon or an AMD EPYC-Core running at up to 3.0 GHz and with 4 GByte of RAM. All instances were synchronized with the Ethereum Rinkeby BC in order to enable separation of the Target T and Mitigator M. Each location was tested separately between the target in Zürich and São Paulo and the mitigator set to Ohio and Tokyo, respectively. Table 4 lists Round Trip Time (RTT) results executed on AWS instances in Zurich, São Paulo, Tokyo and Ohio. RTT times can be used to evaluate, whether a potential statistical significance across RTT may be found during execution of *BloSS*.

Software Setup: A synchronized *BloSS* node utilizes Geth to interact with a SC deployed. Instead of relying on one full node offered by Infura with two accounts, the *BloSS* tests with the Geth client are executed on two synchronized nodes in Zürich and Ohio to measure the *gas usage* and the *performance*.

To rectify the problem of non "full" block times, a synchronization process before the actual registration or signaling process can be completed. By synchronizing the measured time-frame closer to the beginning of the starting block, i.e., maximizing x, global results become comparable to results on a previously configured local Rinkeby. A second approach to retrieve the correct time measurement on the global performance tests was to run tests $n = 20$ times, such that the first run acts as a synchronization step. Thus, depending on the scenario and the deadlines missed, full block times represent the worst case in terms of *BloSS* performance.

By running the target script on an AWS instance in Ohio and a M script on a node located in Zürich (both synchronized to the Rinkeby network), the average global Rinkeby processing time with $n = 20$ is 96.950 s and the average standard deviation is 1.146 s (*cf.* Table 5). Since the control condition has been tested and

Table 4 Average RTT between nodes (ms)

From	To			
	Tokyo	São Paulo	Ohio	Zürich
Tokyo	–	270	159	223
São Paulo	270	–	130	130
Ohio	159	130	–	119
Zürich	276	223	119	–

Table 5 *BloSS* global Rinkeby processing times (s) (Zürich-Ohio)

Scenario	Average time (s)	Standard deviation (s)
1	88.938	3.025
2	88.616	1.099
3	87.973	2.110
4	104.090	0.509
5	88.006	1.361
6	104.358	1.340
7	118.900	0.442
8	103.932	0.458
9	89.265	0.711
10	103.331	1.038
11	88.956	0.509
Average	96.950	1.146

Table 6 *BloSS* control condition processing times (s) (Tokyo-São Paulo)

Scenario	Average time (s)	Standard deviation (s)
1	89.267	1.543
2	89.579	0.830
3	89.661	0.851
4	104.661	0.966
5	89.427	0.658
6	105.187	1.358
7	120.136	1.204
8	104.924	0.921
9	89.149	0.785
10	103.572	0.718
11	88.235	1.611
Average	97.618	1.040

evaluated as well, similar results in terms of average processing time and average standard deviation are expected. However, similar results were reached as shown in Tables 5 and 6, while the nodes were not synchronized at all times due to time-outs, i.e., missed deadlines. This is due to the full nodes, which are geographically in close proximity of the AWS instances in Tokyo and São Paulo, but not being synchronized at all times [54].

For both global averages, average times measured show a similar result, with a slight difference in the average processing time of 0.668 s representing the difference of approximately 0.7%. By reaching these similar results in both global Rinkeby tests and removing the corresponding RTT shown in Table 4 the

Table 7 *BloSS* global Rinkeby gas use (Gwei)

Scenario	Total gas	*T* gas used	*M* gas used
Deployment	3,795,264	3,795,264	–
1	253,999	192,836	61,163
2	253,999	193,236	61,163
3	314,438	169,225	145,213
4	314,490	169,225	145,265
5	315,050	169,225	145,825
6	314,370	168,928	145,442
7	314,508	168,928	145,580
8	315,154	168,928	146,226
9	291,401	169,139	122,262
10	314,716	169,139	145,577
11	315,362	169,139	146,223
Average	301,626	173,450	128,176

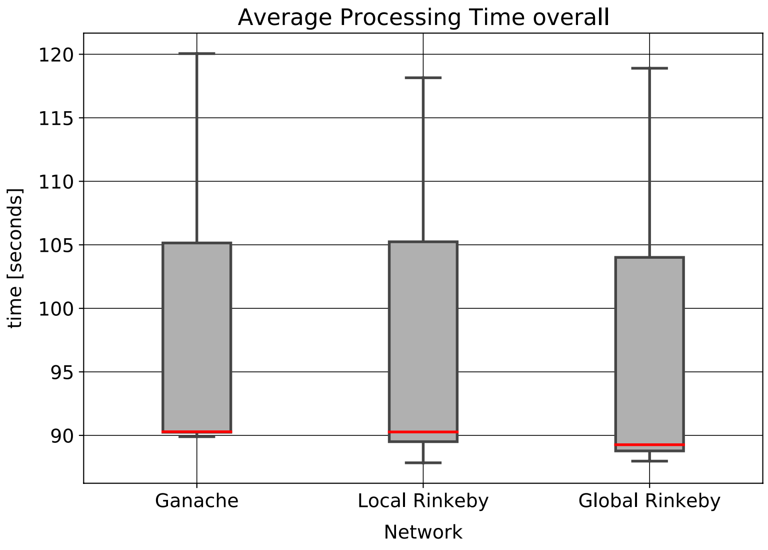

Fig. 10 Comparison of global average times on ganache and localRinkeby

difference in average processing times for the scenarios is only 0.5171 s. It should be noted that 20 test runs per case may not lead to exact average values. Also, every test on the global Rinkeby network was tested with varying (not precise) average block times of 15 s.

Average Gas Cost: The *Gas* used in the global Rinkeby tests show slight differences compared to Ganache and local Rinkeby tests, which were all based on the same truffle test scripts. While the registration gas usage does not differ in any test, the total *BloSS gas used* on average differs at 15,366 Gwei compared to local Rinkeby and Ganache. All global Rinkeby gas used results are shown in Table 7.

Figure 10 depicts overall average times to complete the execution of *BloSS* for the three infrastructures and eleven scenarios, where each scenario had $n = 20$ test runs. In total, 660 scenarios were executed to assess the worst-case time to complete the execution of *BloSS*. Similar processing times were observed as in local and simulated deployments, and it could be shown that by using Infura as a proxy, the behavior of the public test net Rinkeby with a simulated actor in Ohio could be mimicked. This implies that the performance of *BloSS* is stable and can be accessed in a similar way through the Truffle framework and the Geth client. Even the controlling condition, where actors are located in São Paulo and Tokyo, showed similar results, when both nodes were synchronized. Thus, the confidence in the cooperative signaling is enhanced.

The overall gas used to deploy and utilize the *Register SC* is the same, whereas *BloSS* shows slight differences. On average an *M* needs to pay gas costs ranging from 0.04 US$ for the gas price of 1 Gwei to 0.8 US$, when prioritizing transactions with a gas price of 20 Gwei. A *T* is required to pay more gas on average, which ranges from 0.06 US$ to 1.13 US$. However, the deployment of *BloSS* must be paid for as well and one-time costs for a deployment of *BloSS* range from 0.82 US$ to 16.40 US$. Thus, the total costs to deploy both SCs require a payment from 1.03 US$ to 20.51 US$ (cf. Table 8 with a conversion rate of 216.00 US$/Ether).

6 Discussions and Conclusions

DDoS defense mechanisms see an increasing number of cooperative approaches. The DOTS architecture by the IETF is a prominent proposal built on top of a gossip-protocol crafted for DDoS defense signaling [32]. Other approaches, such as DefCOM, are based on lightweight overlay networks to enable signaling in a peer-to-peer (P2P) fashion [24]. Therefore, *BloSS* is more akin to DefCOM, since it builds on Ethereum instead of developing new communications from scratch. By leveraging the distributed nature of BCs, *BloSS* scales to the demand of modern distributed DDoS defense systems. Conversely, DefCOM relies on complex

Table 8 Total average global Rinkeby costs in US$ per *BloSS* instance

	Deployment	Target	Mitigator
Register (Gwei)	952,459	88,866	56,343
BloSS [Gwei]	3,795,264	173,450	128,176
Total Gas Used (Gwe)	4,747,723	262,316	184,519
Total Costs max. (US$)	20.51	1.13	0.80
Total Costs min. (US$)	1.03	0.06	0.04

P2P message exchanges, which are more prone to failure within distributed settings than a BC consensus mechanism as *BloSS* deploys.

6.1 Robustness of the Reputation System

By analyzing the design of the cooperative protocol it is possible to make an analysis of different types of possible fraud. An evaluation with customers *M* and *T* with different profiles (e.g., honest, malicious, lazy, and others) is performed in the authors' previous work [17]. Table 9 summarizes the analysis, which is followed by a discussion on each type of fraud.

Free-riding: This type of activity is prevented by design in *BloSS* by requiring *T*'s to deposit the incentive required by *M*'s into the SC. Since the SC is designed as a state machine, it is not possible to circumvent this step making the mitigation service start before funds are locked into the SC.

False-reporting: Fraud can happen when a malicious *M* assigns a false rate to an honest *T* at the end of the interaction. Although the protocol allows for this, no rational incentive exists, since actions are recorded on the BC. Thus, future interactions of a malicious *M* can be tracked by all *T*'s.

Sybil- and Collusion Attacks: *BloSS* excludes the possibility where a customer can boost its reputation by creating mitigation SCs with itself. A possible deployment on a public ledger would enable actors, i.e., a *M* or *T*, to maintain multiple account pseudonyms on the BC and transacting between them to inflate reputation (Sybil attack). Therefore, it would be necessary to implement within the *BloSS* SCs an access control mechanism to prevent whitewashing or re-entry attacks on reputation systems.

Ballot Stuffing: *BloSS* is not immune against ballot stuffing. Besides transactions recorded on the BC, customers can agree on discounts and benefits over alternative communication channels. For instance, two malicious *T* and *M* would be able rate each other positively independently from the mitigation outcome in rounds where both can perform the role of *T* and *M*.

Bad-mouthing: A BC-based reputation system design impedes bad-mouthing in which a *T* or *M* can only provide feedback for transactions completed. This elevates costs of bad-mouthing a competitor, since a transaction has to be committed for each fraudulent reputation statement.

6.2 Discussion of Achievements

Overall, the technical, social, economic, and legal aspects of *BloSS* have been achieved. *BloSS* is a relevant solution as a defense strategy in cases of DDoS

Table 9 Assessment of reputation frauds

Target	Fraud	Short description	Achieved
System	Free-riding	Incentives are required to request mitigation services	●
	False-reporting	*M* is not incentivized to provide false-reports on *T* but the protocol allows such behavior, which can be tracked on future interactions	◐
Rating	Sybil and collusion	Whitewashing (re-entry) of identities is not prevent in a permissionless deployment	◐
	Ballot stuffing	Malicious *M*'s and *T*'s can collude to elevate their reputation	✗
	Bad-mouthing	Unfair ratings are not incentivized by design once a service is paid upfront by *T* and *M* only rates, when the service is completed	●

●=property ; ◐=property partially provided; ✗= property not provided large-scale attacks, such as the Memcached attack on GitHub servers surpassing 1.35 TBit/s [8] and DynDNS which peaked 1.2 TB/s resulting in the unavailability of significant Internet services [6]. In such cases, traditional centralized defenses can be easily overloaded, and cooperation between organizations is useful to lessen the impact of attacks. Hence, *BloSS* addresses different challenges of a cooperative defense such as providing incentives and tracking reputations among members. This not only encourages participation, but also punishes malicious behavior by members.

Technical: By designing the *BloSS* dApp with SDN and NFV-capabilities the key advantage of a quick deployment was engineered into *BloSS*, inherent to competing systems like CoFence [12] or Bohatei [13]. However, due to *BloSS*' modular architecture, it is neither limited to SDN-based networking infrastructures nor limited to an NFV-based solution. On the contrary, the networking module of *BloSS* termed Stalk can be adapted to various infrastructure deployments, while still maintaining connectivity over the RESTful interface to *BloSS* modules.

Social: Reputation and reward schemes integrated into *BloSS* prevent free-riding (attack targets) and false-reporting (mitigators). These mechanisms incentivize the rational behavior of operators in the long run. Selfish members are identified by looking at their past interactions on the BC. Furthermore, the payment of rewards provides a highly suitable countermeasure to dis-incentivize selfish customers. Mitigators are incentivized to execute the final service rating step, since otherwise they would deprive themselves of payments. In addition, it is possible to create circles by connecting different BloSS instances through interoperability between blockchains. This could be achieved employing a Notary-based agnostic solution such as [55, 56].

Economic: As incentives already stipulate social behavior, by providing a platform to exchange incentives, *BloSS* creates a new scenario for mitigation defenses [17]. By extending cloud-based protection services, a decentralized marketplace for protection services based on BCs is foreseen. In such a model it will be possible for members to create strategic regional alliances and use a portion of their infrastructure to perform mitigation services. Therefore, in combination with recommender services [57] as selection of a suitable protection provider based on specified requirements can optimize costs.

Legal: Legal and regulatory aspects are often intertwined. In addition to the possibility of creating circles of trust, which depends on the social requirements, multiple *BloSS* networks can be defined for national or regional circles of trust, respectively. Also, such circles make it possible to discriminate selection criteria based on the legal settings of each operator, possibly restricting the participation or interaction with certain regions or operators.

6.3 Conclusions

BloSS contributes to the modern security management for DDoS mitigation approaches with a cooperative defense logic and prototype as a proof-of-concept (available in [33]). It enables a flexible and efficient DDoS mitigation solution across multiple domains based on a permissioned PoA Ethereum [34], in which only pre-selected operators participate in the cooperative defense. Therefore, based on recently validated technical tools, such as BCs and SDNs, it became possible to provide a practically deployable, collaborative defense mechanism capable of overcoming the main challenges as stated above and in [9, 10].

The BC-based approach does not only enable the cooperative signaling of attacks, but also provides for an immutable and transparent platform allowing for incentives to be exchanged for mitigation services as well as tracking reputation. *BloSS* manages network operations and deploys service functions for the detection and mitigation of malicious traffic. The definition of contracts, especially SCs, stipulates the cooperative logic based on BCs and allows for the increase of trust among cooperative operators due to their transparent exchange of selected information and respective incentives on a per request basis.

Furthermore, execution times and costs of *BloSS* as presented are based on the worst-case scenario, i.e., a public BC infrastructure. For example, Target and Mitigators were configured to react to requests close to the deadlines configured in the contract. Therefore, it has to be noted that a PoA-based deployment of *BloSS* will reach a much lower, almost neglectable cost basis and an even further reduced block creation time. This was shown for the case of simulated and local Rinkeby deployments.

Overall, the main achievement and advantages reached with the design and prototypical implementation as well as the evaluation of *BloSS* include (a) the use of an existing public and distributed infrastructure, the BC, to flare white- or blacklisted IP addresses and to distribute incentives related to the mitigation activities requested. Furthermore, it provides a proof-of-concept for (b) a cooperative, operational, and efficient decentralization of DDoS mitigation services, and (c) a compatibility of *BloSS* with existing networking infrastructures, such as SDN and BC.

6.4 Future Work

Based on an even further increase in traffic and the frequency of DDoS attacks, it is expected that future network and service management operations will also have to encounter alternatives equally distributed. While existing cooperative approaches present operational challenges, future work for *BloSS* involves the analysis of how actors (especially targets and mitigators) (a) would interact based on different profiles (e.g., with malicious or honest properties) and (b) are impacted by different incentive values required to perform a mitigation service and, thus, simulating a DDoS protection market. Also, instead of storing raw names and strings in the *BloSS* register, hashes of data or even hashes of the storage address could be persisted within the BC, since transparency has to be taken into account. Based on those mechanisms ratings of the mitigator or the target the *BloSS* register can be extended, too, to enable a ranking and to separate positively rated actors from negatively rated ones.

Acknowledgements This paper was supported partially by (a) the University of Zürich UZH, Switzerland and (b) the European Union's Horizon 2020 Research and Innovation Program under Grant Agreement No. 830927, the CONCORDIA Project.

Funding Open access funding provided by University of Zurich.

References

1. Akamai. How to protect against ddos attacks—stop denial of service
2. Felici, M., Wainwright, N., Cavallini, S., Bisogni, F.: What's new in the economics of cybersecurity? IEEE Secur. Privacy **14**, 11–13 (2016)
3. IoTAnalytics. State of the IoT 2018: number of IoT devices now at 7B—Market accelerating, Fev (2018)
4. Akamai. The state of the internet (2020)
5. Maglaras, L., Kim, K.H., Janicke, H., Ferrag, M.A.,Rallis, S., Fragkou, P., Maglaras, A., Cruz, T.J.: Cyber security of critical infrastructures, vol. 4. *ICT Express,*, SI: CI and Smart Grid Cyber Security, pp. 42–45, (2018)
6. The Associated Press. Hackers Used 'Internet of Things' Devices to Cause Friday's Massive DDoS Cyberattack (2016)
7. Gu, G., Yegneswaran, V., Porras, P., Stoll, J., Lee, W.: Active botnet probing to identify obscure command and control channels. In: 2009 annual computer security applications conference, pp. 241–253. IEEE, New York (2009)
8. Kotey, S.D., Tchao, E.T., Gadze, J.D.: On distributed denial of service current defense schemes. Technologies **7**(1), 19–25 (2019)
9. Zargar, S.T., Joshi, J., Tipper, D.: A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. IEEE Commun. Surv. Tutor. **15**(4), 2046–2069 (2013)
10. Peng, T., Leckie, C., Ramamohanarao, K.: Survey of network-based defense mechanisms countering the DoS and DDoS problems. ACM Comput. Surv. CSUR **39**(1), 03–15 (2007)
11. Bocek, T., Stiller, B.: Smart contracts-blockchains in the wings. Digital marketplaces unleashed, pp. 169–184. Springer, Berlin (2018)
12. Rashidi, B., Fung, C.: CoFence: a collaborative ddos defence using network function virtualization. In: 12th international conference on network and service management (CNSM 16) (2016)
13. Fayaz, S., Tobioka, Y., Sekar, V., Bailey, M.: Bohatei: flexible and elastic DDoS defense. In: 24th USENIX security symposium (USENIX Security 15), pp. 817–832 (2015)
14. Rodrigues, B., Bocek, T., Lareida, A., Hausheer, D., Rafati, S., Stiller, B.: A blockchain-based architecture for collaborative DDoS mitigation with smart contracts. In: IFIP international conference on autonomous infrastructure, management, and security (AIMS 2017). Lecture notes in computer science, vol. 10356, pp. 16–29. Springer, Zurich (2017)

15. Rodrigues, B., Bocek, T., Stiller, B.: Enabling a cooperative, multi-domain DDoS defense by a blockchain signaling system (BloSS). In: Demonstration track, pp. 1–3, Singapore, IEEE, Singapore (2017)
16. Rodrigues, B., Trendafilov, S., Scheid, E.J., Stiller, B.: SC-FLARE: cooperative DDoS signaling-based on smart contracts. In: IEEE international conference on blockchain and cryptocurrency (ICBC 2020), pp. 1–3, IEEE, Toronto, (2020)
17. Gruhler, A., Rodrigues, B., Stiller, B.: A reputation scheme for a blockchain-based network cooperative defense. In: 2019 IFIP/IEEE symposium on integrated network and service management (IM 2019), pp. 71–79, Washington, United States of America (USA) (2019)
18. Killer, C., Rodrigues, B., Stiller, B.: Security management and visualization in a blockchain-based collaborative defense. In: 2019 IEEE international conference on blockchain and cryptocurrency (ICBC), pp. 108–111. IEEE, New York (2019)
19. Killer, C., Rodrigues, B., Stiller, B.: Threat management dashboard for a blockchain collaborative defense. In: 2019 IEEE globecom workshops (GC Wkshps), pp. 1–6. IEEE, New York (2019)
20. Rodrigues, B., Eisenring, L., Scheid, E., Bocek, T., Stiller, B.: Evaluating a blockchain-based cooperative defense. In: 2019 IFIP/IEEE symposium on integrated network and service management (IM 2019), pp. 533–538, Washington, United States of America (USA) (2019)
21. Benet, J.: IPFS-content addressed, versioned, P2P file system. arXiv preprint arXiv:1407.3561 (2014)
22. Keromytis, A., Misra, V., Rubenstein, D.: SOS: secure overlay services. ACM SIGCOMM Comput. Commun. Rev. **32**(4), 61–72 (2002)
23. Papadopoulos, C., Lindell, R., Mehringer, J., Hussain, A., Govindan, R.: COSSACK: coordinated suppression of simultaneous attacks. In: Proceedings DARPA information survivability conference and exposition, vol. 1, pp. 2–13. IEEE, New York (2003)
24. Oikonomou, G., Mirkovic, J., Reiher, P., Robinson, M.: A framework for a collaborative DDoS defense. In: 2006 22nd annual computer security applications conference (ACSAC'06), pp. 33–42. IEEE, New York (2006)
25. CloudFare. CloudFlare advanced DDoS protection (2016)
26. Khalimonenko, A., Kupreev, O., Badovskaya, E.: DDoS attacks in Q1 2018. https://securelist.com/ddos-report-in-q1-2018/85373/, April 2018. last visit March 6 (2020)
27. Zhang, G., Parashar, M.: Cooperative defence against DDoS attacks. J. Res. Pract. Inf. Technol. **38**(1), 69–84 (2006)
28. Velauthapillai, T., Harwood, A., Karunasekera, S.: Global detection of flooding-based DDoS attacks using a cooperative overlay network. In: 2010 4th international conference on network and system security (NSS), pp. 357–364. IEEE, New York (2010)
29. Steinberger, J., Kuhnert, B., Sperotto, A., Baier, H., Pras, A.: Collaborative DDoS defense using flow-based security event information. In: NOMS 2016–2016 IEEE/IFIP network operations and management symposium, pp. 516–522 (2016)
30. Sahay, R., Blanc, G., Zhang, Z., Debar, H.: Towards autonomic DDoS mitigation using software defined networking. In SENT 2015: NDSS workshop on security of emerging networking technologies. Internet society (2015)
31. Ioannidis, J., Bellovin, S.M.: Implementing PushBack: router-based defense against DDoS attacks (2002)
32. Mortensen, A., Andreasen, F., Reddy, T., Gray, C., Compton, R., Teague, N.: Distributed-denial-of-service open threat signaling (DOTS) architecture. Internet-draft draft-ietf-dots-architecture-06, internet engineering task force. Work in Progress (2018)
33. Rodrigues, B., Stiller, B.: Cooperative signaling of DDoS attacks in a blockchain-based network. In: Proceedings of the ACM SIGCOMM 2019 conference posters and demos, SIGCOMM Posters and Demos '19, pp. 39–41, ACM, New York (2019)
34. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper **151**, 1–32 (2014)
35. Bada, M.: Sadie Creese. Chris Mitchell, and Elizabeth Phillips. Improving the Effectiveness of CSIRTs, Michael Goldsmith (2014)
36. Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Sirer, E.: On scaling decentralized blockchains. In: International conference on financial cryptography and data security, pp. 106–125. Springer, Berlin (2016)
37. Decker, C., Wattenhofer, R.: Information propagation in the bitcoin network. In: IEEE P2P 2013 proceedings, pp. 1–10. IEEE, New York (2013)

38. Zheng, Z., Xie, S., Dai, H.N., Chen, X., Wang, H.: Blockchain challenges and opportunities: a survey. Int. J. Web Grid Serv. **14**(4), 352–375 (2018)
39. Skierka, I., Morgus, R., Hohmann, M., Maurer, T.: CSIRT basics for policy-makers. Types & culture of computer security incident response teams, the history (2015)
40. PolySwarm. A decentralized cyber threat intelligence market (2019)
41. Just, S., Premraj, R., Zimmermann, T.: Towards the next generation of bug tracking systems. In: 2008 IEEE symposium on visual languages and human-centric computing, pp. 82–85. IEEE, New York (2008)
42. Henshel, D., Cains, M., Hoffman, B., Kelley, T.: Trust as a human factor in holistic cyber security risk assessment. Procedia Manuf. **3**, 1117–1124 (2015)
43. Hawlitschek, F., Notheisen, B., Teubner, T.: The limits of trust-free systems: a literature review on blockchain technology and trust in the sharing economy. Electron. Commer. Res. Appl. **29**, 50–63 (2018)
44. Um, T.W., Lee, G.M., Choi, J.K.: Strengthening trust in the future social-cyber-physical infrastructure: an ITU-T perspective. IEEE Commun. Mag. **54**, 36–42 (2016)
45. Pranata, I., Skinner, G., Athauda, R.: A holistic review on trust and reputation management systems for digital environments. Int. J. Comput. Inf. Technol. **1**, 44–53 (2012)
46. Kim, J., Yoon, Y., Zo, H.: Why people participate in the sharing economy: a social exchange perspective. In: PACIS, pp. 76 (2015)
47. Yaobin, L., Zhao, L., Wang, B.: From virtual community members to C2C E-commerce buyers: trust in virtual communities and its effect on consumers' purchase Intention. Electron. Commer. Res. Appl. **9**(4), 346–360 (2010)
48. Stephan, M., Bruno, R., Eder, S., Salil, K., Burkhard, S.: Toward mitigation-as-a-service in cooperative network defenses. In 2018 IEEE 4th international conference on big data intelligence and computing and cyber science and technology congress (CyberSciTech 2018), pp. 362–367, Athens, Greece (2018)
49. Yao, W. Julita, V.: Trust and reputation model in peer-to-peer networks. In: Proceedings third international conference on peer-to-peer computing (P2P2003), pp. 150–157, 00604 (2003)
50. Cai, Y., Zhu, D.: Fraud detections for online businesses: a perspective from blockchain technology. Fin Innov **2**(1), 20 (2016)
51. Cryptography developers. Python cryptography library. https://cryptography.io/en/latest/ (2017). Accessed 28 July 2018
52. Percival, C.: Cryptographic right answers. http://www.daemonology.net/blog/2009-06-11-cryptographic-right-answers.html (2009). Accessed 28 July 2018
53. Protocol Labs. IPFS stream security transport (libp2p-secio). https://github.com/libp2p/go-libp2p-secio (2018). Accessed 29 July 2018
54. Rinkeby. Rinkeby testnetwork, voluntarely listed full nodes (2019)
55. Scheid, E., Rodrigues, B., Stiller, B.: Toward a policy-based blockchain agnostic framework. In: IFIP/IEEE symposium on integrated network and service management (IM 2019), pp. 609–613, Washington, DC, USA (2019)
56. Scheid, E., Hegnauer, T., Rodrigues, B., Stiller, B.: Bifröst: a modular blockchain interoperability API. In: IEEE conference on local computer networks (LCN 2019), pp. 332–339, Osnabrück, Germany (2019)
57. Franco, M., Rodrigues, B., Stiller, B.: MENTOR: the design and evaluation of a protection services recommender system. In: 2019 15th international conference on network and service management (CNSM), pp. 1–7 (2019)

Bruno Rodrigues is a Junior Researcher and PhD student in Informatics currently pursuing his PhD degree at the University of Zürich UZH, Switzerland, within the Communication Systems Group CSG of the Department of Informatics IfI under the supervision of Prof. Dr. Burkhard Stiller. He received his MSc from the Polytechnic School of University of São Paulo, Brazil, in 2016, where he worked on research projects in partnership with Ericsson Research focused on network management based on SDN and energy efficiency. Bruno focuses his research on collaborative network defenses based on Blockchain, working on research projects like CONCORDIA in the scope of cybersecurity and PasWITS.

Eder Scheid is a Junior Researcher pursuing his PhD since December 2017 under the supervision of Prof. Dr. Burkhard Stiller at the University of Zürich UZH, Switzerland, within the Communication Systems Group CSG of the Department of Informatics IfI. Eder holds an MSc degree in Computer Science from the Federal University of the Rio Grande do Sul (UFRGS), Brazil, which he obtained in 2017 under the supervision of Prof. Dr. Lisandro Zambenedetti Granville. His master's thesis was entitled “INSpIRE: an Integrated NFV-baSed. Intent Refinement Environment”.

Christian Killer joined the Communication Systems Group CSG, Department of Informatics IfI at the University of Zürich UZH, Switzerland, in February 2019 as a Junior Researcher and PhD student in Informatics to obtain his PhD degree under the supervision of Prof. Dr. Burkhard Stiller. He finished his MSc Degree in Informatics at the University of Zürich UZH, focusing on Security Management and Visualization in a Blockchain-based Collaborative Defense. During his Master Studies he also completed a Master Project on “Provotum–Privacy, Verifiability, and Auditability in Blockchain-based E-Voting”. Christian focuses his research on the security of electoral processes and blockchain-based remote electronic voting systems.

Muriel Franco is a Junior Researcher and PhD student in Informatics under the supervision of Prof. Dr. Burkhard Stiller at the University of Zürich UZH, Switzerland, within the Communication Systems Group CSG of the Department of Informatics IfI. Since September 2018 Muriel is working in Zürich on cybersecurity, economics, blockchains, Software-defined Networking (SDN), and Network Function Virtualization (NFV), participating and driving the work of the CONCORDIA project within a team of networking, security, and economic researchers. Besides that, from 2017 to 2020, Muriel developed jointly a federated ecosystem for offering, distributing, and execution of Virtual Network Functions (FENDE project). Muriel holds an MSc from 2017 in Computer Science from the Federal University of the Rio Grande do Sul (UFRGS), Brazil, under the supervision of Prof. Dr. Lisandro Zambenedetti Granville and obtained a BSc from 2014 in Computer Science from the Federal University of Pelotas (UFPEL), Brazil.

Burkhard Stiller received the Informatik-Diplom (MSc) in Computer Science and the Dr. rer.-nat. (PhD) degree from the University of Karlsruhe, Germany, in 1990 and 1994, respectively. In his research career he was with the Computer Lab, University of Cambridge, U.K. (1994–1995), ETH Zürich, Switzerland (1995–2004), and the University of Federal Armed Forces Munich, Germany (2002–2004). Since 2004 he chairs the Communication Systems Group CSG, Department of Informatics IfI, University of Zürich UZH, Switzerland. Besides being a member of the editorial board of the IEEE Transactions on Network and Service Management, Springer's Journal of Network and Systems Management, and the KICS' Journal of Communications and Networks, Burkhard is the past Editor-in-Chief of Elsevier's Computer Networks journal. His main research interests are published in well over 300 research papers and include systems with a fully decentralized control (Blockchains, clouds, peer-to-peer), network and service management (economic management), Internet-of-Things (security of constrained devices, LoRa), and telecommunication economics (charging and accounting).

Affiliations

Bruno Rodrigues[1] · Eder Scheid[1] · Christian Killer[1] · Muriel Franco[1] · Burkhard Stiller[1]

Eder Scheid
scheid@ifi.uzh.ch

Christian Killer
killer@ifi.uzh.ch

Muriel Franco
franco@ifi.uzh.ch

Burkhard Stiller
stiller@ifi.uzh.ch

[1] Communication Systems Group CSG, Department of Informatics IfI, University of Zürich UZH, Binzmühlestrasse 14, 8050 Zurich, Switzerland