Department of Informatics

Blockchain Signaling System (BloSS)

Dissertation submitted to the Faculty of Business,
Economics and Informatics
of the University of Zurich

to obtain the degree of
Doktor der Wissenschaften, Dr. sc.
(corresponds to Doctor of Science, PhD)

presented by
Bruno Bastos Rodrigues
from
Brasília, Distrito Federal, Brazil

approved in February 2021

accepted at the request of
Prof. Dr. Burkhard Stiller
Prof. Radu State, PhD

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.

Cooperative defense systems face many challenges, such as deployment complexity due to high coordination overhead, reliance on trusted and stable channels for communication and the need for effective incentives to bolster cooperation among all involved parties. These 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.

Driven by challenges imposed in a cooperative network defense, Blockchain Signaling System (BloSS) is presented as 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 local and global experiments, without the burden to maintain, design, and develop special registries and gossip protocols.

Evaluation results based on the local and global prototyping of BloSS highlight its effectiveness in signaling information of large-scale DDoS attacks. The world-wide scale evaluation experimenting the interaction between Autonomous Systems (AS) victim of DDoS attack and ASes acting as mitigators, presented an average of 97 seconds to complete all eleven possible outcomes of the BloSS protocol. The reputation assessment, based on the transparency of actions carried out on BC using Beta reputation and individual thresholds of trust for each member, showed that the defined protocol is capable of punishing malicious providers and benefiting providers by acting honestly.

The definition of contracts in BloSS 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. Overall, the main achievement and advantages reached with the design, prototypical implementation, and evaluation of BloSS include *(a)* the use of an existing 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 Software-Defined Networking (SDN) and BC.

Kurzfassung

Distributed Denial-of-Service (DDoS)-Angriffe sind eine der Hauptbedrohungen für Anbieter von Kommunikationsdienstleistungen. Wenn ein Angriff technisch ausgereift ist und keine direkten Gegenmaßnahmen zur Verfügung stehen, wird die gemeinsame Nutzung von Hardware und Verteidigungsfähigkeiten zu einer zwingenden Alternative. Das zukünftige Netzwerk- und Dienstmanagement kann seine Operationen auf vollständig verteilte Systeme stützen, um hochgradig verteilte DDoS-Angriffe zu neutralisieren. Eine kooperative Verteidigung ermöglicht die Kombination von Erkennungs- und Abwehrfähigkeiten, die Reduzierung des Overheads an einem einzigen Punkt und die Blockierung bösartigen Datenverkehrs in der Nähe seiner Quelle.

Kooperative Verteidigungssysteme sind mit vielen Herausforderungen konfrontiert, wie z.B. der Komplexität des Einsatzes aufgrund des hohen Koordinationsaufwands, der Abhängigkeit von vertrauenswürdigen und stabilen Kommunikationskanälen und der Notwendigkeit wirksamer Anreize zur Förderung der Zusammenarbeit zwischen allen beteiligten Parteien. Diese Herausforderungen beeinträchtigen den weit verbreiteten Einsatz bestehender kooperativer Verteidigung: *(a)* hohe Komplexität von Einsatz und Koordination, *(b)* Notwendigkeit einer vertrauenswürdigen und sicheren Kommunikation, *(c)* Mangel an Anreizen für die Anbieter von Dienstleistungen zur Zusammenarbeit und *(d)* Feststellung, wie der Betrieb dieser Systeme durch unterschiedliche Gesetze, Regionen und Länder beeinflusst wird.

Von denjenigen Herausforderungen bestimmt, die durch eine kooperative Netzwerkverteidigung entstehen, wird das Blockchain Signaling System (BloSS) als eine effektive und alternative Lösung für das Sicherheitsmanagement, insbesondere für die kooperative Verteidigung, vorgestellt, indem Blockchains (BC) und Software-Defined Networks (SDN) für den Austausch von Angriffsinformationen, den Austausch von Anreizen und die Verfolgung der Reputation auf vollständig verteilte und automatisierte Weise genutzt werden. BloSS prototypisch entwickelt und durch lokale und globale Experimente evaluiert, ohne die Notwendigkeit, Betriebskosten spezieller Register bzw. Datenbanken und Gossip-Protokolle vorsehen zu müssen.

Die Evaluierungsergebnisse auf der Grundlage der lokalen und globalen Prototypisierung von BloSS unterstreichen seine Wirksamkeit bei der Signalisierung von Informationen über groß angelegte DDoS-Angriffe. Die weltweite Auswertung, bei der die Interaktion zwischen Autonomen Systemen (AS), welche typischerweise Opfer von DDoS-Angriffen sind, und AS, die als Mittelsman fungieren, getestet wurde, ergab einen Durchschnitt von 97 Sekunden, um alle elf möglichen Endzustände des BloSS-Protokolls zu erreichen. Die Reputationsbewertung, die auf der Transparenz der auf den BCs durchgeführten Aktionen unter Verwendung der Beta-Reputation und individueller Vertrauensschwellen für jedes Mitglied basierte, zeigte, dass das definierte Protokoll in der Lage ist, böswillige Anbieter zu bestrafen und Anbieter durch ehrliches Handeln zu begünstigen.

Die kooperative Logik auf der Grundlage von BCs erlaubt die Definition von Verträgen in BloSS und ermöglicht die Stärkung des Vertrauens zwischen den kooperativen Akteuren aufgrund ihres transparenten Austauschs ausgewählter Informationen und entsprechender Anreize auf einer Pro-Anfrage-Basis. Insgesamt gesehen gehören zu den wichtigsten Errungenschaften und Vorteilen, die mit dem Entwurf, der prototypischen Implementierung und der Evaluierung des BloSS erreicht wurden, die Nutzung einer bestehenden verteilten Infrastruktur, der BCs, dem Eintrag von IP-Adressen in White- bzw. Black-Listen und zur Verteilung von Anreizen in Zusammenhang mit Gegenmaßnahmen. Darüber hinaus erlaubt es den Nachweis des einsatzfähigen Konzeptes für eine kooperative, betriebsbereite und effiziente Dezentralisierung von DDoS-Mitigaton-Diensten und eine Kompatibilität mit bestehenden Netzwerkinfrastrukturen auf der Basis von Software-Defined Networking (SDN) und BCs.

Acknowledgments

First of all, I would like to thank God for being able to do what I love. I thank all my family who provided me with all the necessary support before entering academic life, and friends that I could always count on in my free time to talk and share beers and coffees.

Thanks to my advisor Prof. Dr. Burkhard Stiller, who guided me during all the doctorate stages involving the entire scope of tasks (*i.e.*, teaching, presentations, guiding students) performed during the period, offering advice and freedom to suggest ideas and research paths. I would also like to thank my advisors of the master's, Prof. Dr. Tereza Cristina Melo de Brito Carvalho, and bachelor's degrees, Prof. Dr. Charles Christian Miers, who provided me with the necessary background and support to keep following the academic path.

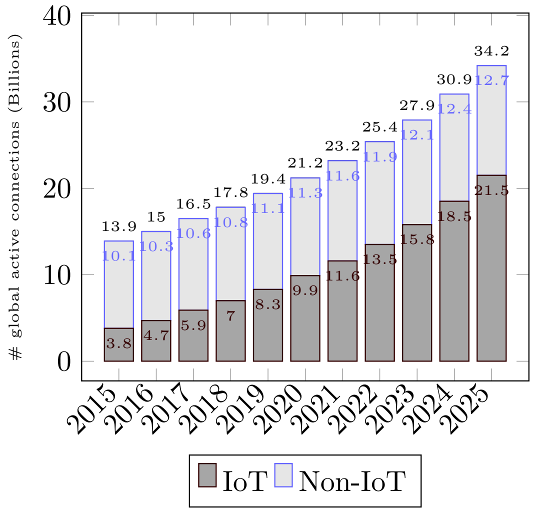
I am very grateful to many people who have helped me directly or indirectly with my thesis work. All the CSG members that helped my work with many suggestions and discussions on how to improve the various specific parts, ranging from helping to build a cluster of Raspberry Pi's to how to run many of the experiments in the thesis. Also, I want to thank the many students who contributed to this thesis conducting and extending experiments, and proposing improvements to my initial BloSS prototype, which was gracefully ported to virtual machines enabling world-wide experiments.

Contents

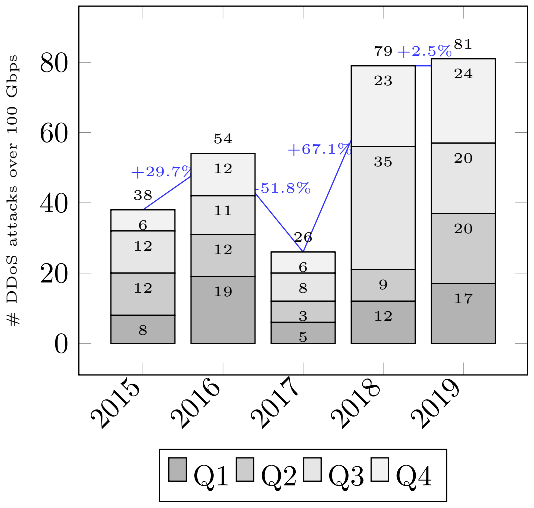
1
Introduction

The technological evolution and the rapid growth of the Internet have built a digitally networked society, which is indispensable for communication and interaction on a planetary scale. As the number of connected devices (portable and stationary) increases, the complexity of systems providing content for these devices or the communication infrastructure, increased in a similar proportion to support the growing volume of traffic [3]. As a consequence, these complex distributed systems are subject to several types of failures and threats that can compromise, for example, entire societies whose Critical Infrastructures (CI) are connected to the Internet [70].

Among the various threats to the Internet and its underlying systems, Distributed Denial-of-Service (DDoS) attacks are one of the biggest threats to the availability of services on the Internet. A DDoS attack is defined as a coordinated attempt to make a target system's resources unavailable to legitimate users by bombarding them with high-bandwidth traffic, or they directly target the computing resources through malformed packets designed to generate overwhelming computational loads [124, 125]. In both cases, defending against these attacks at the ingress point of the traffic quickly becomes infeasible due to the constrained resources available for the defense at the target. In contrast to this centralized approach, a decentralized defense, where the DDoS mitigation takes place at the egress of the attack on the individual targets is preferable [265].



(a) World-wide Number of Connected IoT and Non-IoT Devices. [96]



(b) Large-scale DDoS Attacks Over 100 Gbps Traffic. [4, 188]

Figure 1.1: (a) Number of Connected Devices and (b) Number of Large-scale DDoS Attacks

Although being a widely known type of attack, it remains one of the significant causes of concern for service providers around the world and nations concerned with its CI's [138]. For example, the United States Department of Defense (DoD) declares the cyberspace as the fifth dimension of defense areas, complementing the traditional land, water, sea, air warfare dimensions [146]. Behind the various reasons that motivate a DDoS attack, is the increasing number of, often insecure, devices connected to the Internet. As observed in Figure 1.1a, the number of IoT devices is surpassing the number of non-IoT devices (*e.g.*, mobile phones, laptops, computers, and others), wherein such devices ranges from small sensors to baby cameras and home gateways, and are the main target of software that systematically exploit vulnerabilities to infect thousands of such devices [3, 192]. Such software is termed *malware*, and it is defined as a malicious piece of code using resources of its host system to perform undesirable or malicious activities [83]. In a DDoS context, once a device is infected by malware, it defined as a *bot*, a device running a software whose resources are unintentionally used to execute commands. Moreover, naturally, a Botnet is a system composed of a cluster of Bots controlled by at least one attacker.

Botnets taking advantage of the lack of security of these IoT devices are the primary basis for these large-scale attacks (*cf.*, Figure 1.1b). While 2018 registered the most massive DDoS attack in terms of traffic volume, peaking 1.7 TBps on GitHub servers, the frequency of DDoS attacks also increased more than 29.7% times between 2015 and 2016 [117]. However, the almost 52% reduction in the number of large-scale attacks from 2016 to 2017 did not denote a reduction in attacks. Akamai reported that the majority of attacks were between 250 Mbps and 1.25 Gbps in traffic volume. However, in 2018 there was an alarming increase of 67.1% in the number of large-scale attacks due to the different Mirai variations and reflection attacks based on Memcached [117].

The most prominent example is the Mirai botnet, which exploits default and weak security credentials to take control of the host and spreads itself to other devices. Mirai was the botnet responsible for marking the transition toward an era of super attacks, in which the traffic volume at the target system often surpasses 1 TBps [3]. The first appearance of Mirai was in September 2016 when it peaked 623 Gbps of traffic volume in an attack against Krebs Security. Akamai, the company hosting the Web site, had to shut down this site because defending it during three days became too costly. It was reported that so many devices were used that attackers did not have to use any sophisticated strategy. A month later, in October 2016, the attack on DynDNS peaked 1.2 TB/s, resulting in the unavailability of significant Internet platforms and services (*e.g.*, Twitter, GitHub, PayPal, Spotify, and others) by rendering their Domain Name System (DNS) servers unavailable [192].

The United States of America (U.S.A.) released in 2018 an estimate of costs related to malicious cyber activities (including DDoS attacks) of around 57 and 109 billion USD for incidents appearing only in 2016 [252]. These costs involve not only direct losses for a target and economically linked companies, but also incurs in indirect costs involving the maintenance and improvement of systems security. Further, Gartner [154] corroborates with the U.S.A. estimate, predicting in 2018 a cost of 114 and 124 billion USD in 2019, representing an increase of 8% for one country only. While cost numbers are not precise on a global scale, there exist estimates, such as [155], that predict costs related to cybersecurity (not only including DDoS) activities to exceed 1 trillion USD cumulatively for the five years from 2017-2021, taking into account the growing number of Internet of Things (IoT) devices that empower DDoS attacks.

As a consequence, the cyber-security community (industry and academy) is becoming increasingly concerned with DDoS attacks dimensions, and the possibility of those being used as a cyber-weapon on systems that are crucial for the functioning of the state and society, such as logistics, health, and energy. For example, the US Department of Defense declares the cyber-space as the fifth dimension of their national security along with operations on air, land, sea, and space. Therefore, as observed in academia and industry trends, as DDoS attacks become progressively more distributed and coordinated, the defense from such attacks likewise requires distribution and coordination.

1.1 Cooperative Network Defenses

Different detection and mitigation methods are available to prevent or reduce DDoS attacks damage. A typical implementation is called on-premises defense, which is implemented by the target system based on dedicated ASIC-based (Application-specific Integrated Circuit) appliances to analyze flow records exported from edge routers and to filter malicious traffic or perform load balancing. Alternatively, there are *off-premises* protection services that can be distributed (mostly cloud-based) or decentralized (cooperative). While the former serves as a proxy receiving, analyzing, and redirecting traffic to the target, which delegate detection and mitigation tasks to the protection provider (*e.g.*, Akamai [3] or CloudFlare [43]), the latter is a decentralized approach typically implemented as a cooperative overlay network.

A cooperative network defense allows to combine detection and mitigation capabilities of different domains, reducing the overhead at a single point, and block malicious traffic near its source. However, there is still no widespread deployment of such a cooperative defense system. As identified in [183, 265], the main challenges of existing approaches are identified as: (1) the high complexity of operation and coordination; (2) the need for trusted and secure communication; (3) lack of incentives for the service providers to cooperate; (4) understand how the operations of these systems are affected by different legislation, regions, and countries.

The literature contains many industry and academic proposals. Secure Overlay Services (SOS) [103], COSSACK [175], and DefCOM [169] 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 [169, 175], or requiring the sources to be registered [103], thus, presenting a high complexity of coordination and operation (1).

More recent approaches, such as CoFence [195] and Bohatei [69], are based on relatively new technologies. For example, NFV (Network Function Virtualization) and SDN (Software-Defined Networking), that can reduce the complexity of coordination and operation. However, other challenges, such as economic (2), and social (3) are not fully addressed. Thus, a technical solution for a cooperative defense should avoid not only additional hardware and software costs, but also be simple to deploy and operate. Further, it should also encompass the support for incentives that can be safely distributed among participants and that legal/conformity options can be selected to (4), for example, restrict operation to specific regions/countries or members. These challenges are summarized in the following categories:

1. **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 operates based on existing standards, avoiding to impose additional software or hardware requirements.

2. **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. Thus, the challenge lies in how to establish and manage trust in a cooperative environment.

3. **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.

4. **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.

1.2 Blockchain-based Cooperative Defense

Blockchain (BC) technology is the core element of BloSS not only to exchange information about attacks but also to distribute the necessary incentives. While it simplifies various technical and economic aspects, social and legal aspects may be impaired by its use. All information in a BC is immutable and transparent to all participants of the cooperative defense, and while transparency has a positive impact in trust, information leaks about signaled attacks could result in potential damages to the public image of a domain. Thus, there is a need to build a consensus among participants about the confidentiality of information exchanged in the BC and eventual penalties in case of information leaks.

Smart Contracts (SC) are another important element, which are computer programs running within the BC. SCs are formal algorithms comprising of agreements between stakeholders being automatically enforced, verified, and replicated by nodes composing the BC to ensure its permanent storage and high obstacles to manipulate the contract's content [21]. Combined, SCs running on the BC and decentralized applications (dApp) are BloSS main components. While SCs describe how information is exchanged among service providers, the dApp's contains the parameters that define how a service provider interacts in a cooperative defense.

Advantages of using BC and SCs in a cooperative defense are [200]: (a) to make use of an already existing infrastructure to distribute rules without the need to build specialized registries or other distribution mechanisms/protocols, (b) to apply rules across multiple domains, which means that even if the AS (Autonomous System) of the victim is not applying these rules, traffic can still be filtered, and (c) the victim or its AS can control which customers get blocked. The only central element remaining is to show proof of IP ownership.

Therefore, while BC can (1) reduce the complexity of operation and coordination by using existing infrastructure to distribute rules without specialized registries or protocols, it also can foster a (2) trusted cooperation due to its transparency and decentralized characteristics. Also, it can provide financial incentives (3), which foster cooperative behavior among service providers, [198]. Thus, BC capabilities can be leveraged for signaling mitigation requests across a decentralized network in a similar approach than DefCOM [169] and serve as an immutable platform for the exchange of mitigation services, where participants express their needs in forms of incentives.

1.3 Research Questions

The challenges briefly described in Sections 1.2 and 1.1 (and further detailed in the course of this thesis as in the overview of the state-of-the-art in Chapter 3) reinforce the opportunity for the proposal of coping solutions with the challenges of collaborative defenses. Henceforth, the main goal of this thesis is to provide a cooperative defense approach providing a technical answer for each of these categories combined into a single system. Based on the challenges mentioned above, the following Research Questions (RQ) drive this thesis.

RQ1: Can a BC-based cooperative system reduce operation and deployment complexities? The proposed approach shall be simple to deploy and operate, aiming to avoid extra hardware or software requirements on the underlying network infrastructure. Also, it should not only perform the signaling of attacks but also provide a platform for social, economic, and legal aspects to be implemented.

RQ2: How to balance transparency and privacy in a cooperative system, increasing trust among cooperative members? In RQ2, the proposed solution shall punish malicious behavior of its members, preventing false-reporting and free-riding (*i.e.*, service providers that only request defense without contributing). Also, it defines the confidentiality requirements imposed on the technical challenge concerning messages privately exchanged between its members to perform the signaling of an attack.

RQ3: How to provide financial incentives to foster cooperative behavior among its members? RQ3 concerns the economic impact and how to provide the necessary incentives to cover capital and operational expenses. Thus, the proposed method should provide a platform based on BC, enabling the exchange of incentives to boost cooperative behavior.

RQ4: How to ensure compliance across different jurisdictions? RQ4 refers to enable or disable its operation in certain regions, or countries, or the interaction with selected participants to comply with organizational/legal obligations.

To answer the posed questions, a referenced research method is used to: (a) overview the core concepts on which this thesis is based, (b) analyze the state-of-the-art concerning cooperative defenses, listing their characteristics and proposals, as well as differentiating them from what is proposed in this thesis. Then, the applied research consists of the design, prototyping, and validation of BloSS in order to verify whether the proposed system satisfied the research questions.

1.4 Methodology

This work is based on referenced and applied research, in which the theoretical background related to the state-of-the-art in DDoS attacks and cooperative defenses was obtained by a qualitative research. The applied research involves the steps related to the design and prototyping stages of BloSS applying techniques and concepts related to network and decentralized systems. To summarize, the research methodology employed consists in the following steps:

1. **Literature Review and Analysis**: analyze the state-of-the-art in DDoS attacks, cooperative defenses, and concepts related to BC and decentralized systems.

2. **Design and Architecture Definition**: design an architecture and method able to fulfill the objectives and answer the research goals. Such architecture should consider the the capabilities and requirements imposed by BC and SCs.

3. **Prototyping**: Implement and troubleshoot the designed elements. Thus, this stage involves the implementation and troubleshooting of BloSS, considering that the prototyping process reveals conditions not observed in the proposal.

4. **Evaluation**: Evaluate BloSS in order to determine whether the proposed goals and questions were met and reports the benefits and related problems. In this sense, the experiments and evaluations take place at different stages, starting from a simulated prototyping until the deployment and experimentation of the prototype in real environments at different scales (*i.e.*, locally at a dedicated cluster and globally based on nodes across the world).

1.5 Research Methods

This thesis consist of research in which, besides the referenced research, studies related to the focus area and participation in correlated projects were developed. To verify whether the proposed system satisfied the research questions, the validation stage goes beyond the evaluation of its technical, quantifiable aspects. It also relies on a qualitative evaluation based on factors demanding interaction with external academia or industry partners.

At first, the referenced research method is used to understand concepts and requirements to design the system addressing the research questions (*cf.*, Chapters 2 and 3). Then, the applied research consists of the design, prototyping, and validation of a prototype in order to evaluate the concept proposed. Also, the BloSS prototype is evaluated quantitatively (Subsection 1.5.1) and qualitatively (Subsection 1.5.2). While the quantitative evaluation aims to analyze performance in terms of end-to-end latency for signaling white or blacklisted IP addresses, the qualitative evaluation should answer based on existing systems and approaches whether the BloSS satisfies the research questions.

1.5.1 Quantitative

The goal is to measure how the system behaves in a real world deployment to create a basis for comparative analysis when data is provided by related work. For example, the system performance should measure the latency for signaling DDoS attacks, *i.e.*, how much time is required to propagate the information about an ongoing attack on all nodes involved in the mitigation request. This analysis is particularly important because BC-based approaches typically introduce a delay to create and replicate blocks in the network. Thus, the proposed approach should ideally be able to signal attacks in a timely manner so that cooperative defense become effective. Also, performance analysis should assess the impact on hardware resources, such as network traffic and CPU consumption, memory and storage (*i.e.*, relevant for answering RQ1).

The quantitative evaluation is essential as a basis to a qualitative evaluation. For example, if the DDoS attack signaling time (*i.e.*, latency to report an attack) is longer than the duration of the attack, the BC-based approach becomes inefficient. Conversely, the quantitative evaluation cannot determine how effective is the approach, and a comparison with related work or a user evaluation may be required.

1.5.2 Qualitative

The qualitative evaluation examine whether the research questions were satisfied. It may consider a comparison with related work when comparable data is provided or a user-oriented perception (interviews, questionnaire) over the technical aspects. For example, a claim about the system deployment/operation complexities (RQ1) is based on the system performance (*i.e.*, end-to-end latency for signaling of black or whitelisted addresses).

RQ2 is evaluated similarly. Once security is often based on multiple non-quantifiable dimensions, a statement on the system security should be supported by a user perception concerning CIA (Confidentiality, Integrity, and Availability). Also, a comparative analysis based on related work can be possible analyzing which requirements, methods, and tools could be deployed to ensure that data is confidential, not tampered, and available when requested. RQ3 (concerning financial incentives) and RQ4 (legal compliance) have their qualitative analysis simplified by the quantitative evaluation. Since RQ1 and RQ2 are related to the addition of features in the cooperative defense, the challenge is to verify the impact of the implementation of these features in RQ1 (complexity of deployment and operation).

1.6 Thesis Contributions

This thesis' structure is defined based on these research questions, which driven by the specific research questions posed, determine contributions to the state-of-the-art in cooperative defenses. The **key contributions** of this thesis can be summarized as follows:

- A survey of cooperative network defenses approaches and studies, providing an overview of the major aspects of a cooperative defense and showing the gap in existing approaches.
- The Blockchain Signaling System (BloSS), an cooperative defense approach based on BC and SCs, is presented for the design of a collaborative defense whose technical requirements do not impose restrictions on the underlying systems.
- BloSS is evaluated considering the different relevant aspects composing a cooperative defense, showing its simplicity and effectiveness to signal information related to DDoS attacks, and to distribute the incentives in a decentralized and transparent way.

Those contributions improve the understanding concerning cooperative defenses, providing a solution to apply this understanding in practice, and giving insight into the benefits of decentralized cooperative defenses. BC systems can foster trusted cooperation because they not only operate on the principle of decentralization, helping to eliminate third-party intermediaries but also to provide incentives for stimulating cooperative behavior among service providers. Similarly, this thesis leverage BC capabilities for the signaling mitigation requests and to serve as an immutable platform for the exchange of mitigation services, where each participant can express their needs in forms of incentives.

1.7 Thesis Outline

The core chapters of this thesis and its methodology are highlighted in Figure 1.2. Additionally, **Chapter 7** closes this thesis summarizing the achievements, contributions, answers to research questions, and future work.

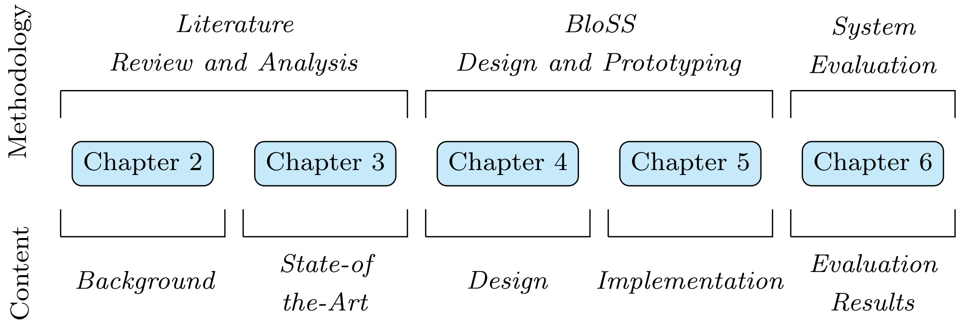


Figure 1.2: Organization and Methodology

- **Chapter 2**: Provides the background on the major concepts involved the thesis. Thus, are presented concepts concerning DDoS attacks and defenses, as well as an overview on major BC elements and operation concepts.
- **Chapter 3**: Reviews the literature related to cooperative defenses in order to reinforce the need for approaches capable of solving the listed challenges. Therefore, the chapter presents an updated review of the works analyzing them from the point of view of technological, social, economic, and legal challenges.

- **Chapter 4**: Introduces the design of the cooperative signaling protocol that defines the sequence of steps in the communication between a target (*i.e.*, DDoS attack victim) and a cooperative mitigator (*i.e.*, third-party involved in the cooperation). Also, this chapter reveals how the main on-chain design aspects are addressed, involving how BloSS strikes a balance between performance and confidentiality of the shared data.

- **Chapter 5**: Addresses aspects of the decentralized application that communicates with the on-chain protocol, as well as the network management system. This chapter complements the cooperative signaling protocol by presenting as data transmitted off-chain are made secure, including options for each participant in the collaborative defense including which members in which regions the interactions are made.

- **Chapter 6**: Presents evaluations and discussions of different aspects of BloSS such as the reputation system, off-chain signaling and cooperative signaling protocol. These assessments include tests in different scopes such as simulations (assessing correctness), local (deployed in local clusters or networks), and global to assess the performance in an environment close to the real world conditions.

- **Chapter 7**: Concludes the thesis summarizing the achievements, contributions, and future work.

- **Appendices**: provide additional information on major elements included in the thesis. These are included as follows:

  - **Appendix A**: Presents detailed description of the works presented in the state-of-the-art of collaborative DDoS defense.
  - **Appendix B**: Shows implementation details (*i.e.*, listings and respective descriptions) of the SCs used in BloSS.
  - **Appendix C**: Presents details of the world-wide BloSS evaluation described in Chapter 6.
  - **Appendix D**: Reveals in full level of detail the evaluation of the reputation system used in BloSS.
  - **Appendix E**: Describes the list of vulnerabilities in smart contracts used as a basis for evaluating BloSS contracts.
  - **Appendix F**: Details scientific publications and bachelor's and master's theses from this thesis, contributing directly or indirectly to obtain the objectives of the thesis.

2 Theoretical Foundations

This Chapter describes the major concepts involved in this thesis. Since the context of collaborative defenses is multi-disciplinary, involving in addition to purely technical elements for developing a system, it is also necessary to understand concepts about fostering trust and incentives for collaboration to be encouraged. Thus, herein are presented concepts concerning DDoS attacks and defenses, Blockchain (BC) elements and operation, and reputation tracking and management. Further, once BC is a fundamental concept in the proposal, design, and evaluation of BloSS, a greater relevance is given to the description of the related concepts in this Chapter.

2.1 Distributed Denial-of-Service Attacks

Different types of Denial-of-Service (DoS) attacks, including the distributed ones, *i.e.*, Distributed Denial-of-Service (DDoS) attacks. Ultimately, all forms of DoS and DDoS attacks share the same goal: denying or degrading legitimate users to access services in a network or server [97]. Thus, the objective involves denigrating a target system's ability to provide a service by limiting (i) access to that service or (ii) the provider's computational capacity. While DoS attacks involve a single attacker, which from a single system seeks to render the target's resources unavailable, DDoS attacks involve multiple attackers distributed in a network aiming to exhaust the target's resources. For instance, whereas DoS attacks can exploit bugs and flaws in an application, DDoS attacks may target to exhaust the target's network bandwidth by flooding packets or server' resources by consuming CPU cycles, random memory, static memory.

In the same way that there are different types and forms of DoS and DDoS attacks, there are also different types of defense for these attacks, either alone or in cooperation. While this section presents the different types of attacks, as well as examples of large-scale DDoS attacks, the different types of defense are covered in the Section 2.2 with a greater focus on the description of cooperative defenses.

2.1.1 Motivations Toward DDoS Attacks

As the number of DDoS attacks increases not only in frequency, but also in their most varied types, their motivations or reasons behind the attacks are still not completely clear. Zargar *et al.* [265] classifies these motivations in five different domains, as outlined below:

1. **Economical**: aims to undermine the ability of other businesses to provide an online service. For example, online stores can choose a significant sales day (*e.g.*, Black Friday or Cyber Monday) to impair the sales capacity of competitors.

2. **Revenge**: it is characterized by frustrated individuals (ex-employees, angry customers, hackers over minor disagreements, among others) with or without knowledge to carry out the attack.

3. **Political**: attacks perpetrated by this reason are motivated by ideological belief reasons, in which attackers can, for example, impair the service of certain companies, which do not align with their ideological beliefs. Examples of ideological attacks are [162, 265]: the Estonian attack in 2007, Chinese hacktivist group on CNN in 2008, WikiLeaks in 2010, and others.

4. **Intellectual challenge**: attackers of this category may comprise from "script kiddies" based on pre-written scripts and tools to launch DDoS attacks until experienced attackers whose goal is to try their skills or new tools.

5. **Fun/Altruistic**: attackers in this category can range from beginners to experienced attackers, who look for fun or pride to make an online product or service unavailable. These can become particularly difficult to have countermeasures in case of experience users since they typically use Botnets to enhance the volume of traffic directed to the victim.

6. **Cyberwarfare**: comprise the range of attacks that are also politically motivated, but performed by (or on behalf of) nations aiming to impair critical services or infrastructures of another

country [178]. Attackers from this category are considered to be well trained individuals (by their own nation or hired from hacktivist groups) with ample resources.

Motivations or incentives to realize a DDoS attack are as complex as the psychology of human relations, which are the basis for several different reasons within these categories outlined above. On the one hand, the most varied reasons are observed for small-scale attacks considering that these often require little technical knowledge from the attackers. On the other hand, it is observed that large-scale attacks are hardly isolated actions from inexperienced attackers [183], due to the complexity of assembling a sufficient number of devices to generate the volume of traffic necessary to prevent a content provider from providing services *e.g.*, the DynDNS attack in 2016 [192]. Thus, large-scale DDoS attacks tend to be the result of group efforts often with economic or cyberwarfare motivations.

2.1.2 Types of DDoS Attacks

Different factors, such as the number of attackers (*i.e.*, devices) involved and the type of resource exploited on the target, can be used to classify DDoS attacks. This subsection follows a general classification, in which the different types of attacks can be classified into three different categories as observed in [44, 64, 94, 133]:

1. **Volumetric**: Include several flood attacks that do not require establishing a connection with the target *i.e.*, stateless. The ultimate goal is to saturate the bandwidth of the attacked site, and magnitude is measured in Bits per second (Bps).

2. **Protocol**: Also known as state-exhaustion attacks typically based on OSI (Open Systems Interconnection) layers 3 and 4 protocols (respectively Network and Transport) [44]. Protocol-based attacks consume actual server resources, or those of intermediate communication equipment, such as firewalls and load balancers, is normally measured in Packets per second (Pps).

3. **Application**: Sometimes it also refers to Layer 7 attacks [3, 44]. Comprised of seemingly legitimate and innocent requests, the goal of these attacks is to crash, for example, the webserver, and the magnitude is measured in Requests per second (Rps).

Although sharing a common objective *i.e.*, to render the target unable to provide service, DDoS attacks have differences concerning their characteristics. While (1) volumetric attacks render target's resources unavailable by flooding their network based on stateless protocols, (2) protocol-based attacks explore particular characteristics of stateful protocols (*e.g.*, Secure Socket Layer - SSL, or Transport Layer Security - TLS) by abusing requests and consuming the target's computational resources.

(3) Application layer attacks are typically measured in Requests per second (Rps) or the number of requests made to an application. This category requires a smaller number of requests in contrast to (1) and (2), and in many cases a single packet exploiting an application vulnerability can be sufficient to render it unavailable. In general, it is focused on overwhelming the CPU and memory.

There are two major differences between (1) and (2) types of attacks. While the first is typically oriented towards the consumption (exhaustion) of the target's network resources, the latter leads to the exhaustion of the target's computational inability to process malformed packets in connection-oriented protocols. Hence, volumetric attacks are measured based on the traffic capacity generated by a unit of time (*e.g.*, Bits-per-second - Bps), attacks on protocols are measured by the number of Packets-per-second (Pps). Table 2.1 summarize the differences between attack types.

Table 2.1: Differences Between Types of DoS and DDoS Attack Types

Type	Metric	Category	Target Resource	Attack	Characteristics	Examples
Volumetric	Bits per-second (Bps)	Stateless	Network	DDoS	High volume of traffic based on Botnets	Flood-based attacks (ICMP, UDP)
Protocol	Packets per-second (Pps)	Stateful	Network and Compute	DoS and DDoS	Malformed packets and Reflection/Amplification	DNS amplification, Ping of Death
Application	Requests per-second (Rps)	Stateful	Compute	DoS	Malicious packets exploring vulnerabilities	SQL injection, XSS

Volumetric attacks are the ones attracting most of the attention of the general public due to the vast amounts of traffic generated in the targets, which has seen during the years of 2017 and 2018 to reach the Terabit per second (Tbps) level (*e.g.*, Memcached attack reached 1.7 Tbps on Github servers in 2018 [117]). The idea behind volumetric attacks is straightforward: to send as much traffic as possible to a target to overwhelm its available incoming bandwidth. Hence, these types of attacks are based on flooding because they exhaust a target's resource with requests, like unwanted pings (ICMP) or UDP traffic.

A report in [4] presents the distribution of attack types in the year of 2020 (*cf.*, Figure 2.1). UDP attacks were no longer the most common individual form of attack in 2020 (with 1.64%), not even nearing the number of SYN DDoS attacks (92.57%). Mixed-method attacks were the largest type of DDoS attack overall, however, and typically involved HTTPS floods and mixed attacks with HTTP elements (0.29%). It is also important to note that these attacks are often combined in multi-vector attacks. For example, volumetric attacks can reach significant numbers in terms of traffic when combined with amplification/reflection techniques (*e.g.*, DNS amplification). The concept of amplification attacks exploits protocol failures in which the size of a response is always larger than the size of the request (*i.e.*, amplifying the attack size).

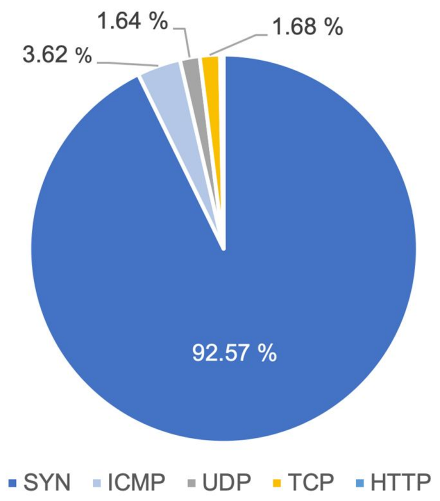


Figure 2.1: SYN Attacks Are The Most Common Form of DDoS Attack as of 2020 [4]

Attack types at the protocol and application level need to establish a connection (*i.e.*, stateful). Both approaches exploit the discrete set of rules defined for the protocol or application to impair a service. A difference between these two attack types is that while protocol attacks often work at layers 3 and 4 of the OSI model (operating in network devices like routers), application-layer attacks operate on layer 7, targeting a service running on edge server (*e.g.*, a web application such as WordPress or Apache). Another difference is that for an application attack, often a single malicious request can impair the application at the target, not requiring the flood of requests on the server.

Ultimately, all attacking types have the same goal — to disrupt an online service. Therefore, an application layer attack may also be multi-vector by combining volumetric and protocol attacks to increase the likelihood of taking a service offline. Nonetheless, multi-vectors attacks present higher levels of complexity for both the attacker and the target. While attackers must coordinate the different attack vectors among the many different available bots, the defense becomes extraordinarily complex on the target's side due to the difficulty of distinguishing legitimate requests in scale.

2.1.3 Botnets

Botnets are the dominant mechanisms that facilitate the different types/vectors of DDoS attacks on computer networks or applications [265]. Botnets are networks of infected devices under control of a botmaster that can be used to execute a wide range of coordinated activities, which include, but it is not limited to, the different DDoS attack vectors [183]. For example, other fraudulent activities such as mail spamming (phishing), information theft (ransomware), and many extortions can be directly attributed to botnet infrastructures. As previously mentioned (Chapter 1), the critical element that makes these attacks successful is mostly the large number of Internet-of-Things (IoT) devices like dolls, toasters, thermostats, security cameras, and Wi-Fi routers [3]. These devices, which are often not properly secured or not secure at all, are connected to the Internet and often execute arbitrary code, being an easy target to leverage these devices to form a botnet.

As a consequence, massive botnets with thousands of bots have arisen posing a constant threat for businesses and governments. A critical component of a botnet is their Command-and-Control (CnC) infrastructure [183]. Botmasters are able to issue commands to bots via their CnC channel to execute coordinated activities and over time, botnets and their corresponding structure of CnC channels have evolved [11]. The origin of botnets is connected to Internet Relay Chat (IRC) where bots were used to control interactions in chat rooms and they were not intended to deal harm. Next, malicious bots targeting IRC users and servers appeared. In addition, more sophisticated botnets using different propagation mechanisms, different architectures, and communication protocols were developed.

A regular approach to classify a botnet is according its network architecture. Most existing botnets employ a centralized architecture, although decentralized botnets have been frequently detected [265]. In summary, the differences between those architectures are described as follows [2, 265]:

- **Centralized**: is comparable to the classic client-server model. Every bot establishes its communication channel to one static CnC server. The CnC server provides malware updates and is used by botmasters to send messages to the bots (clients) in order to initiate any kind of action.

- **Decentralized**: bots are typically connected to other bots in the P2P (Peer-to-Peer) network in order to exchange command and control traffic and every node acts as bot (client) and CnC server. There are variations in which a given bot can act as a server for other bots for certain rounds, similarly to DHT's for distributing CnC commands.

Advantages of botnets with a centralized architecture include the low latency when sending commands to bots, as well as the most efficient coordination [11]. Furthermore, this topology allows status monitoring of the botnet by botmasters such as number of active bots. However, this structure also yields two major weaknesses: firstly, a centralized botnet is easier detectable because many bots connect to the same central CnC server. Secondly, the central server acts as a single point of failure and can compromise the whole botnet [183].

Centralized botnets were primarily based on IRC (Internet Relay Chat) protocols in its origins [11] or more advanced bots predominantly use the HTTP protocol (HyperText Transport Protocol)

[265]. The communication between bots and CnC can be categorized into "push" and "pull." When using the push strategy, bots stay connected to the CnC host and wait for commands. The CnC host, for instance, an IRC server, then distributes commands to all hosts in a broadcast-like manner [11] or to specific bots through private IRC messages [225]. This enables botmasters to have real-time control over bots and the botnet. Examples of centralized CnC botnets using push strategy are Phatbot, Spybot, Sdbot, Rbot/Rxbot, and GTBot [114, 225]. In contrast to that, the pull strategy is the polling-based bots. Bots poll the CnC Server (*e.g.*, an HTTP webserver) in order to obtain new commands. Through updating the central resource like a web page or a file, the botmaster can issue new commands. When using a pull strategy, the botmaster typically does not have real-time control over the bots because of the delay between submitting new commands to the CnC server and the bots fetching it by polling the server in a predefined time interval. A known bot using the pull strategy is "Bobax," which is mainly responsible for sending spam [265].

Due to the significant weaknesses of centralized botnets, decentralized Peer-to-Peer (P2P) based architectures of botnets have emerged. P2P botnets have no central CnC servers and are therefore more resilient to defenses and countermeasures [106, 114]. In this topology, bots connect to other bots in the P2P network in order to exchange command and control traffic, and every node acts as a bot (client) and CnC server, which makes the coordination of the network decentralized. Because of the absence of a centralized infrastructure, it gets much harder to mitigate P2P botnets. If one or more peers of a P2P botnet is lost, the communication will not be interrupted because there are still other nodes in the network which can provide the command and control information [225]. This characteristic is the main advantage of P2P botnets. Disadvantages of decentralized botnets are the higher complexity and the propagation delay when issuing commands as well as a lack of guaranteed message delivery [183].

Decentralized botnets work as an overlay network on top of the Internet and use different P2P protocols. When forming a P2P botnet, generally, three different types of constructions exist: 1) a "parasite" P2P botnet infects vulnerable hosts in an existing P2P network. Therefore, the botnet scale is limited by the size of the existing P2P network, and in terms of flexibility. 2) in a "leeching" P2P botnet, new bots join an existing P2P network and use it for command and control communication. The field of potential bots or vulnerable hosts is as a result of this more prominent than for "parasite" P2P botnets because bot-candidates could be inside or outside the existing P2P network. An early version of the Storm botnet used the "leeching" technique [225]. 3) a "bot-only" P2P botnet builds its dedicated network in which all members are bots. Botnets like Stormnet and Nugache belong to this type [114, 265].

Furthermore, Cooke *et al.* [48] proposes a third, unstructured model for botnets that could represent a new architecture in the future. In their proposed approach, a bot does not know more than another bot, and bots do not actively fetch new commands from the CnC. Instead, they wait until they get contacted by the botmaster. This could be done through randomly scanning the Internet for bots and, if new bots are detected, the botmaster send them commands. The advantage of this architecture would be that the detection of one bot could never compromise the whole botnet, once there are less messages being exchanged between bots and CnC. Nonetheless, the downside is the increased message latency by design in contrast to a centralized CnC architecture [48].

2.1.4 Booters: DDoS as a Service

The growth of Botnets by taking advantage IoT devices and their insecurities has created a new economic model offering DDoS attacks as a service. In addition to preventing a target from providing a service (and creating an economic impact as a consequence), Botnets can profit from each attack once they are made of a network of connected Bots ready to perform attacks on demand. Such DDoS-as-a-Service model is also referred to as *Booters* or *IP Stressers* [213], in which the access to the Botnet is rented to payers through a web interface enabling to specify the targets of the attack for a certain time. Booters are typically sold as type of SaaS (Software-as-a-Service) model, often including email support and YouTube tutorials [115, 212].

The difference between an *IP Stresser* and a *Booter* is that while the first is a tool designed to test a network or server for robustness, the second is mainly used by criminals to render websites and networks unavailable. However, both share the same platform being a network of malware-infected devices, which are "subleased" to subscribers. In both cases what is being "subleased" is the access to the Botnet in order to launch attack "stressing" your own infrastructure or another target [212]. Based on minimal or no regulation across countries, these tools, which require minimal steps to verify identity and ownership of a target, allows for "stress testing" any target, enabling cybercrime, cyber-vandalism and many other types of DDoS-related activities. In addition, it is possible to conclude that the use of *Booters* (*i.e.*, hiring their services), as well as their availability is far from being restricted due to the following main points [92, 115, 213]:

- **Number of devices**: Increasing number of unsafe connected IoT devices.
- **Lack of strict legislation**: Absence of strict legislation's across different countries imposing severe punishments on *Stressers* or *Booters*.
- **Lack of cooperation**: To detect and mitigate attacks once most *Booters* are behind of cloud protection services such as Akamai or CloudFlare [3, 43].

Characteristics of *Booters* were studied in [212], in which their differences in terms of offered types of attack, payment models, Botnet size in terms of controlled bots, among other parameters were specified and compared. The comparison of these parameters helped to establish a first notion of how Botnets evolved and created opportunities for a new business model, that in addition to harming targets financially, they are also profiting from the attacks. Further, [115, 213] analyzed effects of *Booter* attacks by hiring different *Booters*' services to attack their own infrastructure. These works allows to verify in depth the effects at the target's infrastructure, on ISPs (Internet Service Providers) and IXPs (Internet Exchange Points), the effects of using reflectors/amplifiers, as well as mitigation strategies.

Despite developments and evolving business models the botnet economy as part of the cybercrime economy still has many limitations and inefficiencies. [89] found that it is hard to monetize goods and services in underground black markets (*cf.*, Figure 2.2). They argue that the market for stolen goods like credit card information or cybercrime-as-a-service is a market of lemons, where the buyer does not know the quality of the product before buying it. Several factors contribute to this: First and foremost, there is a large asymmetry of information. The seller has a lot more information about the goods or services than the buyer. Secondly, there is no credible disclosure of the goods or services, as they were illegally obtained and the seller has to stay hidden. Thirdly, there is a continuum of seller from high to low reliability, and it being an illegal market without higher authority that is available to punish fraud, the buyer has to reckon to get tricked himself and will tend to assume a low quality of the goods and services, which results in a low price that he is willing to pay. Lastly, there is no quality assurance or regulation [89]. In order to create a working marketplace and avoid a significant undervaluation of the goods reputation, relationships and networks still play a crucial role in the cybercrime market.

Figure 2.2 shows in specific the monthly revenue distribution by payment channel based on [27] whereas the red solid vertical line marks when PayPal was no longer accepted. In addition to Figure 2.2 confirming that Booters handle large amounts of money, it is possible to observe a change in the types of payments transitioning to from PayPal to cryptocurrencies (mainly Bitcoin). By August 2015, payments from PayPal channel decreased by \$12,458 (44%) from an average of \$28,523 over the previous five months. The Bitcoin payment channel increased by \$6,360 (71%), but did not fully compensate for lost revenue from PayPal [27].

Although many attempts have been made to build real post-sales and customer support platforms to market goods and services, affiliates or trust-based middlemen still play a important role [22]. Therefore, [89] classify the underground economy into two tiers. One tier with established profes-

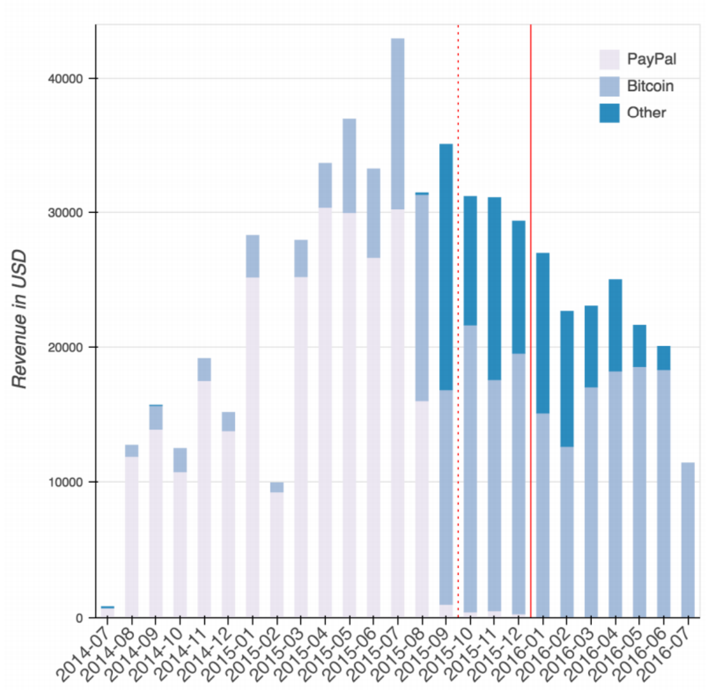


Figure 2.2: Monthly Revenue for Booters by Payment Channel [27]

sional gangs of criminals that have built up a reputation and relationships with other groups, which helps to overcome most of the above mentioned limitations and one tier for the less organized criminals, who will have large difficulties to find buyers for their goods and services and must sell them in ripper infested markets. This leads to low margins for the newcomers in the lower tier and only minor gains, while the members of the upper tiers are engaged in profitable activities. The member of the lower tier accept low pay and high risks because of the believe that the underground economy is a path to get rich quick. These expectations are often promoted by unreliable and exaggerated ballpark estimates of revenue streams. Thus, accurate revenue estimates are important also concerning their influence on perceived attractiveness of participating in the botnet economy [89].

2.1.5 Real-World DDoS Attack Cases

To exemplify the previously exposed concepts and motivate the need for a defense that has a defense comparable to the attack, three selected cases of DDoS attacks are exposed. These are the GitHub DDoS attack exploited by a general vulnerability on Memcached servers [117], and two primarily distributed attacks based on the Mirai botnet (Deutsche Telekom and DynDNS).

GitHub - February 2018

The attack on Github's infrastructure was the biggest to date, reaching a record volume of 1.3 Tbps inbound traffic [3]. The attack was based on a flaw in a cache data replication service (Memcached) in which spoofing and reflection techniques were combined [117]. The goal was to use a reflective amplification in which the attacker makes a spoofed request (with the source IP address of the intended victim) to vulnerable Memcached servers across the world, which replies to a victim IP address with a more massive response. As a result, GitHub was a target being able to respond to after around 10 minutes of unavailability, *i.e.*, Github services were not available world-wide for that length of time [3].

Deutsche Telekom - November 2016

It was an attack caused by a modified version of the Mirai botnet, which uses an included set of 62 default username and password combinations [9], targeted 900 thousand routers of the German Internet Service Provider (ISP) Deutsche Telekom. The modified version aimed at routers configured with standard user/password combinations (reportedly, router devices such as Speedport W 921V, W 723V Type B, and W 921 Fiber [4]). This Mirai-variant included a replication module targeting an HTTP-based protocol used by many ISPs to auto-configure and remotely manage home routers, modems, and other customer-on-premises (CPE) equipment [3]. This incident of unprecedented stopovers for Deutsche Telekom highlighted the concern with the various often insecure devices connected to the internet, demonstrating how hackers' weaponization of more complex IoT vulnerabilities can lead to powerful botnets, such as Mirai.

DynDNS - October 2016

One of the attacks that marked the era of super attacks happened in 2016, in the attack of a significant name resolution service provider (DNS - Domain Name Server) [192]. The attack targeted servers at DynDNS, a company that provides name resolution services to other major Internet content providers such as NetFlix, Github, PayPal, and Twitter. The attack whose primary cause was attributed to the Mirai botnet [9] had ten million unique IP addresses distributed across the world, and happened in different attack waves [192]. For instance, the first wave (7 AM-9 AM EST) managed to render the service unavailable for two hours, and the second wave (12 PM-1 PM) caused one hour of unavailability.

2.2 Distributed Denial-of-Service Defense

The increased frequency, complexity and strength of DDoS attacks have led to the proposal of numerous defense mechanisms. Still, many proposals for counter DDoS have been proposed in which many are still unable to effectively combat large-scale attacks. The literature presents different ways to categorize DDoS defenses [151, 183, 265]. For instance, these can be structured according to the type of deployment *i.e.*, where the defense takes place in the network), type of activity *i.e.*, whether it is a reactive or proactive approach, the architectural type *i.e.*, centralized, distributed, and decentralized, and others.

2.2.1 Architectural Types

The type of architecture concerns the defense strategy against attacks also being related to the category of defense location in the network. Hence, the type of architecture can be defined based on the layout of the infrastructure and selected defense strategy. For example, a simple e-commerce service can be define an on-premises defense based on the scale of visits/customers and resources available on site, as well as combining such defense with off-premises ones. Therefore, there is no mutual exclusion between types of architecture, which offers greater flexibility in possible strategies to combat DDoS attacks.

- **On-premises**: the protection (*i.e.*, detection and mitigation tasks) is implemented by the target system based on locally available resources. The defense can be centralized or distributed across a network under the target's control.
  - *Centralized*: a single target (*e.g.*, firewall, load balancer, intrusion detection/prevention systems) to protect or network under its control.
  - *Distributed*: a single target can have multiple sites where the load related to an attack can be distributed and mitigated.
- **Off-premises**: enforced by third-parties based on mutual agreements. Third-parties can be distributed, in case of cloud-based protection services such as the ones offered by Akamai [3] or CloudFlare [43], or decentralized based on a cooperative network defense.
  - *Distributed*: a third-party acts as a proxy receiving, analyzing and redirecting traffic to the target, which delegates detection and mitigation tasks to the protection provider.
  - *Decentralized*: a target receive, analyze, and communicate attacks to one or more third-parties through an overlay network, which can mitigate the attack based on the request.

On-premises DDoS defenses are most common due to its simplicity to deploy and operate. In case of on-premises centralized defenses, where a single target is attacked, resources available on site can be used to detect and mitigate the attack depending on its scale. If such target has multiple servers distributed across different locations, attack traffic can be redirected (*i.e.*, balancing the load) to reduce the burden of detection and mitigation in a distributed fashion.

Off-premises defenses take place outside the scope of the target, *i.e.*, involves third parties whose traffic targeted to the target may be influenced by them. While distributed off-premises defenses typically involve cloud-based protection services that have a larger capacity for detection and mitigation than the target, decentralized involve multiple independent third parties that can be distributed or not. Therefore, decentralized defenses are more suitable to counter large-scale DDoS attacks due to the potential involvement of multiple organizations.

2.2.2 Deployment Locations

Furthermore, on-premises and off-premises defenses can also be categorized according to deployment location where this defense occurs. In case of volumetric DDoS attacks (*i.e.*, flooding-based), these can be categorized as [151, 265]:

- **Source-based**: are tools deployed at or near the sources of the attack *i.e.*, edge networks. The idea is to prevent network customers from generating DDoS attacks.
- **Intermediate Network**: the defense takes place at the intermediate network, providing an infrastructural defense service to a large number of hosts.
- **Destination-based**: it is the typical location where defense mechanisms are in place, since it suffers the major impact of the attack and, therefore, it is the most motivated to deploy (and bear the cost of) a DDoS defense.

Although the motivation for the existence of source-based mechanisms is valuable, their deployment is not always effective since, due to the highly distributed nature of the attacks, the detection of attacks at the source is not simple, and mitigation goes through the mentioned defense challenges. collaborative. Defenses deployed at intermediate networks can be seen as defense services of third parties (*e.g.*, the service provider or the cloud protection service) in which a large number of customers can be protected. However, due to the large volume of traffic, mitigation detection mechanisms are often complex to operate. Therefore, the most traditional and effective ways to deal with small and medium-scale attacks are defenses deployed at the destination *i.e.*, target of the attack.

2.2.3 Activity Types

The type of activity refers to the behavior of attack defense tools. Behaviors can be coarsely classified as reactive, proactive, or passive:

- **Proactive**: are preventive mechanisms to eliminate the possibility of DDoS attacks and to enable potential targets to endure the attack.
- **Reactive**: are mechanisms aiming to alleviate the impact of attacks by immediately detecting and responding to it.
- **Passive**: mechanisms that can employ detection tools but do not take mitigation measures, which can occur in infrastructures dealing with a high volume of traffic (*e.g.*, core networks).

While proactive measures may involve the set of tools and actions to prevent an attack to happen, reactive measures take immediate mitigation measures [133]. For example, a proactive measure may involve tools to constrain access to resources (*i.e.*, limiting traffic and number of requests per connection). Conversely, a reactive measure would not limit access to resources but react once an attack is detected enforcing mitigation measures on suspect connections. Alternatively, core networks where employing mitigation measures is expensive due to the large traffic volume, may use coarse detection mechanisms that could indicate whether a given host is a potential target.

2.2.4 Cooperation Degree

The relationship between entities in off-premises defenses can occur in different ways. As detailed in [151], there is not always a balance between requests for cooperative DDoS mitigation, as victims are not attacked equally. The authors define three degrees of cooperation [151]:

1. **Autonomous**: perform independent defense at the point where they are deployed (a host or a network).
2. **Cooperative**: capable of autonomous detection and response, but can cooperate with other entities and frequently have significantly better performance in joint operation.
3. **Interdependent**: cannot operate autonomously at a single deployment point requiring either deployment at multiple networks, or relying on other entities for attack prevention, attack detection or for efficient response.

The scenario of DDoS attacks, with an increasing volume of traffic and distribution of attack sources, suggests that an equally decentralized form of defense is the most efficient way to counter such large-scale attacks. In this sense, cooperation between different organizations becomes necessary to orchestrate countermeasures in a timely manner, as well as capable of comprising technical solutions to the need for incentives and possible abuses within a cooperation. Proposed based on the characteristics of BC, BloSS gathers the necessary characteristics to effect the collaboration between organizations providing a defense of cooperative network.

2.3 The IETF DOTS Standard

DOTS stands for DDoS Open Threat Signaling and is a standard currently developed at IETF (Internet Engineering Task Force). DOTS can be described as a set of methods and principles to coordinate defensive measures applied by mitigating peers. As such, DOTS is specified to strictly exist as a standard focused on signaling, which means that in currently published standard document drafts state actual mitigation-related as well as other responsibilities as being out-of-scope with respect to DOTS. The following basic DOTS components are based on the official IETF DOTS Architecture document [156]:

- **DOTS client:** The DOTS client can be described as the downstream DOTS party, which is attached to a target (*i.e.,*, server hosting a web-service) standing under attack.
- **DOTS server:** DOTS servers are the counterpart of DOTS clients, in such that they are affiliated with a mitigation service and therefore represent the upstream DOTS party.

The DOTS standard foresees a simple client-server communication based on DOTS agents (*i.e.,* DOTS clients & servers). The communication is done via a signal channel and optionally, a data channel, which both are mutually authenticated. DOTS does not dictate any specific authentication method *resp.* technique, hence it is an official requirement for DOTS implementing systems to set up the authentication process beforehand, *i.e.,* prior to a signal channel being operational and DOTS servers receiving mitigation requests from DOTS clients. Signal and data channels are specified as follows [156]:

- **Signal channel**: To enable communication among upstream and downstream DOTS parties, a resilient connection – in DOTS called signal channel – must be put into place. DOTS clients attached to an Attack Target use this signal channel to request mitigation support, while the same channel is used by DOTS servers to transmit information about ongoing mitigations back to DOTS clients. The DOTS signal channel is precisely specified in a separate document ([167]), enabling standardization and thus, compatibility among different solutions implementing the DOTS standard.

- **Data channel**: is an optional connection between DOTS parties to exchange DOTS configuration and policy information, *e.g.*, for DOTS clients to send accept-lists holding addresses of trusted sources to their DOTS server. It is assumed, that data is only transferred over the data channel during "normal" conditions, *i.e.*, , not during ongoing attacks. Thus, there is no requirement for the DOTS data channel to be as resilient as the signal channel.

Once the determined authentication process has been successful, both DOTS server and client then establish a DOTS Session in terms of a client-server relationship. However, simple client-server affiliations are not sophisticated enough to implement on top of them distributed, collaborative DDoS mitigation systems. Thereby, a DOTS session traffic may flow over the DOTS signal channel, data channel, or both. Signal channel sessions are specified to run over a single TCP *resp.*, UDP session, while using TLS or, the UDP-compatible DTLS as security protocol. Data channel DOTS sessions are specified to use a single TLS-encrypted TCP connection.

In order to maintain DOTS sessions, agents periodically send heartbeat signals to each other. In the case of a DOTS agent observing an extended absence of heartbeat signals from their counterparts, DOTS sessions can be considered as terminated. Therefore, the reason is that sending an extensive amount of heartbeat signals could potentially result in unintended denials of service at the receiver, which would obviously defeat the purpose of DDoS mitigation services implementing DOTS. IETF allows the DOTS agent operators to configure and set-up the DOTS session maintaining heartbeat exchange mechanism. Nonetheless, the DOTS architecture document [156] states a clear caveat in advising DOTS agent operators to not configure heartbeat signal intervals too short, which would result in a high heartbeat frequency.

Although a decentralized attack mitigation could be achieved, DOTS is not a peer-to-peer protocol (*i.e.*, not fully decentralized). To achieve such decentralization, a fully meshed setup between all peers would have to be established or the recursive DOTS relationships would have to span the whole network. It is unclear whether this condition will introduce a complexity that makes a decentralized approach infeasible. Lastly, the IETF DOTS standard is applicable to a number of scenarios that have been showcased to function with existing implementation that work in various configurations. Most prominently distributed and decentralized mitigation solutions and services are conceptually compatible. However, the complex nature of the DOTS architecture will only pay off for certain scenarios.

2.4 Blockchains and Consensus Mechanisms

BCs offer a significant change for a variety of industries where the disintermediation of trust makes sense by not only modifying technical operations, but also disrupting their business models, operation of existing processes, and legal compliance's [203]. Through a decentralized and immutable data storage, it also enables the enforcement and verification of exchange assets, while changing existing areas by promoting the disintermediation of processes involving multiple stakeholders. By removing a mediator or a third party controlling the operation, less operational costs and higher business agility are expected. This change makes it necessary to understand not only the diverse requirements imposed by each application area but also the technical differences between Blockchain (BC) and Distributed Ledgers Technologies (DLT) in order to understand their trust models/assumptions.

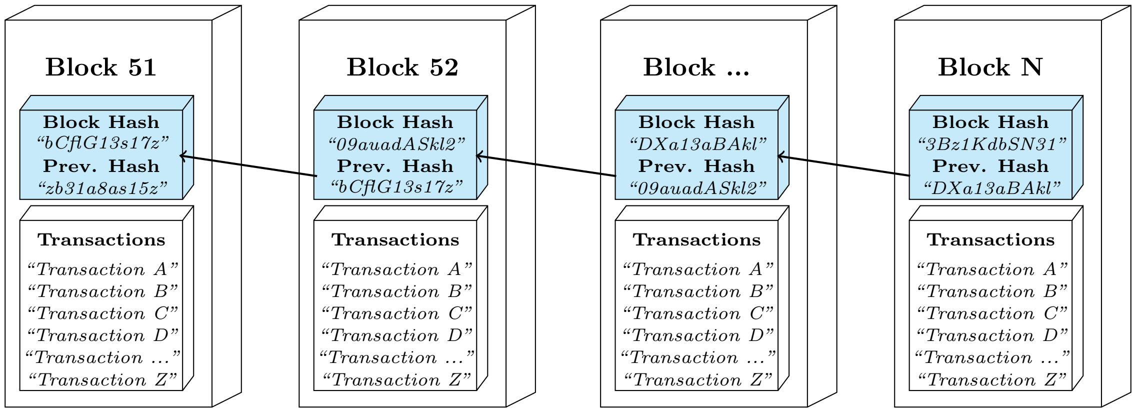


Figure 2.3: Typical Blockchain Data Structure based on Bitcoin [160]

In its purest form, *i.e.*, as proposed in Bitcoin [160], a BC acts like a decentralized and public digital ledger that transparently and permanently record blocks of transactions across computers based on a consensus algorithm without modifying the subsequent blocks (*cf.*, Figure 2.3). However, permissions to write and read, as well as the participation in the block-validation process, can be distributed in different ways in a BC [203]. This allows for different types of BC, commonly referred to as Distributed Ledger Technologies (DLT), which have their characteristics flexible to suit the needs of transparency and confidentiality of each use case. For example, the need to share sensitive data between stakeholders (*e.g.*, patient data in the healthcare industry), could be based on a BC which, however, should not be publicly accessible since sensitive data can be shared.

Although the BC does not provide an utterly trustless solution, the inherent disintermediation contributes through its transparency to an increase of trust among the stakeholders involved [204].

For example, BC technology can be the core element to exchange information about attacks and to distribute the necessary incentives. While it simplifies various technical and economic aspects, social and legal aspects may be impaired by its use. All information in a BC is immutable and transparent to all participants of the cooperative defense. While transparency has a positive impact on trust, information leaks about signaled attacks could result in potential damages to the public image of a domain. Thus, there is a need to build a consensus among participants about the confidentiality of information exchanged in the BC and eventual penalties in case of information leaks.

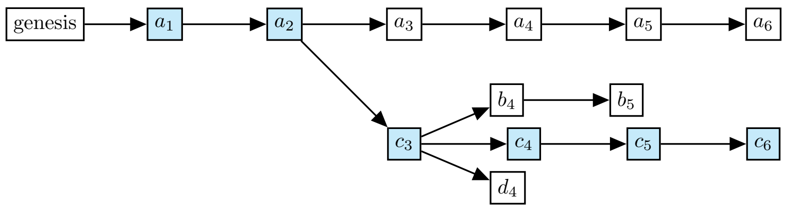


Figure 2.4: A Chain of Blocks with its Canonical Chain Marked in Gray

From a data structure perspective, BC resembles a backward linked-list data structure (*i.e.*, the formal abstract data type name for "chains"), in which access rights to write, read, commit transactions (the consensus result), and to participate in the consensus are required to be all public and open at the same time. Such a chain is depicted in Figure 2.4 whereas the common agreed view of information is built by the changes stored in the blocks marked in gray. As its name suggests, a BC is a sequence of blocks, each holding a collection of transactions. Among other fields, a block always holds a reference to its parent. Thus, a BC acts like a shared, replicated, append-only database (*cf.*, Figure 2.3) where blocks are connected based on a hash value of the previous block, which links current blocks to previous ones composing the chain. Each block represents a number of transactions records, which are confirmed and broadcast to the network so as every node in the BC has their own updated version of the chain.

2.4.1 Blockchain Data Integrity and Immutability

An important aspect of BC concerning other data structures is its ability to guarantee the integrity and authenticity of the data (*cf.*, Figure 2.5). Immutability, refers to the BC ability to prevent changes to transactions that have already been confirmed. Although transactions are generally related to the transfer of assets (*e.g.*, typically cryptocurrencies), they can also refer to the registration of other non-monetary forms of digital data.

Such verification of data integrity is a crucial step in P2P systems because the system is based on a consistent view of the same data stored in different peers. This is possible through cryptography tools such as public-key infrastructure, digital signatures, hashes, among other tools that enable the creation of a unique identifier in time of the information present in a block, which includes the unique identifier of the previous block. Once data (*i.e.*, transactions) are submitted to a BC, they are digitally signed by the issuer and, eventually, stored in a block. This block, containing other transactions, has a hash that is calculated based on the hash of all other information stored in the block. Thus, changing a single piece of information within the block would result in a different block hash, indicating that the block has been modified and, therefore, resulting in a new branch of the BC, *i.e.*, a fork.

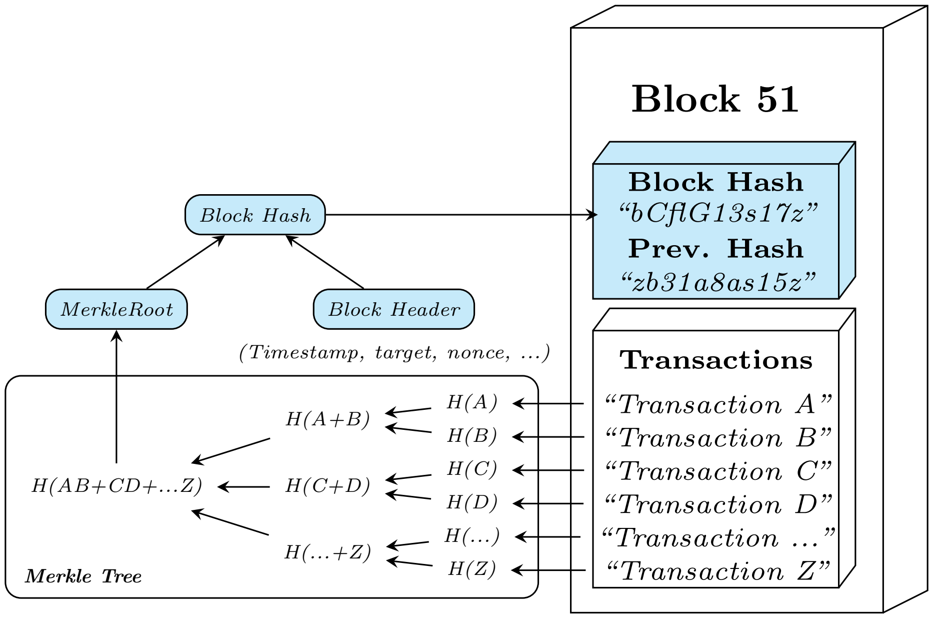


Figure 2.5: Merkle Tree Ensuring Integrity of Blocks [160]

Merkle Trees are a relatively simple and efficient way to guarantee the integrity of the BC, along with the full data replication among all peers in the BC network [160]. Merkle Trees are typically implemented as a binary tree, but they can also be created as an *n*-nary tree, with *n* children per node whereas the leaves (*i.e.*, children) are composed of data (*i.e.*, transactions in a BC). The reason that Merkle trees are useful in P2P systems is that it is inefficient to (a) check the entirety of files and (b) transmit full data over the network. Therefore, the Merkle Tree receives the data on the leaves of the tree as input and calculates the hash of these leaves and us recursively until reaching a single hash, the Merkle Root. A data verification process based on Merkle Trees occurs as follows [237]:

- **Step 1**: System A sends a hash of a file to another system B.
- **Step 2**: System B checks the received hash against the Merkle Tree root *i.e.*, the Merkle Root.
- **Step 3**: If there is no difference between the hashes, then the file has not been modified. Otherwise, continues in Step 4.
- **Step 4**: If there is a difference in a single hash, System B will request the roots of the two subtrees of that hash.
- **Step 5**: System A calculates the required hashes and send to System B.
- **Step 6**: Steps 4 and 5 are repeated until one or more inconsistent data are found.

Decentralized systems can perform data integrity verification in a simple and efficient way. Each time a hash is requested to be verified, it is required n comparisons at the next level where n is the branching factor of a tree *i.e.*, the tree level. Bitcoin implementation [160] relied on Merkle Trees to ensure that transactions within a block are not altered, as well as the ordering of blocks since the hash of previous blocks is included in the hash of the block to generate a unique identifier reflecting the current ordering. Therefore, any participant in this network can verify the integrity of the chain, since the data is replicated to all participants every time a new block is inserted. Thus, Merkle trees are useful to verify information even when sent from an untrusted source, which is a concern in P2P systems.

2.4.2 Permissioned and Permissionless Deployments

BCs can be classified in different types according to their read and write permissions. This differentiation is depicted in Figure 2.6, where each quadrant represents a deployment type, the x-axis represents the two alternatives write permissions (permissioned or permissionless), and the y-axis represents the read permissions (public or private). The light gray square represents the definition of a BC, while the dark gray squares represent DLs. Each deployment type is described as follows [205, 217]:

- **Public Permissioned** denotes the real BC setting, write permissions are restricted to selected entities, but anyone is able to read from the BC. For example, this deployment type can be used for use cases, where multiple trusted authorities want to publish public data, accessible to anyone, (*e.g.*, publishing hashes of academic certificates).

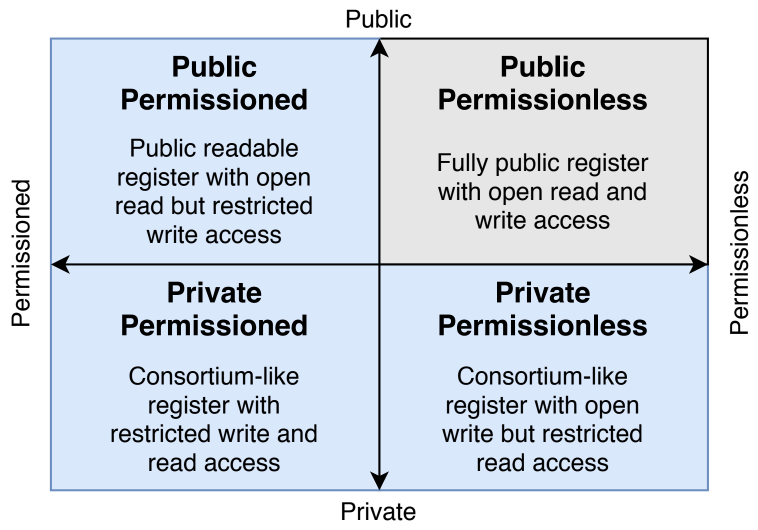

Public
Public Permissioned
Public readable register with open read but restricted write access
Public Permissionless
Fully public register with open read and write access
Permissioned
Permissionless
Private Permissioned
Consortium-like register with restricted write and read access
Private Permissionless
Consortium-like register with open write but restricted read access
Private

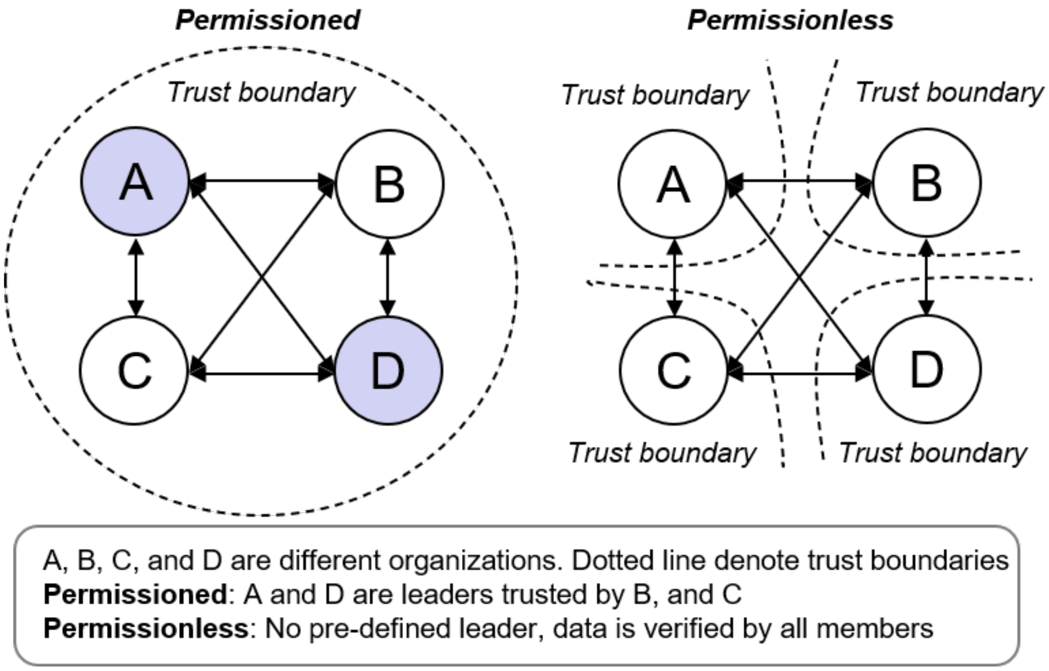
Figure 2.6: BCs and DLs Types of Deployment [217]

- **Public Permissionless** are the most prevalent type of BCs. Bitcoin [160], Ethereum [253], and most of their forks are considered to be public permissionless BCs, due to their read and write permissions, as well as the participation in the consensus, which is open to anyone with Internet access. Thus, public permissionless BCs are the standard type of BC deployment, and most cryptocurrencies are implemented as such.

- **Private Permissioned** trust models resemble traditional databases, where read and write permissions are restricted, and consequently, data can only be read by authorized parties. Restricting these permissions creates a hierarchy between participants (*e.g.*, role-based actions), where the main features of a BC (*e.g.*, transparency, immutability, and decentralization) may not be advantageous for a potential application.

- **Private Permissionless** are comparable to public permissionless BCs, but the notion of the reading access control is restricted to a particular (pre-defined) group or community. Therefore, writing and reading permissions are open to all participating members of such private groups. A dedicated supply chain BC would also act as a possible example since the information exchanged is only readable by its authorized members, but all members can issue transactions without limitations.

While permissioned represents a class of algorithms that restrict participation in the consensus to a single or a set of trusted members, permissionless allows all members of the decentralized network to participate in the consensus process through a selection process. Further, while PBFT [32] and XFT [128] are examples of algorithms supporting BFT including a portion of malicious members in a peer-to-peer network, Proof-of-Work (PoW) [160] and Proof-of-Stake (PoS) [161] are examples of algorithms based on (different) processes of selecting a leader, responsible for replicating information on the network. Therefore, while permissioned algorithms require that there is direct trust in the selected members, permissionless requires trust in the selection process *i.e.*, in the algorithm, being transparent to all members and verifiable.



A, B, C, and D are different organizations. Dotted line denote trust boundaries
Permissioned: A and D are leaders trusted by B, and C
Permissionless: No pre-defined leader, data is verified by all members

Figure 2.7: Trust Boundaries in Permissioned and Permissionless Consensus Algorithms [25]

Permissioned and permissionless consensus with respect to trust boundaries are illustrated in Figure 2.7. Permissioned protocols (*e.g.*, PoA, PBFT), but also traditional decentralized databases (*e.g.*, P2P torrent, DHT - Distributed Hash-Tables) that do not necessarily have an append-only structure. Also, although there is no trust between all members, there is a trust of the members in a portion of the leaders making the right to write in this decentralized database to be shared among these previously selected leaders. However, in a permissionless consensus in which the members do not trust each other, but in the protocol, leaders are selected in each round by a selection process in the protocol (*e.g.*, PoW, PoS) in which the result can be validated by all members.

There are significant trade-offs that should be taken into consideration before choosing a technical option. For example, the trade-off between transparency and confidentiality is a relevant discussion to be considered, which also influences the BC type of deployment. While transparency is necessary to increase the levels of trust between cooperative entities, privacy is also essential to ensure that exchanged data do not reveal sensitive information. For example, if an organization A is cooperating with an organization B, such cooperation should be visible on-chain but sensitive details and data could be exchanged off-chain.

Table 2.2: Mapping Tradeoffs between Application Requirements to Blockchain Types

Types / Req.	**Public Permissionless**	**Public Permissioned**	**Private Permissionless**	**Private Permissioned**
Transparency	World visibility	World visibility	Community visibility	Role-based visibility
Control	Distributed, validators are defined in an election process	Distributed, validators are defined in a selection process	Distributed, validators are defined in a selection process	Centralized based on trusted nodes
Reliability	Full replication	Full or partial replication	Full or partial replication	Full or partial replication
Performance	Slow	Medium	Medium	Fast

A balance between total transparency and full confidentiality is necessary in order to be able to enhance trust in a cooperative defense scheme, and at the same time, preserve confidentiality of the members. Therefore, a possible way to preserve confidentiality while the verification/validation process of exchanged information is transparent to all members, *i.e.*, the consensus within the permissioned setting, while essential data is transparently negotiated on chain but lists of addresses to be black or white-listed are exchanged off-chain.

Another trade-off to be considered is the relation between the reliability and performance aspects (*cf.*, Table 2.2), which is also the result of the discussion between centralization or decentralization. BCs are naturally slower than any centralized database due to their complete replication of the data, and consensus mechanism. Not only there is a latency to synchronize the state in all node, but also the transaction performance is affected in temporal and spatial dimensions to keep a consensus in the network (*i.e.*, ensure that all nodes are in the same state). In this sense, the PoW algorithm has a fundamental importance to guarantee that the BC is completely distributed, that is, the nodes responsible for deciding which transactions should be in the next block are not predefined entities but chosen by a process that demands a computational effort of the elected node.

2.4.3 Consensus Algorithms

The consensus algorithm is the main element of a decentralized system, being typically defined based on the level of trust among its participants [268]. It is also important to note that consensus mechanism emerged before BCs, as a way to handle synchronization failures in distributed or decentralized systems. However, in any scenario, the consensus has as goal applying and verifying a set of rules established by a set of participants acting in an organized manner. Thus, the specification of a consensus scheme involves, besides the organization of the nodes and the definition of its rules of operation, the infrastructure on which the BC operates, (*e.g.*, the way messages are exchanged, the organization of the network and the algorithms employed).

In distributed or decentralized systems, it is not only important to guarantee resistance to failures, but also to ensure that transactions are ordered in a temporal manner according to their dispatch. For example, if several people concurrently try to book the a seat on an airplane, then a consensus algorithm could be used to determine which one of these mutually incompatible operations should be the winner [134]. Therefore, a consensus algorithm must satisfy four properties [145]:

- **Uniform Agreement**: No two nodes decide differently.
- **Integrity**: No node decides twice.
- **Validity**: If a node decides on value v, then v was proposed by some node.
- **Termination**: Every node that does not crash eventually decides on some value.

Thus, given a cluster of N nodes and a set of proposals P_1 to P_m, every non-failing node will eventually decide on a single proposal P_x without the possibility to revoke that decision. All non-failing nodes will decide on the same P_x. In addition, there are different ways to compose the set of nodes that will participate in the process and consensus. These different ways, in which different assumptions of trust are taken as a premise, distinguish the different ages or stages of the evolution of consensus algorithms (*cf.*, Figure 2.8).

2.3.3.1 Classic Models

Safety in distributed systems was widely studied since the 1970s, alongside the rise of distributed databases and transactions [16]. In this first generation, most consensus algorithms were developed considering a replication model aiming to strike a balance between the required levels of performance in contrast to a level of data redundancy towards the assurance that the system will be always available

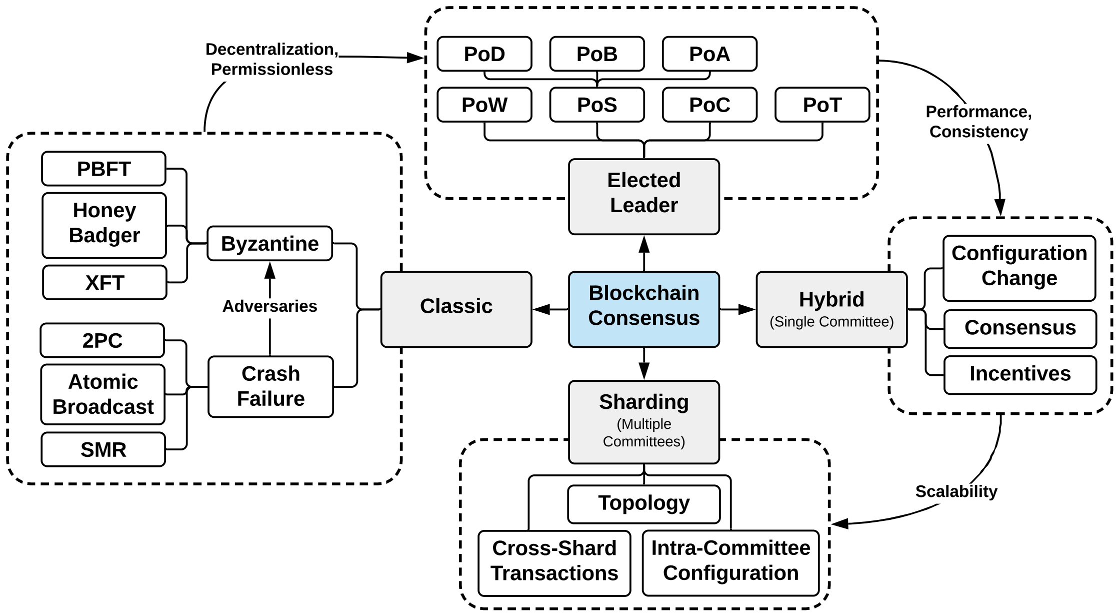


Figure 2.8: Evolution of Blockchain Consensus, from Classic Consensus Models Until Sharding [45]

regardless of different types of failures. For example, the Two-phase Commit (2PC) was proposed in 1978, allowing a transaction manager to atomically commit a transaction, depending on different resources held by a distributed set of resource managers [79].

In the *crash failure* model, nodes may fail at any time by stopping to process, emit or receive messages. Usually these failed nodes remain silent [55], although a number of distributed protocols consider recovery. Other aspects, were observed as distributed systems began their decentralization process, *i.e.*, nodes belonging to third parties could act maliciously. Whereas in distributed systems nodes belong to the same organization (*i.e.*, *N* servers, single organization), in decentralized systems there are several nodes that belong to several organizations (*i.e.*, *N* servers, *N* organizations). Thus, such models may lead to a byzantine failure model, assuming that failed nodes may take arbitrary actions—including sending and receiving sequences of messages that are specially crafted to defeat properties of the consensus protocol. Examples of **Crash Failure-Tolerance (CFT)** approaches are:

- **Two Phase Commit (2PC)**: is an atomic commitment protocol ensuring that a transaction either commits at all the resource managers that it accessed or aborts [79]. Further, it guarantee that transactions cannot fail silently without the transaction manager being aware.

- **Atomic Broadcast**: is one of the major issues in distributed systems. It consists in providing processes with a communication primitive that allows them to broadcast and deliver messages in such a way that processes agree not only on the set of messages they deliver but also on the order of message deliveries [51].

- **State Machine Replication (SMR)**: is a general method to implement fault-tolerant services in distributed systems [220]. A fault-tolerant state machine can be implemented by replicating itself and running a copy on each processor in a distributed system. Thus, each copy run by a non-faulty processor starts in the same initial state and executes the same requests in the same order in order to produce the same output.

The idea of reaching consensus in adverse situations was illustrated as an allegory of generals in the *Byzantine* empire attacking Rome [121]. In this allegory, generals surrounding Rome should make a unified decision to obtain a successful attack or retreat. While some generals may prefer to attack, others would prefer to retreat or could even be bribed by Rome to retreat. Therefore, the allegory evidences the importance of communication in a decentralized scenario and obtaining consensus among most generals concerning the decision to attack or retreat. Formally, a byzantine agreement is defined as [121]:

> *Finding consensus in a system with byzantine nodes is called byzantine agreement. An algorithm is $f-$ resilient if it still execute correctly with f byzantine nodes.*

Therefore, as a natural evolution of crash failure models, consensus algorithms known as Byzantine support that up to a percentage of nodes act maliciously, *i.e.*, node with arbitrary behavior including "anything imaginable" such as not sending messages at all, or sending different and wrong messages being sent to neighbors, or lying about the input value [267]. In the generals' problem described in [121], even when some generals (nodes) did not act in a unified manner either to attack or to retreat, it was possible to obtain a scenario of success. Examples of **Byzantine Fault-Tolerance (BFT)** approaches are:

- **Practical Byzantine Fault Tolerance (PBFT)**: The PBFT organize nodes to operate in rounds, so that in each round a primary node is selected according to pre-determined rules [32]. The primary node (*i.e.*, the leader) is then responsible for distributing information to other nodes. The process is divided into three phases: Pre-Prepared, Prepared and Committed. To move from one phase to the next, a node must receive at least 2/3 of votes from all nodes. Therefore, for this to be possible is required that the total number of nodes be known by the network.

There are no computationally expensive mechanisms in this case, just a simple check between nodes is necessary to reach a consensus.

- **Paxos**: is one of the most important and widely used consensus algorithm since it was the first algorithm proved to be correct [122]. It works by selecting a single value (proposal) from one or more proposed values and distribute the value among the participants. Then, consensus is achieved when a majority of systems running Paxos agrees on one of the proposed values.

- **RAFT**: works by assigning three states: follower, candidate, or leader. Then, a leader is elected after a candidate node receives enough votes, and all changes then have to go through the leader. The leader commits the proposed changes once replication on the majority of the follower nodes is completed [171].

- **Tendermint**: requires three consecutive steps (forming a round) in order to agree on a new block: pre-vote, pre-commit, and commit [28] . In each, a 2/3 majority of nodes has to be found which agree on the problem statement. In each round of following these steps, *validator* nodes start by deciding whether they submit a pre-vote for a proposed block. If a node receives the necessary majority of pre-votes stated above, it broadcasts a pre-commit message. Again, if the node has received a majority of such messages, it validates the block and publishes a commit message. Eventually, if the majority of commit messages is reached, a node accepts the block.

- **Cross-Fault Tolerance (XFT)**: uses a combination of asynchronous and synchronous methods for network communications further combining the speed of Crash-Fault Tolerance (CFT) and the reliability of Byzantine Fault Tolerance (BFT) [128].

- **HoneyBadgerBFT**: first practical asynchronous BFT protocol, which guarantees liveness without making any timing assumptions, *i.e.*, HoneyBadgerBFT tolerates faults in the wild range of wide-area-networks outperforming other BFT consensus algorithms [149].

2.3.3.2 Elected Leader

Classical consensus algorithms are marked by the presence of one or a set of nodes determined as leaders, *i.e.*, they require a certain level of trust pre-established in the network. With the introduction of Bitcoin and PoW a new type of consensus was introduced, the lottery-based. In these algorithms, a game is proposed by means of a protocol that typically requires the solution of a computational problem (*e.g.*, partial-hash collision), in which the winning player becomes the leader. Therefore, the leader is elected based on a protocol known and verified by all participants in the network. A list of selected algorithms is described below [148, 217]:

- **Proof-of-Work (PoW)**: was introduced in Bitcoin [160], being the first consensus algorithm completely decentralized. In PoW, nodes known as miners participate in a hash competition in which a miner needs to solve a partial hash collision competition, *i.e.*, miners are required to find a hash with a minimal number of zeros in the input, satisfying a certain target. This mechanism introduces a computational overhead between miners to find the target hash in minimal time.

- **Proof-of-Stake (PoS)**: is an alternative to PoW in which, instead of requiring a computational effort, miners are able to create blocks based on the amount of resources they have at stake [267]. Based on such approach, PoS is able to reduce the energy costs of the expensive PoW mining processes as well as the dependence on specialized hardware. However, in a PoS-based BC, where no resource expenditure is required, the network is more susceptible to attacks.

- **Delegated Proof-of-Stake (DPoS)**: projects a representative democracy where stakeholders elect nodes to generate and validate blocks [267]. Malicious nodes will not pose a problem as they can be easily removed from the delegation.

- **Proof-of-Stake-Time (PoST)**: proposed in the cryptocurrency VeriCoin [187] as a time accepted nonlinear consensus that maintains the efficiencies of PoS, while attempting to increase the distribution and security of the consensus. Time-acceptance is defined by a proof function that defines a fraction of time active and idle at a given block. It is similar to the Proof-of-Importance (PoI), but taking into consideration the age of coins to leverage older nodes in the network.

- **Proof-of-Authority (PoA)**: belongs to the same family as the PBFT [55]. A group of nodes are recognised as authorities and identified by an unique id. It is assumed that the majority can be trusted. The mining happens in rotation, where the creation of blocks is evenly distributed among the authorized nodes. PoA is most often present in permissioned BCs.

- **Proof-of-Capacity (PoC)**: defines hard disk space as a resource to mine the blocks [255]. Differently from PoW where CPU resources are required to participate in the partial-hash collision game, in in PoC, the miners need to rapidly change a number (*i.e.*, nonce) in the block header aiming to find a correct hash value (above the target). The first miner to identify the correct hash value broadcasts that information to the network. PoC is also called Proof-of-capacity (PoC) or Proof-of-storage (PoS).

- **Proof-of-Contribution (PoCo)**: is based on users' contribution to the network, describing their interactions and leveraging features such as staking (*e.g.*, resources as computational power) to build the required incentives [257]. It is also similar to protocols that rewards volunteers for donating their computer time to scientific computation, such as astronomical observation data research.

- **Proof-of-Elapsed Time (PoET) or Proof-of-Time (PoT)**: designed to be a production-grade protocol capable of supporting large network populations that include byzantine actors [170]. PoET leverages a Trusted Execution Environment (TEE) to provide randomness and safety in the leader election process via a guaranteed wait time. The protocol stochastically elects individual peers to execute requests at a given target rate. These individual peers would then sample an exponentially distributed random variable and wait for an amount of time dictated by the sample [170]. The peer with the smallest sample wins the election.

- **Proof-of-Burn (PoB)**: encourages users to burn – or make permanently unavailable – mined coins in a verifiable manner [93]. To burn the coins, miners send them to a verifiably unspendable address *i.e.*, not possible to recover the coins. This process does not consume resources and ensures the is active. Depending on the implementation, miners are allowed to burn the native currency or the currency of an alternate chain. In turn, miners receive a reward in the native currency token of the PoB chain.

- **Proof-of-Brain (PoBR)**: stands for user activity and encourages engagement and quality content. The mining process occurs by creating or interacting with content through voting (*i.e.*, likes and comments). The more likes, comments or proved views a page gets, the more coins can be minted [231].

- **Proof-of-Deposit (PoD)**: nodes in this scheme have to make a security deposit before they can mine and propose blocks [28]. Hence, each validator node has to pay a security deposit in order to be able to create new blocks. The logic behind the deposit is that a provably faulty node would lose their security deposit on which a block reward has to be paid as interest. Therefore, if a validator node produces an invalid block, it is penalised by losing its deposit. Conversely, a honest validator can make a small profit on its deposit.

- **Proof-of-Importance (PoI)**: is a modified version of PoS (Proof-of-Stake) where it takes into account more factors than only the nodes' stake [210]. PoI can consider for example nodes' reputation, number of transactions, usage and movement of tokens to determine a level of trust and importance, and others. Further, PoI is suitable for IoT networks as it requires less computational resources and low latency.

- **Proof-of-Location (PoL)**: mostly used for IoT networks in which authenticated devices will confidentially store location data [7]. It can be seen as a digital certificate that attests someone's presence at a certain geographic location, at a certain time. Then, users can reveal this personal information at will.

- **Proof-of-Activity (PoAC)**: combines PoS and PoW towards an hybrid approach ensuring that a stakeholder is selected in a pseudorandom but uniform fashion [18]. PoW and PoS are combined together to achieve consensus and good level of security. In PoA, the mining process starts as a standard PoW process with various miners trying to outpace each other with higher computing power to find a new block. When a new block is mined, the PoA switches to a PoS-mode, with the newly found block containing only a header and the miner's reward address.

Bitcoin's PoW introduced a significant change for decentralized systems by also decentralizing the consensus algorithm. Thus, by electing a leader through a lottery-based game in which participants (*i.e.*, miners) are challenged to solve an often difficult (computationally intensive) problem, the protocol was able to democratize the process of participation in the consensus in line with an incentive scheme through transaction fees collected by the miners. Nevertheless, the popularization of BCs and BC-based applications, imposes a series of performance requirements, which PoW is not able to satisfy. Thus, many Proof-of-X algorithms emerged to balance the performance and security requirements of the applications. For example, for applications where maintaining confidentiality to a selected group of members is required, it is possible to use permissioned algorithms in which it is possible to obtain a performance increase and the guarantee of restricted visibility to members.

An overview of these consensus mechanisms based on elected leader is presented in Table 2.3. In addition to the categorization by performance and type, the finality of these algorithms is defined as probabilistic or deterministic. While probabilistic models includes elements of randomness, deterministic models removes any possible uncertainty and the same outputs are obtained given a certain input. In addition, it is possible to distinguish between its algorithms according to the way (*i.e.*, Type as determined in Table 2.3) in which the algorithm reaches consensus. For example, in PoW-based algorithms it is required that a non-trivial and self-adjusting computational problem be solved within a certain period of time. Also, there are algorithms that require some kind of resource at stake, *e.g.*, coins, computational power, storage capacity, among others.

Table 2.3: Comparison of Elected Leader Consensus Algorithms

Consensus	**Type**	**Openness**	**Performance**	**Finality**	**Short Description**
PoW	Work	Permissionless	Low	Probabilistic	Complex solution to find but easy to verify
PoS	Stake	Permissionless	Medium	Probabilistic	Validator stake its coins, bigger stake has higher chances to validate transactions
DPoS	Stake	Permissionless	Medium	Probabilistic	Network delegates permissions to validate transactions to a small and selected number of participants
PoST	Stake	Permissionless	Medium	Probabilistic	Improvement of PoS by giving a mining preference to older nodes (coins)
PoA	Identity	Permissioned	High	Deterministic	Known participants are determined as authorities in order to validate transactions
PoC	Stake	Permissionless	Medium	Probabilistic	The more hard-drive disk space dedicated the higher chances to participate in mining
PoCo	Stake	Permissionless	Medium	Deterministic	Leverages the idea of desktop grid, being based on the contributed computational power
PoL	Identity	Permissioned	Medium	Deterministic	Beacons are used to detect nodes and timestamp its presence in a synchronized manner
PoET	Identity	Permissioned	Medium	Probabilistic	Transactions are validated in a trusted execution environment with equally distributed time periods
PoB	Stake	Permissionless	Medium	Probabilistic	Participants need to burn coins to be entitled to participate in the mining process
PoBR	Stake	Permissionless	Medium	Probabilistic	Incentivize participants to create and curate content stored then in a blockchain
PoI	Hybrid	Permissionless	Medium	Probabilistic	Similar to PoS including additional features that influence nodes' ranking
PoD	Stake	Permissionless	Medium	Probabilistic	Similar to PoS where participants need to make a deposit to be entitled to validate transactions
PoAC	Hybrid	Permissioned	Medium	Probabilistic	Hybrid between PoW and PoS

In addition, there are algorithms in which leaders are pre-determined based on their identity (or uniquely identifiable set of characteristics), which resemble algorithms of the first generation of consensus but which, however, still allow the selection of leaders. For example, this is the case with Proof-of-Authority (PoA) in which the identity of the participants or the confidence of the other participants in this identity, determines their permission to validate blocks. Similarly, Proof-of-Importance (PoI) aims to determine the relevance of the participants through their identity and other characteristics that allow to trace a reputation history of this node *i.e.*, an hybrid between stake and identity. Finally, algorithms can also be based on hybrid mechanisms to balance aspects of security and performance. This approach is observed in Proof-of-Activity (PoAC) and, as described, in Proof-of-Importance (PoI).

2.3.3.3 Hybrid Consensus in a Single Committee

Hybrid consensus models emerged as a way to overcome the limitations of just one consensus algorithm (*cf.*, Figure 2.9). It is possible to think, for example, of an application/use case that has a public and a private part in which it is possible to use a specific consensus mechanism for each. For instance, in the scenario of issuing educational certificates [205], educational institutions can use BC as a platform to increase transparency between departments or different campuses of the same institution, and use another public part using, for example, based on Bitcoin, to create a public record of issued certificates.

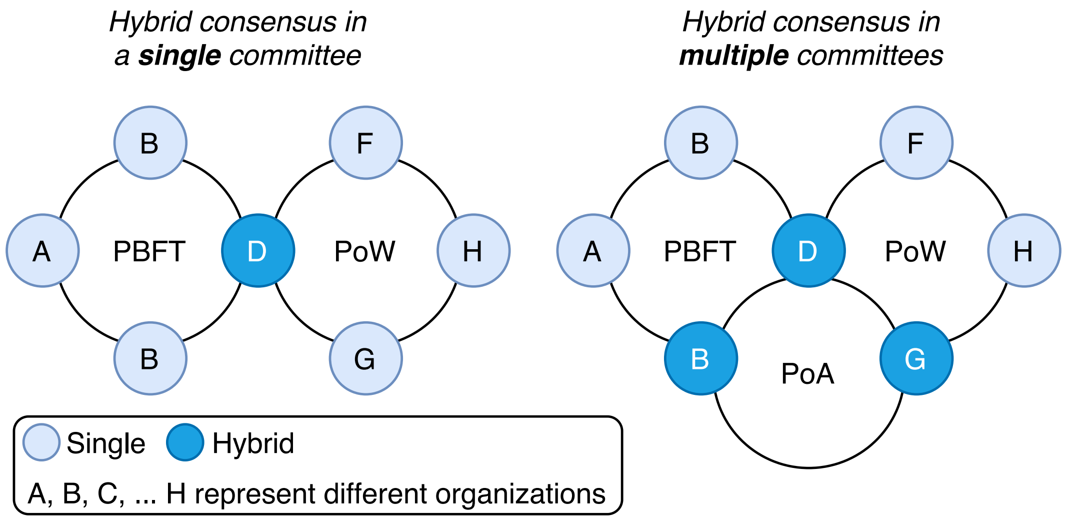

Hybrid consensus in a **single** committee
Hybrid consensus in **multiple** committees
B
F
A
PBFT
D
PoW
H
B
G
B
F
A
PBFT
D
PoW
H
B
PoA
G
Single
Hybrid
A, B, C, ... H represent different organizations

Figure 2.9: Hybrid Consensus in Single and Multiple Committees

The expanded idea would be to form a committee (*e.g.*, based on PoW), and then have the committee use another consensus protocol (*e.g.*, PBFT and its derivatives) within the committee to agree on blocks [16]. Thus, by combining the use of different consensus into a single application would be an approach to counter performance as well as safety limitations, such as weak consistency and low fault-tolerance.

2.3.3.4 Multiple Committees, Sharding

While a single-committee consensus can improve significantly performance over a single consensus, its major limitation is that it scalability (*cf.*, Figure 2.9). Thus, adding more members to the committee would decrease the throughput in terms of transactions per second. This motivated the design of a consensus that is achieved through multiple committees to transactions scalable by splitting across multiple committees (shards), which then would process these transactions in parallel [45].

For example, in an application (*e.g.*, recording of educational certificates [205]) that only needs to create a proof of existence of a document (*i.e.*, time-stamping), it is possible to make *N* records in the form of transactions to be distributed simultaneously between *M* BCs. Thus, the application would support hybrid sharding based on multiple communities. However, with multiple committees there are questions concerning the arrangement of the topology of committees, whether those committees can be trusted, and other concerns inherent to the traditional use of an application based on a single consensus algorithm (*e.g.*, performance of each committee, block size, block time) [16].

2.4.4 Smart Contracts

The notion of a Smart Contract (SC), as a computerized transaction protocol that executes the terms of a contract agreed between the parties involved, was proposed long before the arrival of BCs [236]. The lack of a decentralized infrastructure providing a suitable disintermediation was resolved by BCs, and SCs now became implementable, since BCs outline the perfect distributed environment for their deployment and operation. The goal of an SC is to (a) satisfy common contractual conditions as with any regular paper-based contracts, *e.g.*, in terms of payments, liens, confidentiality, or even enforcement, (b) reduce malicious and accidental handling, and (c) avoid any trusted intermediaries. Thus, the SCs concept is a viable path to automate and ensure agreements reliably and more efficiently than paper-based contracts as of today. Henceforth, as specifically designed from scratch in Ethereum [253], the BC has proven to be a highly appropriate infrastructure for the fully decentralized and transparent execution of a mutual agreement between parties.

Despite Ethereum becoming the first and widely accessible platform that unveiled the entire potential of SCs, Bitcoin also features contracts, however, only in the form of simple scripts for performing transactions, which suite the finality of the application, *i.e.*, a cryptocurrency, in a simple and efficient manner [160]. In contrast, Ethereum proposed a sandbox environment through the Ethereum Virtual Machine (EVM) [253], in which it is possible to execute arbitrary and Turing-complete code directly on-chain, (*i.e.*, allowing for the execution of loops). An EVM defines an environment isolated from the host itself, being precisely the same for all Ethereum nodes (called "Ethereum Clients") in the BC network. A client software, *e.g.*, Geth and Parity, is used for external communications and interactions with the operating system of the host node.

Although there are currently many different BCs providing support for the execution of SCs, the majority follows the model determined by Ethereum, in which a sandboxed environment ensures

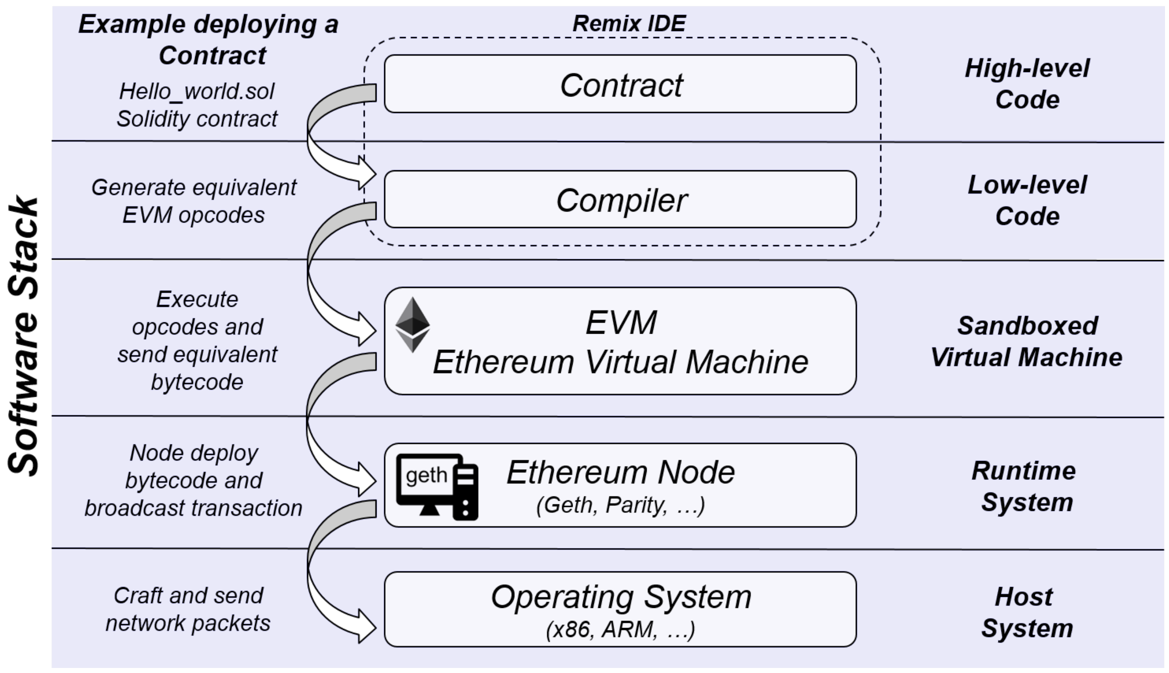


Figure 2.10: SC Deployment on the Ethereum Virtual Machine (EVM) [253]

that the execution of the SCs is precisely the same across all nodes of the network. Thus, Figure 2.10 illustrates the respective components involved in the creation of an SC in an EVM as well as the deployment steps needed. While the SC is defined in a high-level language, *e.g.*, by applying the Integrated Development Environment (IDE) Remix, it is transformed and interpreted in bytecode, until it is propagated on to the BC network according to the consensus algorithm configured in the EVM. The EVM itself operates on the respective operating system and runs the Ethereum protocol. Thus, the client communicates with the host's operating system to broadcast the transaction containing the bytecode corresponding to the SC, which is crafted into different IP packets and sent to the BC network. The role of the EVM (*i.e.*, the BC's "virtual machine") is crucial, since code must be identical across all Ethereum nodes in the BC network and must comply with well-defined interfaces. Therefore, it is possible to enable flexible support for different clients, which, in turn, can provide different abstraction levels for the development of applications.

A relevant factor for the popularity of general-purpose, decentralized applications (dApps) based on SCs, is the familiarization of developers with high-level SC programming languages and the *modus-operandi* of BC. Following the SC example as shown in Figure 2.10, it is possible to create a simple application returning a string the text *hello world* whenever it is called (*cf.*, Listing 2.1). Ethereum provides direct support for its high-level language "Solidity", with a syntax based on JavaScript; however, support for different languages that developers are familiar with exist if that language is compiled into EVM op-codes, which are interpreted and executed by the EVM. Thus, such flexibilization is made possible through such abstraction layer allowing the use of different languages as long as your compiler generates EVM opcodes.

```
pragma solidity ^0.4.10;
contract HelloContract {
  string helloWorld;
  function getHello() public {
       helloWorld = "Hello World";
  }
}
```

Listing 2.1: A Simple Hello World SC Example in Solidity

Regardless of the high-level language of the SC, the EVM interprets and executes EVM opcodes based on an incentive scheme to execute these contracts, the *gas*. Thus, the higher is the complexity of an SC, the higher will be the cost for its deployment and operation, demanding a higher amount of *gas* for its methods to be executed. It is important to note that such an incentive scheme is required for BCs, *i.e.*, their permissionless deployments, since anyone with Internet access can participate and, thus, a mechanism is needed to prevent the BC from DoS attacks, either maliciously or just by accident, *i.e.*, an endless loop within an SC. In contrast, for DLs, especially permissioned deployments, such a necessity of incentives may not be needed, since the BC network consists out of permissioned, *i.e.*, pre-selected stakeholders. Once a sufficient amount of *Gas* is provided for the SC's deployment, the EVM generates the bytecode, which is sent to the client.

2.5 Blockchain as an Enabler of Trust in a Cooperative Defense

Trust is a concept involving human relationships and builds the foundation for decision making in different contexts or communities [41]. Trust factors exert a fundamental relevance in the influence of decision-making processes, forming the understanding, establishment and management of trust in these contexts. However, trust is a highly subjective concept, often relying on individual's behaviours within particular contexts, which are governed by a set of rules [74]. For individuals, the consistent ethical behaviour among peers enables the creation and establishment of trust, whereas the subset of ethics may break out of the given set of rules, but still remain within the context itself.

With the creation of Bitcoin in 2008 [160], Blockchains (BC) played a key role in the perception and establishment of trust. Instead of relying on a direct trust relationship in which there is a need to know the identity and reputation of entities, BC follows a permissionless, or trustless, and fully decentralized trust model where it is not necessary to have knowledge of entities but only their actions, which are immutably and transparently available on the platform [21]. Therefore, there is a paradigm shift from placing trust directly in an entity, now placing trust in the platform, whose capabilities enable transparency and public verifiability. Therefore, individuals can verify whether the actions of another individual are in accordance with their morals even without revealing their identity. However, it is observed that there are different types of BC [203] (*e.g.*, permissioned, permissonless), in which each type follows a specific trust model, making it necessary to evaluate these different flavours with respect to trust modeling.

Trust is a fundamental aspect of any cooperative environment and difficult to obtain since it may rely on many non-technical aspects [88]. Also, 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. *BC has a role of a "trust-enabler" in this context, providing transparency between cooperative organizations and thus possibly increasing trust-levels based in their interactions.* However, it is not possible to quantify the role of BCs as a trust enabler, being not possible to determine a "probability" in which the use of BCs is a determining factor in ensuring trust between organizations. The role of BCs in building trust has been studied by [87], in which the authors present solutions on how these conflicting notions may be solved and explore the potential of BCs technology for dissolving the trust issue. According to [191, 246], 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 timeframe.

- **Context-dependent**: the degree of trust on different contexts is significantly different. For example, organization A may share threat indicators but may not disclose actual malware intelligence due to several factors (*e.g.*, legal issues). Thus, trust may exist between organizations A and B only for sharing "threat indicators" context.

- **Non-transitive**: if organization A trusts organization B, and organization B trusts organization C then organization A may not trust organization C. However, A may trust any organization that organization B trusts in a given context.

- **Asymmetric**: trust is a non-mutual reciprocal in nature. That means if entity A trust organization B, then 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 [110, 135], 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, consensually agreed and publicly (within the context) available record of transactions is seen as an enabler of trust in the platform [87]. In addition, the definition of rules between participants through SCs would allow for all parties involved to verify the execution of the code that defines the cooperation. It is important to note, however, that the 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 that goes beyond any technical solution.

Information disclosed in this cooperation network (*i.e.*, the cooperative defense) should not be disclosed to the general public, but kept within the alliance. It is important that all members participate in the BC consensus on an equal basis, preventing members from *e.g.*, having the ability to censor certain transactions. Finally, BC allows to build a reliable and robust platform for signaling DDoS threats in a transparent and verifiable manner, but it does not cover all the security needs of such platform. For example, while transparency favors a trust-free platform, it is needed to strike a balance with confidentiality requirements of each member in order to securely exchange information. Thus, data exchange should be done off-chain through an encrypted data channel (*e.g.*, blacklisted addresses signed by the sender) via, for example, the Inter Planetary File System (IPFS) [17], ensuring the confidentiality as well as the integrity of the attack information based on a per-message signature bundled with the attack information.

2.6 Reputation Tracking and Management

Reputation and incentive-based mechanisms have been widely explored for decentralized systems. However, the BC introduces a novel way to decentralize trust creating a certification infrastructure that records all traces of transactions, allowing users to look up and record histories of transaction outcomes. By relying on the public ledger to form reputation scores and reward contributors, reputation and rewards of users are tightly coupled to their actual network activity. Furthermore, BC provides the underlying structure towards a financial reward scheme through cryptocurrency. This currency represents an incentive to behave reputable, if actions are rewarded accordingly. Financial rewards and reputation can be treated independently or combined.

While providing many benefits, a cooperative defense also poses many challenges. For example, why such organizations should help each other. In a competitive environment, trust needs to be established. Solely relying on voluntary contribution (*i.e.*, , 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 Figure 2.11. First, the attack target publishes malicious IP addresses. Second, multiple mitigators adjust the configuration of their network devices to filter and drop the malicious packets. This second step is referred to as the actual "mitigation service". In a third step, the attack target evaluates the effectiveness of the mitigation service.

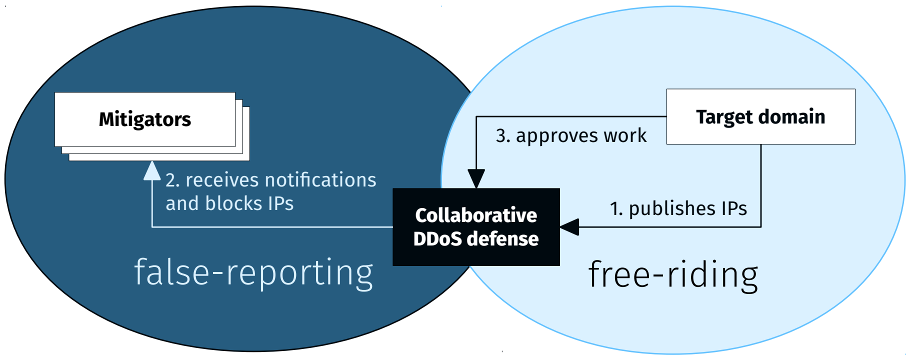


Figure 2.11: Social Dilemma of False-reporting and Free-riding

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. Moreover, similar social dilemmas exist in other research areas, *e.g.*, crowdsourcing [264]. Even in large Peer-to-Peer (P2P) networks, peers maintain lasting business relations and transact repeatedly [251]. This increases the potential benefit of such type of systems in P2P related domains. In Mobile Ad-hoc Networks (MANET), researchers also identified the same need, to provide incentive and credit-based mechanisms for cooperation among peers [59, 105]. Thus, incentive or reward mechanisms are required. Therefore, this Section emerged from a literature review about reputation and reward schemes in P2P and BC networks, in order to gather information about how to design such as system for cooperative DDoS defense.

2.6.1 Reputation Tokens

Electronic tokens are a common way to represent reputation. Liu *et al.* propose an identity and reputation management system on Ethereum [131]. A reputation coin termed *RpCoin* can be earned by completing tasks, which is like reputation itself, a non-transferable coin. The authors define two different types of tasks:

1. The reputation task contains a reputation claim beneficial to the publisher. All task participants cast a binary vote. Votes corresponding to the final result of the task will increase the reputation of the voter. The reputation of the task publisher is only increased if the result of the voting is positive.

2. The incentive task is used to discipline the peers. Without these tasks, the system is vulnerable to ballot stuffing and speculation of voting peers about final results. Therefore, these incentive tasks contain a negative claim about another peer in the system. The voting rules are similar to the reputation task but voters are rewarded for detecting fraudulent peers.

Reputation and reward are omnipresent in academia, citations and degrees can be interpreted as indicators for relevance. Sharples and Domingue analyzed the potential of BCs for educational record management [224]. In the educational economy, credits and degrees are stored in the BC. This economy is fueled by Kudos, an educational reputation currency. In comparison to *RpCoins*, Kudos is envisioned to be traded according to rules enforced by SCs. Concrete projects to realize such a system have already been undertaken in the Ethereum community.

Further, from an academic point of view, the Work.nation project is designing a similar infrastructure for decentralized portfolio management and skill attestation for working professionals [254]. Contributions of work during projects are verified by team members and the transparent system allows for rapid team building based on attested skills. The proof of concept is built using Ethereum.

As of another example targeting the decentralized prediction market, Augur is used to weight the reports about real world events [185]. Much of the theory behind Augur was developed in Sztorcs work on Truthcoin [179]. A prediction market in Truthcoin is called "oracle corporation" and consists of a customer and an employee layer with different currencies. Reputation is represented by VoteCoins (VTC) and is the currency transacted on the employee level. Another coin (called CashCoin) on the customer level is solely used to buy/sell shares of an expected outcome. The purpose of prediction markets is to forecast outcomes of real world events based on share prices. The market price of these shares reflect the expected probability of the outcome. VTCs are tradeable between the employees of the oracle corporation, but the total supply of reputation tokens in the system is fixed. VTCs are gained when a report about a real-world event is consistent with the consensus and lost otherwise. Only reputation owners can create reports about such events. VTCs are also withdrawn if the user does no longer participate in event-reporting. In such a case, the voter can maximize its utility by selling the VTCs, because they are liability as much as asset.

The economy of oracle corporations and the other examples mentioned above show, that a (tradeable or decaying) coin can be a meaningful abstraction for reputation. Like cryptocurrency, accumulated reputation is valuable and can indicate trustworthiness. However, these signals have to be treated with caution, since users owning an exceptional amount of tokens or reputation either now how to behave according to the rules, or found a way to cheat the system.

2.6.2 Event Reputation Factors

In the trust module proposed by Moinet *et al.* BC payloads trigger particular events [153]. The module is applied to establish trust in decentralized sensor networks. Every event is associated with a positive or a negative reputation factor, depending on the nature of the event from the originating transaction. Then the cumulative reputation score can be calculated by stepping through all past events of all blocks and process the associated reputation factors. Moinet *et al.* use an exponentially decaying function to present recent events more strongly (reputation decay). With this approach, the required minimum reputation of an agent to execute a specific action can be calculated based on the reputation factors. These minimum requirements can be designed trust defaultive for small sized networks and scale well with network size. A practical benefit of using events for reputation aggregation is that common BCs and SC languages have built in support for event management (*e.g.*, events in Ethereum or Hyperledger Fabric chaincode event listeners).

2.6.3 Reputation Thresholds

In mobile crowd-sensing, workers smartphone sensors are used to aggregate knowledge. The reputation can indicate reported measurement data quality and help to validate data [58]. Zhang and van der Schaar propose a threshold-based incentive protocol as seen in Figure 2.12. This system is designed to be robust against "false-reporting" and "free-riding".

The incentive protocol works like a state machine. Assume a newly activated user starts with a reputation θ at the social norm threshold h_k. At the end of each time period the requester evaluates

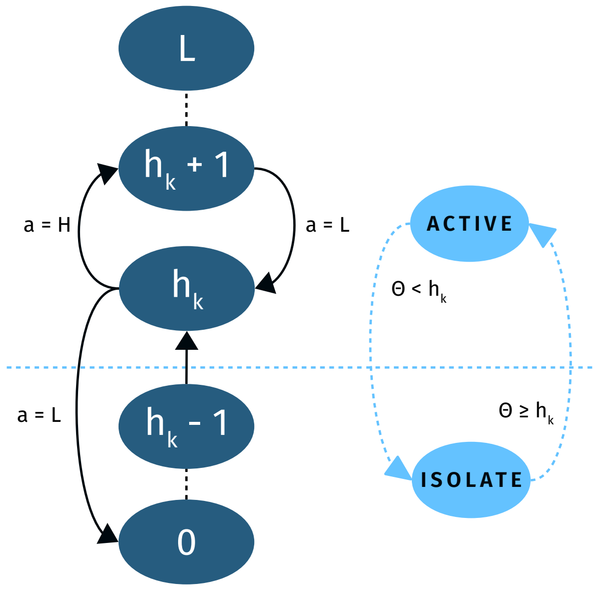


Figure 2.12: Threshold-based Reputation Model for Crowd-sourcing Applications [264]

the action (a) of the user. If the service requester is satisfied with the action, the evaluation will result in $a = H$ and the user is given one additional unit of reputation $(\theta = h_k + 1)$. If the task is not solved at the end of the time period, then the service requester rates $a = L$ and one unit of reputation is deducted from the user. We can assume ex-ante payment and thus a truth-telling requester.

As long as the users reputation (θ) is in the active region above the social norm threshold $(\theta >= h_k)$, the user is allowed to work and actions will be evaluated by the requester. In this active region the worker need to be in compliance with the social norm. Still, whenever $\theta = h_k$ and $a = L$, the user will be forbidden to solve tasks and its reputation will be reset. An isolated user receives one unit of reputation each time period and automatically builds up reputation over h_k time periods. Then, the user is reactivated and its actions are evaluated. The users reputation is again adjusted by the requester from this point onward.

2.6.4 Multi Signature Transaction (Multi-Sig)

As described in the previous section, the social dilemma still exists without the assumption of ex-ante payment. On the one hand, the requester of the service may not confirm the job and refuse to pay the requested (free-riding). The requested on the other hand would rather pretend to show effort in order to minimize costs (false-reporting). Contractual agreements to ensure the order of payments and voting rights can be seen as a solution to mitigate this dilemma. However, malicious requesters might still rate untruly.

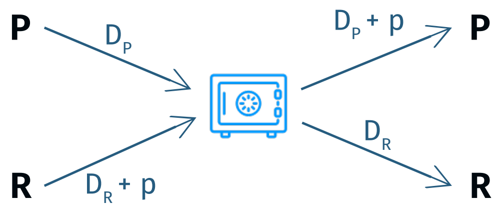
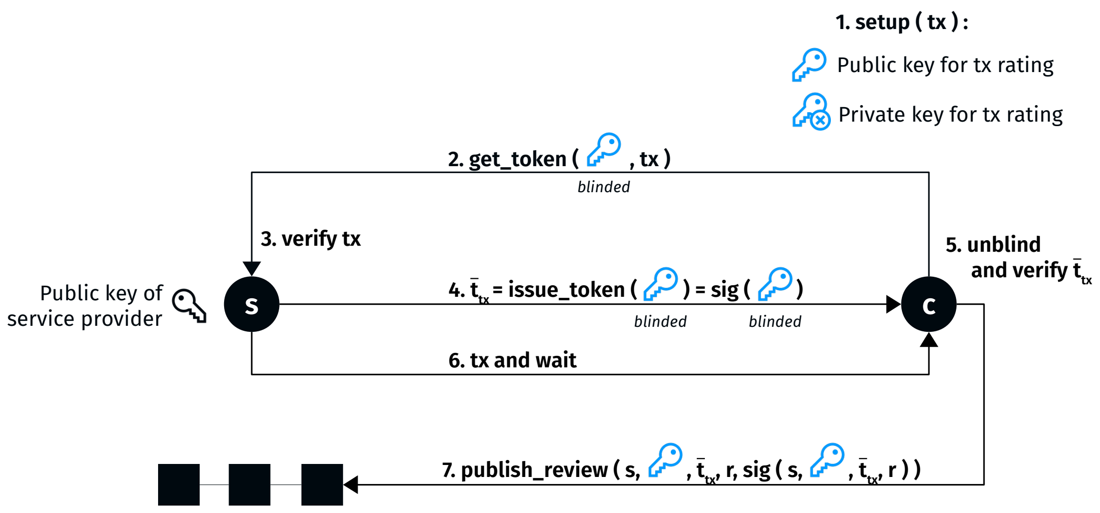


Figure 2.13: MAD (Mutual Assured Destruction) Transaction: Price (p) is Locked Inside the Contract until User (U) and Provider (P) Agree to Payout [116]

A SC protected by a multi-sig schema presents another solution to mitigate the same problem (see Figure 2.13). Considering that a service provider (P) and requester (R) both pay a deposit (D_R and D_P). Additionally, the requester has to pay the service price to the service provider. Then, funds D_R and D_P are protected with the Multi-sig, which means that they can only be spent if all parties agree. Kopp *et al.* call this concept Mutual Assured Destruction (MAD) transaction and use it to build a decentralized file storage with financial incentives [116]. As both parties store a safety deposit in the contract, they both have the incentive to resolve the transaction and retrieve the locked funds. Again, under the assumption of rational agents who want to retrieve the deposit, this contractual agreement can reduce counter-party risks. After the successful termination of the contract, the service provider receives the price (p) for the service and both parties retrieve their deposits.

2.6.5 Anonymous Feedback

Some applications require the reputation system to be privacy preserving [214]. In e-commerce or anonymous marketplaces [228], it might pose an issue that the producer knows which consumer gave a particular bad rating. Also, anonymous feedback mechanisms allow for honest feedback and remove the bias towards positive ratings [101]. An anonymous reputation system is not a contradiction since reputation can also be bound to temporary pseudonyms [58]. Schaub *et al.* designed a BC-based trustless reputation system which preserves the privacy and anonymity of the party giving the rating. This is achieved with blind-signatures on key pairs created for each transaction. In the end, the service provider does not know from whom he received the rating. This process is illustrated in Figure 2.14.



1. setup (tx) :
Public key for tx rating
Private key for tx rating
2. get_token (, tx)
blinded
3. verify tx
Public key of service provider
S
4. t̄tx = issue_token ( ) = sig ( )
blinded
blinded
5. unblind and verify t̄tx
C
6. tx and wait
7. publish_review (s, , t̄tx, r, sig (s, , t̄tx, r))

Figure 2.14: Anonymous Feedback with Blind Signatures [214]

First the consumer (c) prepares a public/private key pair for the transaction. Before the transaction happens, the consumer requires a blind signature on the public key for that transaction from the service client (s). It is not possible for the service client to relate the public key to either transaction or consumer at this point, since only the client knows the secret and random part of the blinded key. This random number allows the client to verify and unblind the token in combination with the public key of the service client. After a successful service delivery the consumer is advised to wait for some time until other consumers require service. Otherwise, a single feedback could still be linked to a single consumer. In the end, the consumer publishes a rating (r) about the service client (s) to the BC. This rating is valid because the blinded token was issued and made public by the service client beforehand and there is only one rating per token. The token from the producer gives the consumer the right to give feedback to a transaction. However, neither producer nor other clients know the identity of the consumer.

Limited token issuance for s can be an effective method to fight against ballot stuffing [214], because according to Cai and Zhu (2016) in [30], "this creates a trade-off between rating and profit for the seller" s. For every token, s can either choose to use it for a real transaction or pay the opportunity cost and use it to inflate its own reputation [214]. The opportunity cost equals to the unrealized profit from a legit sale of goods or services. Even though a limited supply of tokens can incentivize s to use the tokens for genuine purposes, s can never be completely discouraged from "buying" its own reputation [214].

2.6.6 Incentives to Vote

If the feedback in a reputation system is not mandatory, it might be required to provide the rater an incentive to leave feedback. Otherwise, peers might rather not give feedback. Carboni's feedback-based reputation system on top of the bitcoin BC incentivizes the rater with a voucher to give feedback for a transaction [31]. Figure 2.15 depicts the situation.

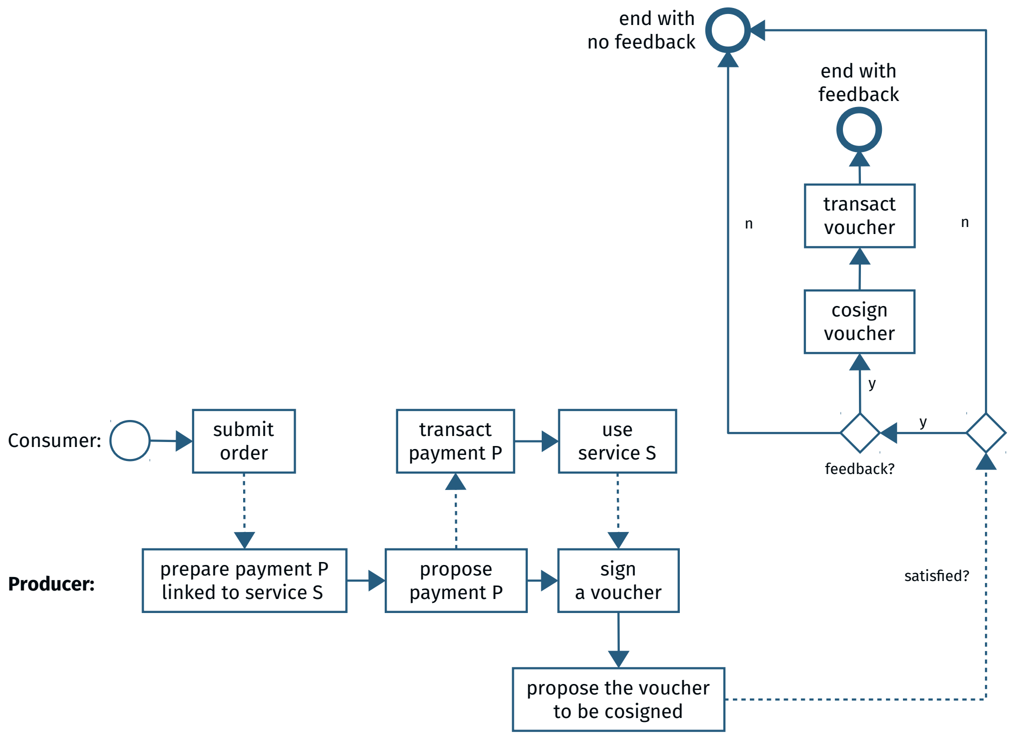


Figure 2.15: The Cosigned Voucher Creates a Monetary Incentive to Leave Feedback [31]

The voucher is linked to the past payment (P) and contains a vote fee and an incentive. The vote fee is a proportional share of the price (P) and represents the reputation increase. Pricier transactions result in higher reputation increase for the producer. The incentive is an amount chosen by the producer. When a consumer leaves feedback, the reputation of the producer increases by the vote fee and the consumer is rewarded with the incentive.

2.6.7 Insurance Models

Trust can be established based on reputation information [101]. A trust model helps the user to take decisions in concrete situations [127]. A reputation model however is more holistic, as it includes the view of all peers in the system about a given user [101]. Nevertheless, reputation and trust are closely related concepts.

Some trust models work in a similar fashion to insurance models. To better understand these models, the example of a supply chain with upstream suppliers and downstream customers is helpful. Peers trust each other up to some amount of money. In TrustDavis for example, peers can obtain weighted references from their neighbours [57]. If a user gives a reference, she would become liable to pay the reference price to any customer, if the supplier she gave the reference to cannot deliver the product or service. These references can be modeled as edges in a trust graph. In a trust graph each edge represents the maximum amount of money, that the originator is liable for or trusts the target of the reference with. A similar approach has been presented recently by Litos and Zindros [127]. Trust and reputation are not directly linked to rewards, but can be expressed in monetary terms using such schemes.

In Litos and Zindros, indirect trust is derived by relying on the peers that are trusted by your trusted peers (*i.e.*, transitive trust, whereas if *a* trusts *b* and *b* trusts *c*, then *a* trusts *c*). If an intermediary in the trust chain defects, she is free to either take a loss or steal the amount from a friend in the trust chain. Since this game can be played transitively the financial loss can be carried over to other trusted peers. The maximum trust (and equally potential maximal loss) of two peers is limited by the maximum flow in the trust graph between them.

Such a network of liabilities can serve as basis for concrete financial decisions. In a DDoS cooperative defense scenario, the target domain could assess if it is possible to trust a mitigator domain to solve a task, given the target domain can estimate the expected mitigation costs correctly. However, the reputation of a peer cannot be directly derived with this scheme, as reputation is mostly defined through a more collective view [101].

2.6.8 Reputation Engines

Besides reputation aggregation protocols, another important aspect of a reputation system is the reputation computation engine, where scores and metrics are calculated based on the inputs [101]. Most reputation systems in the P2P space are single-dimensional, meaning that only one factor (*e.g.*, number of contributions) serves as input for the system [60]. For the output metric there are a broad range of different models. Surveys and classification about the different types of reputation engines are presented by Jøsang *et al.* in [101] and Schlosser *et al.* in [218]. The presented overview is mainly based on these two works, with an emphasis on engines that seem suitable for the design of reputation systems in cooperative DDoS defense.

- **Probabilistic Engines**: The reputation score is mostly accumulated linearly, exponentially or calculated with help of probabilistic density functions (*e.g.*, the Beta reputation system) [13, 20].

- **Fuzzy Engines**: The peers behavior can be analyzed through fuzzy queries on the data stores holding this reputation, storing multi-dimensional reputation data [101].

In probabilistic engines, reputation is expressed as a probability. The expected value of the beta distribution forecasts the probability of a positive, future event x, based on the past binary events x and $\bar{x}$. The binary events represent positive and negative historical reputation ratings. The expected value of this distribution can be interpreted as a reputation score [13]. One particular example of a Beta reputation system applied to BC is found in the Topl protocol [42]. Topl is a proposed BC protocol to create profit sharing agreements with producers in emerging and frontier markets. The protocols reputation engine "Divine" builds upon a Beta reputation engine that facilitates due diligence and reduces counter-party risk [112].

The peers behavior can be analyzed through fuzzy queries on the data stores holding this reputation data. A reputation score is stored in tuple-form. $rep = (a, b, i, d, v)$ is the rating from a about b, in interaction or transaction i, on dimension or skill d, with v being the actual rating value (*e.g.*, in range $[-1, 1]$). With a fuzzy query $q = (a, \_, \_, quality, \_)$ an agent can retrieve and aggregate all ratings concerning the *quality* dimension from peer a, to any other peer ("_" is the wildcard) in any interaction and with any score value. Insights on the different interaction dimensions like quality, price and service time can be gained by executing and aggregating the results of such queries on the data store.

2.7 Security Basics and Blockchain Security

This section revisits basic security concepts that permeate the system proposed in this thesis. Subsection 2.7.1 presents fundamental security concepts. Subsection 2.7.2 presents how basic security concepts are used to identify and classify vulnerabilities in SCs. Lastly, Subsection 2.7.3 presents an overview of automated tools for identifying vulnerabilities in SCs.

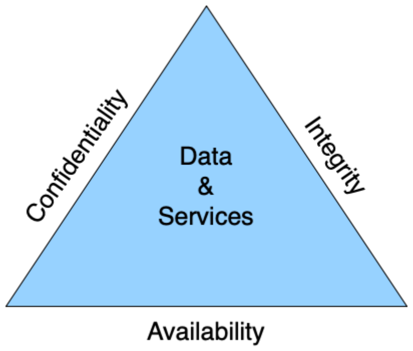


Figure 2.16: The CIA Triad. Adapted from [80]

2.7.1 Confidentiality, Integrity, and Availability

The Confidentiality-Integrity-Availability triad (also known as the *CIA triad*) is one of the fundamental concepts of information security. As Figure 2.16 shows, it aggregates three major principles of information security: confidentiality, integrity and availability [80].

- **Confidentiality**: restricted access to particular information or functionality by only people who are authorized.
- **Integrity**: implies that data is managed and updated correctly and, therefore, remains accurate over time, *i.e.*, information is not changed and its source is authentic.
- **Availability**: possible to access the data or functionality any time needed.

While directly DDoS attacks deplete the availability of the targeted service, the system that is proposed in this thesis also needs to address the CIA triad to prevent the system remedying DDoS from causing more significant impacts than the DDoS attack itself (for example, releasing confidential information from the victim to third parties). In this sense, confidentiality is addressed by the need to maintain information about the relationship between the target and private collaborative mitigator, ensuring confidentiality between the peer. Integrity refers to the impossibility of altering data once it is sent by a verified source, property whose BC-based platform can offer by default. Likewise, it occurs with availability since a BC-based system is fully replicated among members of the collaborative defense, and they maintain a synchronized version of the current state of the network. Therefore, one of the nodes' failure does not imply in Byzantine failures of the collaborative defense but only in isolated nodes.

2.7.2 Security of Smart Contracts

One of the first systematic reviews of Ethereum and Solidity vulnerabilities was published in 2017 by Atzei, Bartoletti, and Cimoli [12]. Overall, they identified and described 12 vulnerabilities dividing them into three categories: Solidity vulnerabilities (call to the unknown, gasless send, exception disorders, type casts, reentrancy and keeping secrets), vulnerabilities related to Ethereum Virtual Machine (immutable bugs, Ether lost in transfer and stack size limit) and general BC vulnerabilities (unpredictable state, generating randomness and time constraints).

In the same year, another classification of vulnerabilities in Ethereum SCs was proposed by Alharby and van Moorsel [5]. However, their study's scope included not only security vulnerabilities such as transaction-ordering dependency, timestamp dependency, mishandled exceptions, reentrancy, criminal SCs, and the lack of trustworthy data feeds, but also codifying, privacy, and performance issues.

Similar to Atzei *et al.* [12], Praitheeshan *et al.* [190] distinguish general BC vulnerabilities from the ones typical only for Ethereum and Solidity SCs. However, they put Ethereum and Solidity vulnerabilities together and added one more group of vulnerabilities – general software security issues. Overall, Praitheeshan *et al.* distinguishes three groups of SC vulnerabilities. The first one consists of BC related vulnerabilities and includes immutability, sequential execution, complexity, transaction cost, and human errors. The second one contains general software security issues (*e.g.,* buffer overflow, command injection, poor usability). Finally, the last group is a list of Ethereum and Solidity related vulnerabilities, including reentrancy, transaction-ordering, timestamp dependency, exception handling, call stack limitation, integer overflow, and underflow, unchecked and failed *send*, suicidal contracts, unsecured balance, use of *tx.origin*, unrestricted write and transfer, non-validated arguments, greedy and prodigal contracts, and costly gas patterns.

Chen created the most extensive list of vulnerabilities in Ethereum SCs [35], which contains in overall 44 security issues (six of them are marked as already eliminated). The vulnerabilities are distinguished by their location in Ethereum's architecture (application layer, data layer, consensus layer, network layer, or environment layer). Besides, Chen et al. divided the vulnerabilities into different categories depending on their cause, resulting in four primary categories include vulnerabilities related to SC programming (*e.g.*, reentrancy, integer overflow and underflow, use of *tx.origin*), to Solidity language and toolchain (*e.g.*, Type casts), to Ethereum design and implementation (*e.g.*, timestamp dependency and generating randomness) and, finally, to human, usability and networking factors (*e.g.*, weak password, broken access control). At the time of writing, this is the most comprehensive overview of Ethereum SC vulnerabilities.

Dingman *et al.* suggested another approach to the classification of vulnerabilities in Ethereum SCs. [63]. They applied the NIST Bugs Framework [168] to a list of known Ethereum smart contacts vulnerabilities. The NIST Bugs Frameworks is based on the data from Common Weakness Enumeration [53], its clustering Software Fault Patterns, Semantic Templates, and other sources and allows unambiguous classification of software weaknesses [168]. In order to map SC vulnerabilities to the NIST Bugs Framework, Dingman *et al.*. analyzed their cause, attributes, and consequences. As a result, the study presents a master list of SC vulnerabilities with matching categories and classes from the NIST Bug Framework [63].

Finally, studies focus on describing the most severe vulnerabilities instead of providing a systematic classification, such as Luu *et al.*. [137] discusses four vulnerabilities that can be used to manipulate SCs and gain profit by malicious actors. These vulnerabilities include transaction-ordering dependence, timestamp dependence, mishandled exceptions, and reentrancy. In addition to these four security issues, Dika and Nowostawski [62] describe other severe vulnerabilities, such as the use of *tx.origin*, call stack depth limitation, external calls, unchecked *send*, DoS with unexpected revert, blockhash usage, and gasless send.

2.7.3 Tools for Automated Security Audit

In general, three major types of automated security analysis can be distinguished: static analysis, dynamic analysis and formal verification. In accordance with the first approach, the programming code is scanned for vulnerable patterns without its execution. In contrast to this method, dynamic analysis is performed in a run-time. This approach simulates the behaviour of an attacker who is trying to find vulnerabilities by inserting malicious code and providing input to the code. Due to this technique, dynamic tools for automated security audit can identify vulnerabilities missed by static tools. Finally, formal verification methods rely on mathematical formal methods and theorems for the programming code validation and the prove of vulnerabilities [190].

All the previously described analysis types can deploy different strategies. For example, the static analysis can be performed on the bytecode using symbolic execution, control flow graph construction, pattern recognition and decompilation or direct on the Solidity code by rule-based analysis and compilation. The dynamic analysis always executes the bytecode. The possible strategies include the run-time execution trace, transaction graph construction, symbolic analysis and true positives and false positives validation. Besides, the formal verification analyses the bytecode with the help of theorem provers and program logics construction and the Solidity code by translating it to a formal language [190].

The overview of tools is presented in Table 2.4. The first tool for automated security analysis of SCs is presented by Luu *et al.* [137] in 2016 and is called Oyente. It performs static analysis and deploys a symbolic execution strategy. Symbolic execution (also called *abstract interpretation*) was introduced by Cousot and Cousot [50] in 1977. This strategy regards variables as symbolic expressions and checks if path conditions are satisfiable. As a result, Oyente is able to detect four types of vulnerabilities: transaction ordering, timestamp dependency, mishandled exceptions and reentrancy. When the tool was run on the 19'366 existing at that time Ethereum SCs, at least one vulnerability was found in 8'833 contracts which is about 46% of the total number [137].

Table 2.4: Comparison of Tools for Automated Security Audits

	Oyente	**Mythril**	**MythX**	**Securify**	**Securify 2.0**	**Remix**	**MAIAN**	**Manticore**
Analysis Type	Static	Static	Static	Static	Static	Static	Dynamic	Dynamic
Strategy	Symbolic execution	Symbolic execution	Symbolic execution	Formal verification + Symbolic execution	Formal verification + Symbolic execution	Formal verification	Symbolic analysis + Concrete validation of true/false positives	Symbolic analysis
Number of Vulnerabilities in Scope	4	12	37	18	38	21	3	12
Command Line Interface	✓	✓	✓		✓	✓	✓	✓
Web Graphical User Interface	✓		✓	✓		✓	✓	
Free of Charge Usage	✓	✓		✓	✓	✓	✓	✓

Another tool using the symbolic execution is Mythril [158]. It supports a security analysis of SCs not only in Ethereum, but also in Tron, Quorum, Vechain, Roostock, Hedera, and others [47]. The tool scans the bytecode and is able to detect a wide range of vulnerabilities including write to arbitrary storage location, arbitrary jump with a variable of function type, delegate call, weak randomness, deprecated opcodes, unprotected Ether withdrawal, exception handling, reentrancy, integer overflow or underflow, DoS with failed call, suicidal contracts and unchecked return value. Unlike Oyente, Mythril does not have a Web Graphical User Interface (Web GUI) and can be only run on the command line [158]. However, there is also a professional version of Mythril called MythX. In contrast to Mythril, it is not free of charge. Besides, it covers a wider range of vulnerabilities than Mythril [158].

The list of 37 supported security issues can be found in the SC Weakness Classification Repository [47] created and maintained by MythX team. Moreover, MythX can be integrated directly into developer tools such as Remix and Truffle which allows to perform security analysis continuously during the whole life-cycle of the project [46].

Another tool for automated security audit of Ethereum SCs combines symbolic execution with formal verification [245]. Securify was developed by ETH Zürich and its start-up ChainSecurity [34]. Its first version has a Web GUI [34] and detects 18 different vulnerabilities including reentrancy, transaction-ordering dependency, exception handling and arguments validation [245]. However, in January 2020, a second version of Securify [221] was announced [244]. It supports 38 vulnerabilities and is available only on the command line [221]. The new list of vulnerabilities is primarily based on the SC Weakness Classification Register [47] maintained by MythX team and ConsenSys. Besides, Security 2.0 analyses Solidity code and not bytecode as the previous version. Moreover, according to the developers, it is more precise and scalable than the original tool [244].

Automated formal verification is also offered by a built-in Solidity static analysis tool in Remix IDE [62]. At the time of writing, its latest version (0.10.1) was able to identify seven security issues, five gas-related issues, one ERC20 issue and eight miscellaneous vulnerabilities. Thus, the tool detects overall 21 different vulnerabilities [196]. In contrast to previously discussed tools, MAIAN [165] performs dynamic analysis. It deploys symbolic analysis and concrete validation of true and false positives. Instead of supporting a wide range of security issues, MAIAN focuses on three vulnerabilities: prodigal, greedy and suicidal contracts [166].

Another dynamic tool for automated security audit of SCs was developed by Trail of Bits and introduced by Mossberg at al. [157] in 2019. Manticore [242] relies on symbolic analysis of bytecode. It does not have a Web GUI but can be run from the command line and with Python Application Programming Interface (API) [242]. According to the information on the command line during the execution (see Figure X), Manticore detects 12 vulnerabilities including integer overflow, reentrancy, delegatecall and suicidal contracts.

Finally, some frameworks such as F* Framework [19], formalisation with Isabelle/HOL proof assistant [6] and FEther [259] implement formal verification analysis. These frameworks do not search for vulnerabilities as previously described tools but define correctness and safety properties for SCs and, then, prove them. Besides, the frameworks are semi-automated which means that a lot of manual work is required for their set-up [190].

2.8 Key Observations

This Chapter provided the theoretical basis on the core concepts involved in the thesis. Thus, aspects of BC and SC and reputation assessment were introduced in detail. Understanding these concepts is fundamental, and it is fundamental to compare the solution proposed in these theses with state of the art in collaborative defenses, and thus, verify under the proposed questions their benefits and drawbacks. For example, while BC and SC offer a relatively simple platform for the construction of a collaborative attack signaling solution, there are concerns regarding the confidentiality of the platform, considering transparency as one of the fundamental characteristics of BC.

Another relevant aspect of the platform is related to the temporal aspect of collaborative defense: the latency of communication between requester (victim of the attack) and requisite (cooperative mitigators). According to [265], negotiation must occur in the order of minutes for the effective re-stabilization of their services. Therefore, in addition to restrictions on the confidentiality of the data exchanged between the victim and the mitigator, a restriction on the negotiation time is imposed. This makes permissionless BCs like Bitcoin not viable due to the relatively high time for creating blocks as well as the impossibility of creating elaborated contracts between the parties.

Consensus mechanisms are the main component of a BC, determining aspects of reliability in the validators and performance aspects. Considering the requirement to restrict information access to a selected audience, and to make the exchange of information between the parties involved occur promptly, the Chapter analyzed the different consensus mechanisms available in BC that have been described in different stages. The first stage refers to the beginning of distributed systems, and the need to make the replication of information across different sites were resilient to failures. The second step in the first era of consensus mechanisms considers the possibility that some could even act maliciously, supporting Byzantine failures of a portion of the network's nodes. The explosion of BC-based applications has also encouraged the development of various consensus mechanisms to balance performance and confidentiality requirements in different ways, meeting the needs of these different applications. Thus, the Chapter evaluated these algorithms proposed in the second era in permissioned and permissionless, in which the first allow the selection of validators and the second area based on a competitive process seeking to elect validators within the network.

This Chapter also introduced one of the main elements facilitating a collaborative defense based on BC - SCs. As introduced in subsection 2.5, the combination of permissioned BC with the collaborative logic implemented in SC allows for transparency in the collaborative logic within the context of the alliance, as well as the verifiability of reputation and exchange of incentives. While BC and SC are relatively new concepts concerning reputation management and incentives in P2P systems, the latter is of equal importance. In addition to the need to reduce technical complexities in the deployment and operation of a collaborative defense, there is a need for incentives for third parties (*i.e.*, mitigator) to perform mitigation services using their infrastructure. Considering that such a task typically incurs operational expenses, there is a need to add an incentive scheme to cover expenses related to this mitigation service. As a consequence of the addition of incentives covering such expenses, there is a need to verify the participants' reputation as a way to reinforce the incentive to good behavior within this alliance.

Therefore, this Chapter introduced reputation and incentive-based mechanisms considering its application in a BC-based platform, which introduces a novel way to decentralize trust creating a certification infrastructure that records all traces of transactions, further allowing participants to verify and record histories of transaction outcomes. Furthermore, BC provides the underlying structure of a financial reward scheme through cryptocurrency. This currency represents an incentive to behave honestly if actions are rewarded accordingly, whereas those financial rewards and reputation can be treated independently or combined. Therefore, the main challenges listed in collaborative defenses can be addressed by a BC-based solution, including incentives and tracking member reputations within a specific context, or regional alliance between participants aligned under regulations or legislation in compliance. However, it is essential to note that this Chapter did not cover aspects related to the effective mitigation of attacks. Autonomous systems implement their own specific and heterogeneous technology within their network management system. Thus, the Chapter focused on the aspects mentioned above of the collaboration platform.

3
State-of-the-art of Cooperative Network Defenses

On the one side, the points closest to the victim present a higher concentration of traffic in a similar way to a funnel, making it relatively easy to detect whether a DDoS attack is happening due to sudden increases in inbound traffic. On the other side, devices sending attacking traffic represent the source points scattered through multiple regions, impairs the detection of an attack unless a large number of attack flows are initiated from that specific source.

Blockchain provides an interesting alternative toward a cooperative defense by offering a platform relatively simple to deploy and operate (in contrast to existing solutions), able to increase trust levels, and provide means toward the exchange of financial incentives. Therefore, this chapter provides an overview of the existing mechanisms following the classification described in Section 3.1 to list how the challenges listed in different dimensions described in this thesis are addressed by the existing solutions.

3.1 Classification of DDoS Defense Mechanisms

In source-based mitigation, the defense system tries to identify malicious outgoing traffic at an AS (Autonomous System) and block the attack traffic upon detection. In contrast, destination-based mitigation analyzes incoming traffic at the edge routers and access routers of the attack-target AS to filter the traffic before it reaches the AS [265]. Both perspectives have in common that they only consider individual, isolated ASes, either on the attack's source or destination side. This gives them the characteristic of a centralized defense approach since no coordination among the ASes involved in the overall attack exists. This gives them the characteristic of a centralized defense approach since no coordination among the ASes involved in the overall attack exists.

Furthermore, network-based mechanisms are deployed inside intermediate networks (*i.e.*, between source and destination AS), usually leading to high storage and processing overheads at the routers. Overhead occurs due to the need to infer trust values for the access routers by employing calculations, decision making, and trust negotiations among the routers to efficiently and effectively detect and filter the malicious routers is the ultimate goal of these mechanisms. Lastly, while previous approaches (source-, destination- & network-based) may work without coordination and cooperation amongst the various involved parties, hybrid mechanisms are deployed at multiple locations, and there are coordination and cooperation between the locations. DDoS defense mechanisms are summarized according to their placement as follows:

- **Source-based**: mechanisms deployed at the egress point, typically being difficult to detect attacking traffic due to their highly distributed nature.
- **Destination-based**: traditional approaches and mechanisms deployed at the ingress point of attacking traffic usually overwhelmed by attacking traffic.
- **Network-based**: deployed at intermediate networks involving statistical approaches to detect ongoing attacks, or, when cooperative, based on trusted destination points (*i.e.*, victims).
- **Hybrid**: involve a combination of all other mechanisms in a cooperative setup.

Furthermore, the study is based on a bibliographic review whereas data (*i.e.*, works related to each category) is obtained from sources indexed on IEEE, ACM, and Springer bases. Furthermore, works are described and evaluated according to the categories presented in Section 2.2 of Chapter 2 (Distributed Denial-of-Service Defense). Then, works are according to their categorization form *i.e.*, network placement according to [265], and subcategories, as proposed in [151]:

- **Architectural Types**: on-premises (centralized or distributed), off-premises (distributed or decentralized).
- **Activity Types**: proactive, reactive, passive.

- **Cooperation Degree**: autonomous, cooperative, interdependent.

To maintain this Chapter focused on the goal of discussing different approaches, the technical description of each work is presented in Appendix A. It is important to note that this method applies to the first three placements: source-, destination- & network-based. As these do not support a holistic view of DDoS attacks nor cooperating with various mechanisms, they provide support for hybrid mechanisms. Therefore, such distinction is relevant because the hybrid category's mechanisms may or not be based on the functioning of these mechanisms in different locations on the Internet. The comparison of hybrid mechanisms is based on the challenges described in this thesis:

- **Technical**: complexities of operation and deployment
- **Social**: concerning data privacy, integrity, and reputation management
- **Economical**: whether solutions provide support for incentives
- **Legal**: regarding the conformity across different legislation's

As highlighted in [183, 265] the analysis of how works address challenges, enabling to verify the advantages and disadvantages of the solution proposed in this thesis. Such an analysis also extends the work of categorizing and classifying collaborative defenses done in previous surveys by explicitly defining the road-map for broad adoption of cooperative defenses against large-scale DDoS attacks.

3.2 Source-based DDoS Mechanisms

The mechanisms presented in Table 3.1 are source-based approaches to decrease the attack footprint at its origin. Advantages of these mechanisms are the low amount of traffic required to be analyzed, the ability to stop an attack right at the source, easy tracing of the attack back to its origin and the ability to dedicate more resources toward the mitigation efforts since the attack itself does not consume many resources due to its small footprint [152]. Thus, most often, they can be placed at a source's edge router of the local network or the access router of an autonomous system connected to the source's edge router. They are not entirely sufficient because (a) the attack's source(s) can be distributed, (b) the differentiation between legitimate packets and malicious is confusing, and (c) the motivation to install such services is low since it is unclear who pays the services [265].

Source-based mechanisms typically involve active approaches to traffic analysis aimed at identifying patterns that lead to attacks. It is unanimous that stopping DDoS attacks at the source is the most efficient alternative (in contrast to network and destination-based approaches) to prevent other intermediate points and recipients from being overloaded. However, source-based mechanisms fail to obtain an overview of the attack, which is highly distributed. Among the main challenges of the mechanisms proposed in this category are the difficulty of identifying these patterns and the overhead in terms of processing and storing information on network devices, which must actively compare traffic patterns with attack patterns. The details of the works described are summarized in Table 3.1 (full description of each work is presented in Appendix A).

Table 3.1: Comparison of Source-based DDoS Defense Mechanisms

Approach	**Architectural Type**	**Activity Type**	**Cooperation Degree**	**Short Description**	**Year**
MANANet [140]	On-premises, distributed	Proactive	Autonomous	Reverse firewall that can cooperate with similar MANANet-enabled routers	2020
D-WARD [150]	Off-premises, distributed	Reactive	Cooperative	Distributed and cooperative DDoS detection and mitigation system deployed at multiple sites	2002
MULTOPS [76]	Off-premises, distributed	Reactive	Cooperative	Similar as D-WARD but optimizing the data structure to alleviate the overhead on routers	2001
Lu *et al.* [136]	On-premises, centralized	Proactive	Interdependent	SDN-based approach performing a statistical inference on flows to detect early-stage attacks	2016
Priyadarshini *et al.* [193]	On-premises, centralized	Reactive	Interdependent	SDN-based approach using a deep learning model to detect attack patterns	2019
Yi *et al.* [260]	On-premises, centralized	Proactive	Interdependent	Provide support for collaborative tools by accumulating (legitimate) host information to prevent IP spoofing	2008
Soldo *et al.* [227]	On-premises, centralized	Proactive	Interdependent	Builds an in-memory ACL (white or blacklist) based on coordinated, suspicious, traffic sent by hosts	2011
Badis *et al.* [15]	On-premises, distributed	Reactive	Interdependent	Cloud detection system that relies on a cooperation among hypervisors to detect malicious outgoing traffic patterns	2015

Although the D-WARD [150], MULTOPS [76] and Badis *et al.* [15] solutions are cooperative, they are not interoperable. Therefore, these cooperate between homogeneous instances of different organizations but not between different solutions from different organizations. The degree of cooperation is also reflected in the architectural type, resulting in off-premises operations as a distributed solution acting reactively (*i.e.*, upon request in a master-slave mode) to install filters close to the source.

In general, source-based mechanisms require the analysis of outgoing traffic patterns to compare them with attack patterns. In this sense, the need to reduce overhead in network devices becomes exceptionally relevant and evident in the analyzed works. Also, the advancement of recognition techniques based on machine learning gains space combined with the offloading of management and network functions for centralized servers (*e.g.*, SDN and VNF). At this point, works such as Yi *et al.* [260] and Badis *et al.* [15] can be highlighted in the use of these relatively new concepts to alleviate the burden on the network infrastructure.

3.3 Destination-based DDoS Mechanisms

Destination-based approaches toward DDoS mitigation (*cf.*, Table 3.2) often face the challenge of handling high traffic bandwidth arriving at routers. This leads to high resource usages on detection and mitigation systems due to the overhead of analyzing the attack traffic near to its destination [265]. Once in destination-based approaches attack traffic is untouched throughout its preceding route, resources are wasted which could have been saved if the traffic would have been classified before. The alternative of destination-based mitigation relies on block-holing and traffic shaping to prevent the system from falling over.

The advantage of the late response to the attack traffic is that it is much easier to identify malicious traffic based on its volume and destination[265]. However, in attacks involving thousands of devices that individually do not spend large amounts of traffic, it is not easy to distinguish between legitimate and malicious users. Another definite advantage is the direct incentives for the operator of destination-based mitigation systems: By running such a system, they can directly protect hosts within their AS from attacks, which represents an added value to all or their users. Following defense mechanisms demonstrate different approaches to mitigate DDoS attacks in a destination-based manner. These mechanisms take place at the edge routers or access routers of the AS deploying them [265].

Destination-based mechanisms are in a unique position to analyze the entire attack traffic since they are close to the attack target. This can trace back the attack origins to better identify malicious traffic and easily differentiate it from regular traffic. However, tracing back attack traffic to its origin is not a straight forward process, since attackers often spoof their IP addresses, making it much harder to find the actual origin [100]. IP Traceback approaches are divided into preventive and reactive approaches [100]. The less common preventive approaches try to block packets originating from spoofed IP addresses as a direct measure against DDoS attacks [100]. This is succeeded by examining every packet arriving at the edge or access router of an AS, which can become quite resource-intensive with the increasing DDoS attack volume.

While mechanisms deployed at the source do not have an overview of the attack, defense mechanisms deployed at the target (*i.e.*, target) are overloaded by volumetric attacks. In large-scale DDoS attacks, destination-based approaches typically fall short on the packet-level analysis to distinguish legitimate users and attackers. Thus, there is an increasing demand for increasing the processing power to analyze packets and compare them with recognized traffic patterns. However, there is the oppor-

Table 3.2: Comparison of Destination-based DDoS Defense Mechanisms

Approach	**Architectural Type**	**Activity Type**	**Cooperation Degree**	**Short Description**	**Year**
MIB [29]	On-premises, centralized	Proactive	Autonomous	Utilize information available on router's MIB groups IP, ICMP, and others for DDoS detection	2001
Pi [258]	Off-premises, distributed	Proactive	Interdependent	Packet marking approach providing basis for IP Traceback solutions	2003
Takemori *et al.* [238]	On-premises, centralized	Reactive	Autonomous	Intended for Personal Computers based on a whitelisting of IP addresses whereas a daemon verifies the system in non-usage periods	2008
HCF [99, 250]	Off-premises, distributed	Reactive	Interdependent	Packet marking and filtering approach based on the IP Traceback approach	2003
Yu *et al.* [263]	Off-premises, distributed	Reactive	Interdependent	Packet marking and filtering approach based on flows' variation instead of probabilistic or deterministic packet marking	2010
Packetscore [111]	On-premises, centralized	Reactive	Autonomous	Define scores on a packet-level to identify and prioritize whitelisted sources	2006
HIF [182]	Off-premises, distributed	Reactive	Interdependent	Edge routers keep a historic of IP addresses in order to whitelist trusted sources	2003
FlowGuard [98]	Off-premises, distributed	Reactive	Interdependent	Intended for IoT networks where edge routers can detect and perform a coordinated mitigation	2020
Cziva *et al.* [54]	Off-premises, distributed	Reactive	Interdependent	Traceback approach based on VNFs where a domain can request to push VNFs for mitigation	2017
Cloud-based Protection [3, 43]	Off-premises, distributed	Proactive	Autonomous	Cloud-protection services acting as a proxy to analyze and filter potential DDoS attacks	2020

tunity for approaches seizing regular traffic (*i.e.*, without bursts the network traffic) periods to derive traffic patterns from legitimate users, creating whitelists.

Once an attack is detected, which is a common task to all mechanisms listed in Table 3.2, two divergent approaches are noted: IP traceback and White-listing. Traceback approaches rely on compatible routers along the path for acting on malicious packets flagged at the destination. These mechanisms are mostly interdependent and reactive, *i.e.*, they can act autonomously when there is no compatible router and request compatible routers to mitigate malicious packets by pushing filters near the attack source (*e.g.*, Pi [258], HCF [250], Yu *et al.* [263], HIF [182], Cziva *et al.* [54], and [98]). There are discussions about these approaches' practical effectiveness while the technical difficulties (*i.e.*, heterogeneous network equipment) and social difficulties with the need to rely on third-party requests changing the traffic [183, 265].

Whitelisting-based mechanisms feature variations of traditional in-house tools, such as Intrusion Detection System (IDS), Deep Packet Inspection (DPI) systems. For example, HIF [182] stores a packet history to identify in traffic patterns which sources are legitimate (and therefore white-listed). Over a while, sources display constant traffic patterns without spikes in the traffic volume for short periods, and it is possible to place these addresses as white-listed. In the event of an attack, white-listed sources are prioritized. This same pattern appears in different mechanisms varying the underlying technology and approaches toward an efficient use of hardware (*e.g.*, Takemori *et al.* [238], MIB [29], Packetscore [111]).

Destination-based mechanisms are similar to the source-based either done on the edge router or at the target's autonomous system. Nonetheless, most of the source and destination-based mechanisms cannot accurately detect and mitigate the attack. While source-based mechanisms do not have an overview of the entire attacking sources, it is not straightforward to detect widely distributed attacks. In the case of destination-based mechanisms, resources to detect and mitigate the attack are typically overwhelmed by incoming traffic, which ultimately causes legitimate users to be prevented from accessing services (*i.e.*, the service is denied) due to the difficulty of detection systems in distinguishing traffic patterns from attackers or legitimate users.

In this case, the conventional alternative is to increase the capacity to detect and mitigate traffic at the destination, which for many organizations, is not a feasible strategy as it requires a significant investment. In this sense, cloud-based services such as the ones provided by Akamai [3] or CloudFlare [43] appear as a viable alternative, serving as a proxy to the destination and performing with a larger pool of hardware and software resources. However, it is observed that despite having more resources, there are some problems in the adoption of Cloud-based Protection Services (CPS):

- **Security**: organized in terms of confidentiality, integrity and availability as below:
  - **Confidentiality**: incoming traffic is is directed to the protection service that acts as a proxy. Thus, CPS pose a vulnerability for companies whose confidentiality of traffic patterns is relevant for marketing reasons.
  - **Integrity**: due to the centralization of traffic in the CPS, it is necessary to trust that the CPS will not modify packets or send to other destinations.
  - **Availability**: large traffic processing capacity ends up being diluted among the different CSP customers. Therefore, since the CSP is a single organization (centralized or distributed), it is liable to suffer periods of unavailability in the event of large-scale attacks.
- **Performance**: concerns availability in the sense that more resources typically represents more traffic absorbing capacity. However, the efficiency of hardware and software is important to determine performance in terms of Quality of Service (QoS). Like any traditional cloud service, it is important to specify the service quality parameters in the Service Level Agreement (SLA), so that even with the increase in the number of consumers, the QoS standards are maintained.

Concerning the technologies on underlying networking equipment, there is a tendency for virtualization of resources based on SDN [98] and NFV [54]. These concepts allow mechanisms to make a more efficient use of the hardware regardless of which is its placement in the network, *i.e.*, regardless of source or destination-based. While NFV allows mitigation functions to be distributed as a code that can be executed on generic servers acting on traffic, SDN allows a global awareness of organizations about changes in network flows and detection of attacks.

3.4 Network-Based DDoS Mechanisms

The goal of network-based Mechanisms is to stop an attack in the network between source and target and help the source and destination-based mechanisms to do their work correctly. Since source-based mechanisms have difficulties finding attack patterns and destination-based mechanisms are usually overloaded, network-based mechanisms have been proposed to address this problem and help both source and destination-based mechanisms carry out their duties more accurately. These mechanisms are deployed within networks, on routers, often extending the source-based mechanisms by providing the marking of suspicious packets allowing the route of individual suspect packets.

Network-based mechanisms attempt to balance inefficient aspects of source and destination based mechanisms. While source-based mechanisms have difficulties in detecting attacks since they lack an overview of the attacking traffic, network mechanisms have a certain similarity in that they are also not the final destination of the attack traffic. Thus, only a portion of the traffic passes through traffic routers. Another obstacle is the costs of implementing detection and mitigation mechanisms, which impose not only drawbacks in terms of performance but also financial costs to deploy and operate DDoS detection and mitigation tools. Since operators implementing network-based approaches are not the final destination of the traffic, carrying out inspections, analyzes and mitigations impose these drawbacks that must be taken into account in a cooperative defense, *i.e.*, § there is a need for incentives for operators acting as traffic transit to enforce some measure to combat attacking traffic.

Since network-based mechanisms are not the target of the DDoS attack, costs involved can become an impediment for their implementation. Thus, it is observed that many of the mechanisms listed work in an interdependent manner (*cf.*, Table 3.3 - note that the full description of each work is presented in Appendix A), *i.e.*, they enable collaboration between domains using similar approaches. Hence, transit domains may also be the target of an attack and require cooperation from another mechanism that was once the target.

Table 3.3: Comparison of Network-based DDoS Defense Mechanisms

Approach	**Architectural Type**	**Activity Type**	**Cooperation Degree**	**Short Description**	**Year**
PPM [177]	Off-premises, distributed	Reactive	Interdependent	Probabilistic packet marking to flag packets that potentially belong to a DDoS attack	2001
DPF [176]	Off-premises, distributed	Reactive	Interdependent	Distributed packet filtering that, based on PPM, disclose a method to propagate filters across DPF enabled devices	2001
Sterne *et al.* [233]	Off-premises, distributed	Reactive	Interdependent	Based on the concept of active networks to distribute intrusion detection rules	2002
PFS [222]	Off-premises, distributed	Reactive	Interdependent	Propabilistic filter scheduling approach aiming to alleviate PPM's intensive hardware usage by sampling packets	2011
APFS [223]	Off-premises, distributed	Reactive	Interdependent	Optimize PFS toward an Adaptive approach, which is calculated based on hop count, sender availability and filter availability.	2013
CITRA [219]	Off-premises, decentralized	Reactive	Interdependent	Proposes a cooperative network of intrusion detection and traceback systems between CITRA-enabled agents	2001
SHDA [90]	On-premises, centralized	Reactive	Autonomous	An SDN application to detect slow HTTP DDoS attacks by analyzing flow counters	2018
MiddlePolice [132]	Off-premises, distributed	Reactive	Interdependent	Edge routers keep a historic of IP addresses in order to whitelist trusted sources	2016

All mechanisms are based on the reactive type of approach, in which it is required to define metrics and analyze traffic to determine whether any mitigation measures are necessary. It is also noted that most approaches operate on routers with probabilistic packet marking and filtering of marked packets. Among these, PPM [177] stands out as an approach to aiming to reduce performance overhead on routers to define whether packets are marked as a suspect without taking into account mitigation measures. These were defined in later work with the Distributed Packet Filtering (DPF) [177], in which domains with PPM-enabled routers could propagate specific filters for these packets in a traceback approach.

These works also have a degree of interdependent cooperation them subject to independent operation and cooperation, when necessary. The exception is the CITRA [219] approach, which has an exclusively cooperative approach based on an interconnected network of intrusion detection systems. These CITRA-enabled devices maintain an isolated communication channel (overlay network) to exchange information about white or blacklisted IP addresses and maintenance of these lists. CITRA has a similar operating approach to Sterne *et al.* [233], PFS [222] and APFS [223], with the exception that it is exclusively focused on IDS systems. PFS and APFS, however, are not necessarily cooperative and operate based on statistical information collected from routers.

Among the approaches listed, the ones more efficient in performance work with statistical analysis extracted from routers. Since these metrics can be sampled without additional overhead, they can provide data for a detection approach based on this data, as presented in PFS and APFS. However, mitigation-related activities can be more complicated since the detection of attacks is similarly sophisticated. Network-based mechanisms lack a full view of the attack as destination-based mechanisms and may result in loss of performance to enforce a blacklist, and legal aspects of specific addresses are improperly blocked.

It is also essential to highlight the differences between network infrastructure types that network-based mechanisms may operate. For instance, in core networks where Content Delivery Networks (CDN) operate, packet-level detection and mitigation approaches can result in loss of performance and, consequently, content quality (as reported in [159] concerning the impact of the traffic of growing CDNs and streaming services). However, for intermediaries that operate access (edge) and aggregation networks, and consequently, deal with smaller volumes of traffic in contrast to core networks, cooperation can be viable as long as there are incentives. In this sense, approaches must be efficient in detecting and mitigating attacks and have incentive mechanisms for more organizations to participate in cooperative mitigation of large-scale and highly distributed attacks.

3.5 Hybrid DDoS Defense Mechanisms

The general idea behind combining (source, network, and destination) mechanisms is to extend their detection and mitigation capabilities across multiple networks. A significant difference concerning mechanisms previously presented is the architectural type that refers to the differences between distributed and decentralized architectures. A distributed architecture has central servers that effectively distribute information to other nodes acting as slaves, but a decentralized architecture every node in the network acts as master and slave. For instance, source-based approaches such as D-WARD [151] or MULTOPS [76] often fail to identify the attack as a whole and only see parts of it. Destination-based approaches can detect the entire attack volume. However, this quickly turns into their most significant weakness since they have to mitigate massive traffic amounts at once.

Hybrid mechanisms can better cope with these highly distributed attacks by allowing the individual defense systems to communicate among each other, attack information can be shared to be able to react to attackers that might have remained undetected on a source system. In contrast, a destination system might have identified the attack affecting one of the hosts within their domain. However, hybrid mechanisms, in addition to extending detection and mitigation capabilities, combine the negative aspects of other mechanisms, with the addition of a coordination network layer in order to communicate actions between cooperative instances (*cf.*, Figure 3.1).

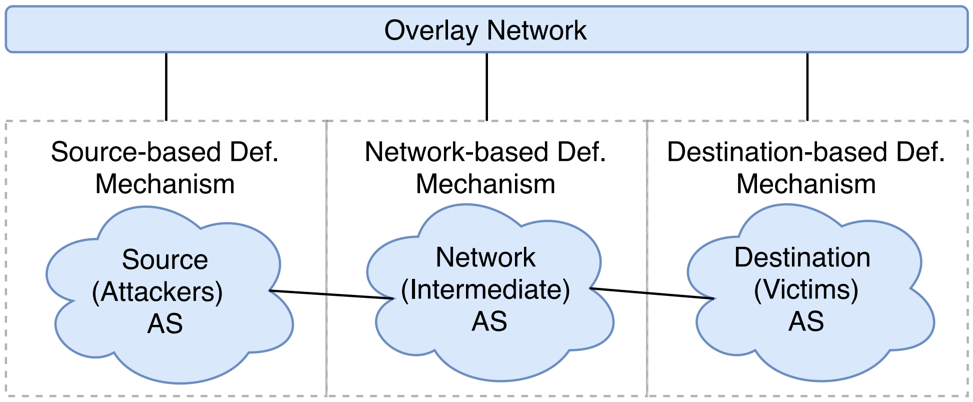

Overlay Network
Source-based Def. Mechanism
Network-based Def. Mechanism
Destination-based Def. Mechanism
Source (Attackers) AS
Network (Intermediate) AS
Destination (Victims) AS

Figure 3.1: Communication Overlay on Hybrid DDoS Defense Mechanisms

While DDoS attacks typically have a higher organizational capacity (since they inadvertently infect connected devices), collaborative defenses present a series of challenges for them to become active. Those challenges were identified in surveys [183, 265], which are broadly categorized herein as (a) the high complexity of operation and coordination; (b) the need for trusted and secure communication; (c) lack of incentives for the service providers to cooperate; (d) understand how the operations of these systems are affected by different legislation, regions, and countries. It presents a comparison of how characteristics perform according to challenges in Table 3.4 (note that the full description of each work is presented in Appendix A).

Secure Overlay Services (SOS) [103], Pushback [95] COSSACK [175], and DefCOM [169] 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), Pushback, COSSACK, and DefCOM based their approach on detection and enforcement points in access networks. However, these approaches required changes in routers or requiring the sources to be registered, thus, presenting a high complexity of coordination and operation.

SDN-based solutions allow greater agility to enforce decisions that require a global network view. Bohatei [69] demonstrates the scalability and performance advantages of using SDN in conjunction with VNF to build a DDoS defense system on top of proven, existing mitigation components. Leveraging NFV with SDN makes Bohatei's solution up to 5.4 times more cost-effective than fixed defense facilities [69]. Utilizing existing mitigation solutions like Snort, Bro, and IPtables, make sure that proven defense technologies are employed. However, although the combination of SDN and NFV simplifies the technological aspect, the solution does not include cooperative other aspects of the mit-

Table 3.4: Analysis of Cooperative Defense Challenges

Work	Cooperative Defense Challenges				Year	Activity Type
	Technical	Social	Economical	Legal		
SOS [103]	●	✗	✗	◐	2002	Proactive
Pushback [95]	◐	✗	✗	◐	2002	Reactive
COSSACK [175]	◐	✗	✗	◐	2003	Reactive
Mayday [8]	◐	✗	✗	◐	2003	Reactive
i3 [234]	●	✗	✗	✗	2004	Proactive
Koutepas *et al.* [118]	●	✗	✗	◐	2004	Proactive
AITF [10]	◐	◐	✗	◐	2005	Reactive
DefCOM [169]	◐	✗	✗	◐	2006	Proactive
Zhang *et al.* [266]	◐	✗	✗	✗	2006	Proactive
Speak-up [248, 249]	◐	✗	✗	◐	2006	Reactive
Chen *et al.* [39]	◐	✗	✗	◐	2007	Reactive
DOW [261]	◐	✗	✗	◐	2007	Reactive
StopIt [129]	●	✗	✗	◐	2008	Reactive
TMH [262]	●	◐	✗	●	2009	Proactive
Velauthapillai *et al.* [247]	●	✗	✗	◐	2010	Proactive
NetFence [130]	◐	✗	✗	◐	2010	Reactive
FireCol [72]	◐	✗	✗	◐	2012	Proactive
Bohatei [69]	◐	✗	✗	◐	2015	Reactive
Sahay *et al.* [208]	●	✗	✗	◐	2015	Reactive
Chin *et al.* [40]	●	✗	✗	◐	2015	Reactive
Giotis *et al.* [78]	●	●	✗	◐	2016	Reactive
Steinberger *et al.* [232]	◐	✗	✗	✗	2016	Proactive
CIPA [38]	●	✗	✗	◐	2016	Reactive
CoFence [195]	●	✗	✗	◐	2016	Reactive
IETF DOTS [167]	◐	✗	✗	◐	2017	Proactive
Hameed *et al.* [85]	◐	✗	✗	◐	2018	Reactive
OverWatch [86]	●	✗	✗	◐	2018	Reactive
CoChain [1]	●	✗	✗	✗	2019	Proactive
Spathoulas *et al.* [229]	●	✗	✗	✗	2019	Proactive
Essaid *et al.* [66]	●	✗	✗	✗	2019	Proactive
Pavlidis *et al.* [180]	●	●	✗	◐	2020	Reactive

● = provides property; ◐ = partially provides property; ✗ = does not provide property igation, such as how cooperative mitigation requests can impact operational expenses (economic) or the potential damages to the public image of a domain in cases of information leaks (social).

It is also important to distinguish two types of hybrid mechanisms: (a) cooperative overlay networks to share information and (b) capability-based that extends (a) enabling peers to enforce actions on traffic.

- **(a)**: Cooperative overlay networks are distributed frameworks enabling service and information exchange among all defending nodes. All nodes should collaborate and coordinate such that the mitigation is successful and effective. The nodes in this approach are specialized for their particular purpose (*e.g.*, detection or trace-back), depending on the virtual distance to the source. Thus, it is important that they can communicate all the time, despite the attack's force.

- **(b)**: Capability-based mechanisms let the victim explicitly authorize and allow the traffic it wants to receive. The routers along a path check whether the traffic is legitimate. If not, *i.e.*, , there is no permission that a system can send data to the victim; the data flow is stopped. These mechanisms are always active, and their processing and memory overhead is high.

Technical Challenges

The technical challenges deal with the set of concepts and technologies distributed or decentralized to make communication and effective mitigation actions possible. Thus, all works present different alternatives to propose a collaborative defense, including mitigating actions or just the signaling (capability-based) and detecting attacks (cooperative overlay meshes). Within the technical challenge, it is possible to make a broad distinction in these approaches between hardware (◐) and software-based (●). While hardware-based approaches require network equipment (*e.g.*, router, firewall, IDS, IPS) to be modified to operate with the overlay network and take action on-demand, software-based approaches seek to minimize interference in the underlying infrastructure operating on standard off-the-shelf servers. Minimizing interference in the underlying infrastructure by software-based options is essential due to the Internet's high heterogeneity. However, while there may be drawbacks in terms of performance, the solution may be more compatible with a more significant number of participants.

Social Challenges

The social challenge concerns how overlays networks are composed, having as the main challenge the establishment of trust between independent organizations. Several proposals listed in Table 3.4

have the assumption that the overlay network members are trusted. However, in a heterogeneous environment such as the Internet, this assumption limits the solution's applicability to consortia with a specified complexity, since different organizations and, often, competitors can use privileged information to obtain competitive advantage [81]. Notwithstanding, it is still essential that there is a certain level of trust between the members of this cooperative overlay and approaches that can prevent malicious use and free-riders. Thus, approaches taking this need into account are marked as (●) in the Social column, otherwise marked as (✗).

Economical Challenges

Solely relying on voluntary contributions within the collaborative environment creates a favorable environment for free-riding (consuming resources without contributing). Hence, incentives among the participating members need to be provided. The problem of fixing an incentive chain to counter DDoS attacks was discussed in 2007 [91], in which the authors sought cost models and possible technological solutions to enable an incentive chain. Costs are typically defined in the form of CAPital EXpenditures (CAPEX) to configure and maintain the communication infrastructure and OPErating EXpenditures (OPEX) to cover resource utilization costs for the actual attack mitigation. Among the alternatives, the authors concluded that usage-based pricing based on the capacity provision enabling victims to disseminate enough incentive along attack paths. None of the works (✗) listed in Table 3.4 proposed a technical solution to encourage the participation of other organizations in the dissemination of information about attacks and their mitigation.

Legal Challenges

It is essential to point out that these challenges are interdependent, *i.e.*, to provide incentives or map contributions, and possible abuses in the collaborative environment, it is necessary to have support in the technical environment. Concerning legal aspects, this interdependence also exists being necessary to consider 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. Based on the premise of most of the works that the participating members of the alliance are trusted, such inter-dependence is highlighted on the condition that most of the works partially (◐) address legal aspects. The major challenge in the legal dimensions is not precisely from a technical nature, although relying on it. In many cases, legal issues rely on government legislation and regulations of the organization itself. Such challenges can become the biggest impediment to sharing information, with the general concern about leaking information that can harm both users (mostly IP addresses) and the company itself (information about service availability).

3.6 Analysis of Cooperative Defenses Characteristics and Challenges

This Section it is structured in three subsections in order to (subsection 3.6.1) analyze the development of cooperative defenses over time, their characteristics (subsection 3.6.2), and how those cooperative defenses cope with the aforementioned challenges (subsection 3.6.3).

3.6.1 Timeline of Cooperative Defenses

Figure 3.2 shows the distribution of works described in this chapter by year. This distribution is essential to analyze the evolution of the proposals in this area and evaluate which sets of factors can motivate or discourage new proposals. It is also noteworthy that not all existing works were described but the most relevant ones considering their citations' number.

As depicted in Figure 3.2, the years of 2000 to 2005 denote the initial period of research by academia and industry in the area of cooperative defenses to combat DDoS attacks. In these years, proposals for mechanisms focused on countering or detecting attacks, not necessarily using a hybrid-mode. Instead, a predominance of source, network, and destination mechanisms is observed as tools that can operate independently (*i.e.*, according to their cooperation degree).

From 2006 to 2020, hybrid mechanisms based on overlays networks predominate to exchange information between different organizations. Many of these hybrid mechanisms are based on approaches proposed in the early periods, such as IP traceback [258, 263], to mitigate attacks close to their origin networks. During this period, technologies based on the virtualization of resources, such as Cloud Computing (2006), emerged. Cloud computing was a milestone for service providers to maximize the efficiency of using their hardware infrastructure, typically allocated to specific applications in grids/clusters. In this sense, there are protection service providers like CloudFlare [43] and Akamai [3] that use their infrastructure as a proxy to alleviate the victim in the case of DDoS attacks, by using their large pool of resources.

Other networking concepts such as Software Defined Networks (SDN) and Virtual Network Function (VNF) emerged, driven by softwarization generated by cloud computing. This denotes the boom in related works seen in Figure 3.2 during 2015 and 2020, in which most are based on SDN such as Bohatei [69] Sahay *et al.* [208], and CoFence [195] - based on VNF. SDN is an excellent platform for building collaborative defense mechanisms as it provides a global view of the network,

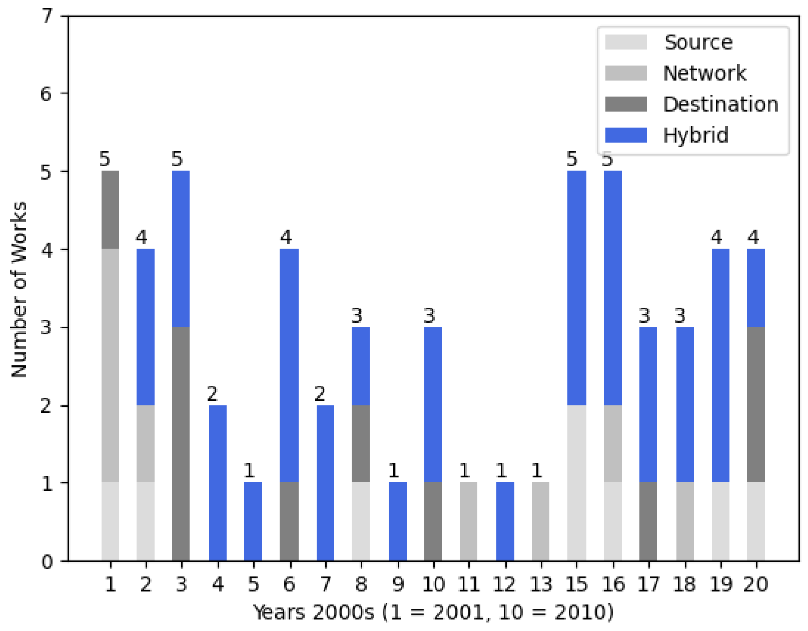


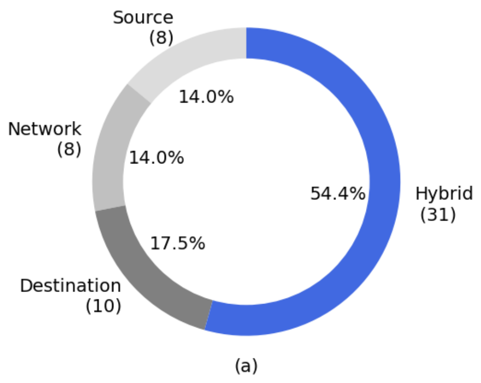
Figure 3.2: Distribution of Cooperative DDoS Defense Works per Year and Classification of Mechanism (*i.e.*, Source, Network, Destination, and Hybrid)

facilitating its management. Also, by relying purely on software, the implementation of detection algorithms for various types of attacks ease the propagation of information about attack patterns.

The period from 2017 to 2020 denotes the combination of SDN-based and Blockchain-based approaches, such as [200] and others based on it [77, 180]. While SDN provides greater efficiency in detecting and mitigating attacks, Blockchain is an excellent platform for signaling these attacks and providing incentives. In addition, in mid-2016 an IETF approach emerged, DOTS [156, 167], whose architecture and standardization process is still in progress. While the IETF aims to solve technical compatibility by standardizing the DOTS protocol between manufacturers and major Internet players, there are challenges of economic dimensions that are not addressed in the protocol's proposal.

3.6.2 Analysis of Cooperative Defense Characteristics

As the number of DDoS attacks increases, there is also an increase in the number of proposals to counter these attacks based on different approaches. The literature presents different ways to categorize DDoS defenses [151, 183, 265], which were the basis to categorize the works presented in this Chapter. Figure 3.3 shows the percentage of distribution of the different characteristics in the analyzed works. In particular, Figure 3.3 (a) details the percentage of distribution of the types of



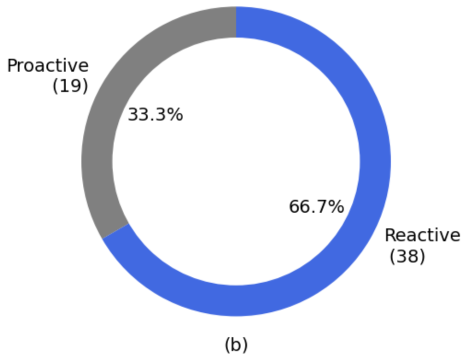


Figure 3.3: Percentage of (a) Work per Classification and (b) Activity Type

mechanisms. Hybrid mechanisms represent 54.4% of the total, while the minority is divided between destination (17.5%), network (14%), and source (14%) mechanisms. However, it is observed that most hybrid tools are based on tools and approaches from the destination, network, and source-based mechanisms. For instance, IP traceback to push mitigation actions toward the attack source and statistical packet marking for signaling suspect flows. Therefore, source, network, and destination-based mechanisms often serve as platforms, in which hybrid mechanisms are implemented across different organizations and infrastructures.

The activity type (*cf.*, Figure 3.3 - b), concerns the regards the mode of operation of the mechanism. While proactive (33.3%) mechanisms are typically targeted toward the detection of DDoS attacks at multiple points across a collaborative alliance, reactive (66.7%) mechanisms (*i.e.*, capability-based), react to the attack enforcing actions such as black-holing traffic or blocking blacklisted IP addresses. Since reactive mechanisms typically rely on proactive detection mechanisms (*i.e.*, there is a combination of proactive and reactive mostly observed in hybrid mechanisms), the proportion 3/4 shown in Figure 3.3 (b) is aligned with reality. However, there may be independent attack detection points deployed across a collaborative defense continually looking for potential attackers based on in-line tools, such as firewalls and IDS/IPS systems (*e.g.*, Snort [33] and Bro [181]). Also, collaborative detection mechanisms are not always proactive in contrast to capability-based mechanisms that enforce action on detected attacks are always reactive as they depend on confirmation of the attack. For instance, a detection mechanism may leverage flow data exported from the edge routers and switches,

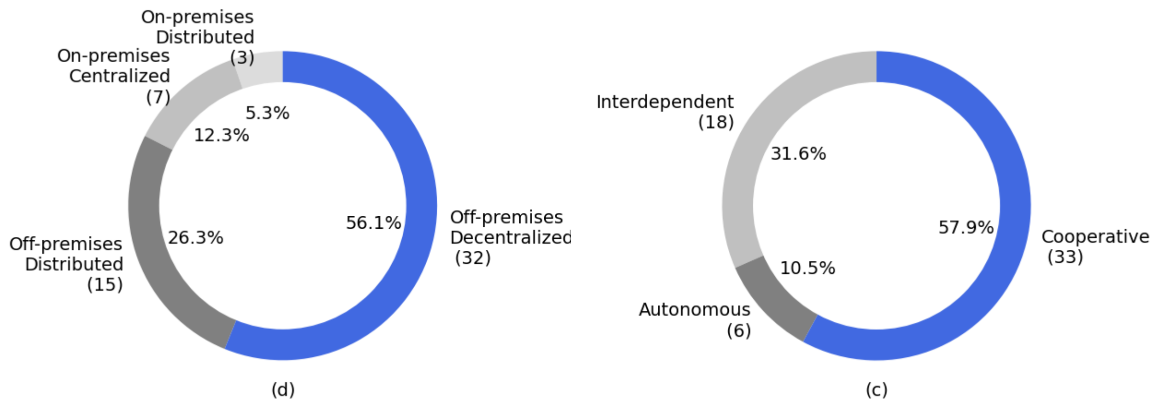


Figure 3.4: Percentage of (c) Cooperative Degree and (d) Architectural Type

and perform meta-data analysis to detect anomalies. Then, possible anomalies can be contrasted with a similar analysis at a segment where the traffic flow traverses to confirm an eventual DDoS attack.

Figure 3.4 (c) and (d) shows, respectively, the cooperative degree and architectural type. These figures reinforce the correlation between the type of architecture and the degree of cooperation, which is related to the classification of the mechanism. For instance, off-premises decentralized architectures (56.1%) are directly related to the cooperative mode (57.9%). However, interdependent (31.6%) and autonomous (10.5%) cooperation degrees do not directly correlate with the architectural type, often being organized in distributed off-premises mechanisms operating independently.

3.6.3 Cooperative Defense Challenges for Hybrid DDoS Mechanisms

Figure 3.5 extends Table 3.4 presenting details on the percentage in which the hybrid mechanisms fulfill the challenges. This emphasizes the need for hybrid mechanisms able to provide technical responses in these dimensions. Technical challenge is the main one faced by these mechanisms considering since it reflects on the system design which is the platform where social, economical, and legal challenges are implemented.

The technical dimension depicted in Figure 3.5 encompasses this vision with all mechanisms providing solutions with different approaches and characteristics. The distinction between Check (51.6%) and Partial (48.4%) implies how to implement these mechanisms, imposing a greater or lesser need for hardware changes on the peers involved in the collaborative defense. An ideal solution should avoid extra hardware or software requirements on the underlying network infrastructure, which can

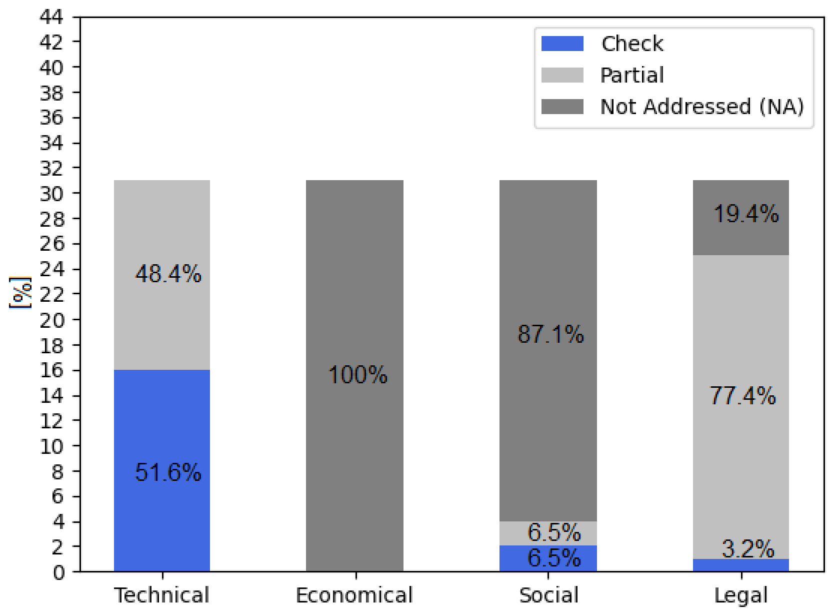


Figure 3.5: Percentage of Addressed Challenges by Hybrid DDoS Mechanisms

be achieved either by using a novel technology (*e.g.*, , SDN, and NFV) or a novel architecture, such as IETF DOTS [156] and DefCOM [169].

However, challenges in dimensions such as economic and social, are not adequately addressed. In case of economic challenges (*i.e.*, whether there is an approach to provide incentives that can cover detection costs or collaborative mitigation), 100% of the mechanisms do not address the challenge. In this sense, there are mentions [91, 265] of the need to build a chain of incentives to foster the adoption and use of collaborative defenses, but there is no mechanism in itself. Also, there is the challenge of possible peers who can use services without contributing, even abusing these services if there is no compensation for this use (*i.e.*, free-riding [81]).

Another challenge of fundamental importance is the social one, in which most approaches consider as a premise that all participants are trusted. Thus, security details such as confidentiality and integrity of information exchanged in the overlay are not addressed by most works (87.1%). Although the premise that members participating in the cooperative alliance are true, it is necessary to ensure that the origin of the information can be verified at each interaction, avoiding malicious acts that may generate competitive advantages in the market (in competing organizations). Henceforth, the addition of reputation mechanisms is a reliable answer widely used in collaborative P2P systems, which is tackled 6.5% of the mechanisms (*e.g.*, Giotis *et al.* [77, 78] and partially in 6.5% (*e.g.*, AITF [10] and TMH [262]).

Concerning the legal aspect, most mechanisms deal partially (77.4%) based on the premise of participation by trusted members. However, even among trusted members, it is necessary to understand and react upon the differences in each region or country's legal aspects, which can influence the cooperation among members. For example, for legal reasons, a member may be prevented from blocking a suspected host's traffic. Thus, a technical solution based on BC should avoid additional costs regarding hardware and software and be simple to deploy and operate. However, BloSS encompasses the support for incentives based on BC that can be safely and reliably distributed among participants and that legal/conformity options can be selected to, for example, restrict operation to specific regions/countries or members.

Therefore, cooperative defenses can benefits from BC in different dimensions. While BC can *(i)* reduce the complexity of operation and coordination by using existing infrastructure to distribute rules without specialized registries or protocols, it also can foster a *(ii)* trusted cooperation due to its transparency and decentralized characteristics. Also, it can provide *(iii)* financial incentives which foster the cooperative behavior among service providers [198]. Thus, BC capabilities can be leveraged for signaling mitigation requests across a BC network in a similar approach than DefCOM [169] and serve as a stable platform for the exchange of mitigation services, where participants express their needs in forms of incentives.

3.7 Key Observations

This Chapter provided an analysis of the state-of-the-art concerning cooperative defenses. The analysis was performed during the thesis's development, which is fundamental to identify critical points that prevent widespread adoption of collaborative defenses, such as listed in the challenges mentioned above. In specific, the objective was to analyze how the different mechanisms provide an answer to counter DDoS attacks and analyze how these mechanisms tackle challenges in different dimensions. While all mechanisms provide technical solutions for signaling or collaboratively mitigating attacks, an interesting question is how these mechanisms are implemented *i.e.*, referring to the first dimension of analysis concerning technical challenges. In this sense, approaches that provide less overhead in the underlying infrastructure are more desirable, considering that the Internet is composed of diverse subsystems and heterogeneous peers.

The underlying infrastructure of the Internet is highly heterogeneous. Thus, cooperative defense solutions imposing hardware requirements also impose a limitation on its deployment and operation. As discussed in subsection 3.6.3, there are different approaches to leverage the adoption of a given solution across the Internet. While the first involves standardizing a communication protocol and corresponding architecture, the second involves proposing solutions based on software capable of running on standard servers. The most prominent example of a standardization attempt case is the IETF DOTS protocol [156], which seeks to define a protocol and architecture involving major telecommunications players and content providers, for a cooperative solution operating based on a distributed client-server model. Therefore, the proposed solution does indeed introduce hardware requirements to support the proposed protocol, but since the change is a consensus among influential organizations, it may indicate a cooperative solution's widespread adoption. Solutions of the second category are based on recent software-based technologies, whether in the part of network management with Software-Defined Network (SDN) or in the Virtualization of Network Functions (VNF) that allow greater flexibility in the network or execution of rules requested by a third party without imposing restrictions on hardware. In this sense, the state-of-the-art assessment covered in Section 3.5 was relevant to confirm technical challenges for widespread deployment pointed as indicated in previous surveys [183, 265].

The heterogeneity issue is related to a challenge in the social dimension reflected in the decentralized nature of different organizations, their trust premises, and the competition among organizations in the same commercial segment (*e.g.*, content providers). These and other factors indicated in this Chapter (*cf.*, subsection 3.6.3) make the use of Trusted-Third Parties (TTP) difficult in the context of collaborative defenses. Also, even if there is a platform of relatively easy implantation and operation and potential trusted relations among its participants, without an incentive scheme, the collective defense would become difficult. In this sense, there is an implication in operational costs to mitigate large-scale attacks involving specialized equipment and personnel.

Therefore, cooperative defenses can benefit from BC in different dimensions since there are no major hardware requirements in a permissioned setting. There is a possibility of increasing trust levels through the transparency of actions and adding incentives in exchange for mitigation services. However, as indicated in previous surveys [183, 265] and described in detail in this Chapter, there is a need for technical solutions to address all the challenges presented in different dimensions. Economic, social, and legal challenges directly impact trust in the cooperative system, which combined, exert fundamental relevance in the influence of decision-making processes (*e.g.*, whether to adopt a given system or cooperate), and establishment and management (*e.g.*, reputation systems) of trust in the cooperative defense context.

4 Design of the Cooperative Signaling Protocol

This Chapter discusses the design details of the cooperative protocol, which governs a technical solution to the challenges listed in the technological, social, economic, and legal environment. Hence, the protocol is deployed in a BC-based environment whereas allied members can request and offer mitigation services between pairs of Targets (T) and Mitigators (M). Considering the need to minimize the impact on the underlying networking infrastructure and maximize the applicability of the cooperative solution, an entire software-based approach is used for both the deployment of the on-chain protocol and the Decentralized Application (dApp) operating on the peers.

The Blockchain Signaling System (BloSS) is structured in two parts:

- **On-chain**: include the processes of integrated payment and reputation ranking, being based on a sequence of defined steps mapped as states whereas each step's outcome is transparent and verifiable. Confidential information pertaining to collaboration between pairs is sent off-chain.

- **Off-chain**: includes the dApp with interface to the network management system and the deployed on-chain protocol. BloSS dApp stores individual settings related to when and how to request or accept mitigation services including legal aspects, fully detailed in Chapter 5.

4.1 Design Considerations

Based on the discussion of the challenges listed in the state-of-the-art (*cf.*, Chapter 3), structured in the technical, economic, social, and legal dimensions, this chapter presents the rational behind the design. In this way, the design seeks to reflect on a system bringing together the following characteristics in order to amplify the deployment and operation of collaborative defenses:

- **Technical**: provide software-based approach enabling the exchange of information in the temporal scale of seconds to minutes.
- **Economical**: enable the exchange of incentives for mitigation services, in order to cover operational expenses.
- **Social**: interactions should be private and transparent to members within an alliance.
- **Legal**: members should select which members to interact based on internal organizational and legal issues.

Henceforth, considering the need to minimize the impact on the underlying networking infrastructure and maximize the applicability of the cooperative solution, an entire software-based approach is used. Protocol' contracts are defined on-chain between pairs of Targets (T) and Mitigators (M) with a defined and verifiable sequence of steps within a permissioned network. Details the on-chain cooperative signaling protocol including processes of integrated payment and reputation ranking, and concerns the Blockchain Signaling System (BloSS) dApp based on SDN to facilitate with the network management system signaling attacks and the implementation of mitigation actions. 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 [253] 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 [14]. Thus, a BC-based solution can provide through its natural transparency a higher degree of transparency and trust between collaborative instances.

Table 4.1: 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
Security	Enhancement of trust through full transparency & decentralization	Excess of transparency may impair confidentiality

4.1.1 Performance — Block Size and Delay

A limiting factor in a BC-based cooperative defense concerns the deployment and operation of a platform, referring to a technical usability aspect. Being widely tested platforms, resilient to DDoS and providing immutability and transparency are advantages of the BC that impact user confidence in a positive way. However, it is worth noting the drawbacks regarding latency performance (*e.g.,* time for block propagation), which even in a permissible deployment based on a consensus mechanism such as PoA, present a performance degradation due to total replication of the data. As certain nodes require a higher level of trust in a PoA network [55] and trust is already given in a consortium-based BC; a trustless PoW-based consensus would only lead to excessive computational effort. Therefore, depending on the governance model established for the alliance, two or more nodes (even all nodes) can be defined as authorities or super nodes, entitled to collect transactions, build blocks, and propagate them across the network.

In addition, a limiting factor in performance (throughput) is the relation between the block size and the propagation delay, *i.e.,* latency. The BC throughput is defined as [52]:

$$TP = \frac{blockSize}{propDelay} \tag{4.1}$$

Considering a *blockSize* of 1 MB and a *propDelay* of 10s, 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.* Figure 4.1).

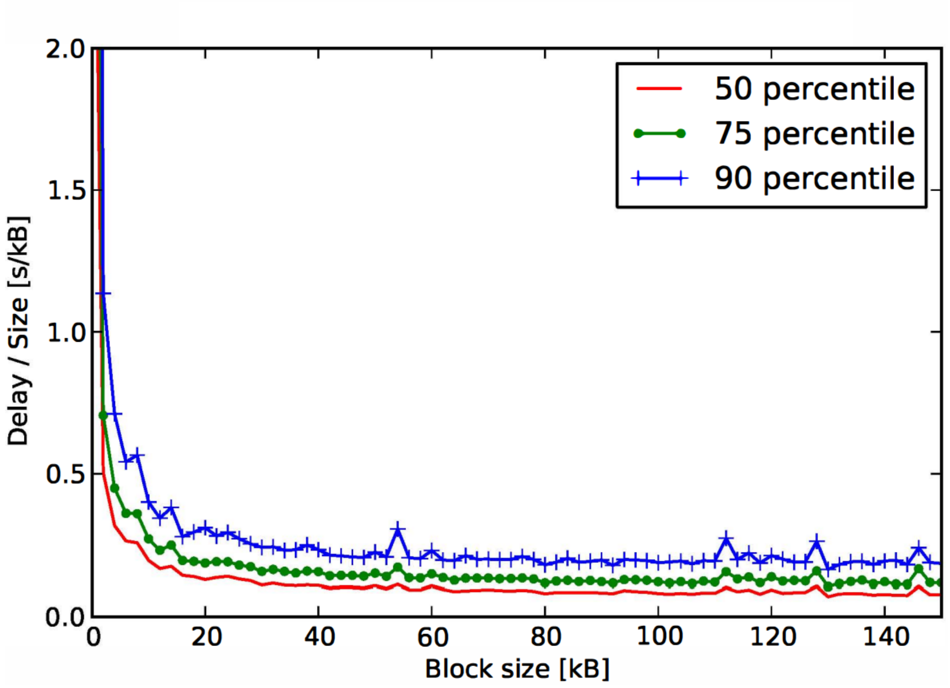


Figure 4.1: Influence of Block Size on Propagation Delay [56]

The effects of an increasing block size on the propagation delay is shown in Figure 4.1. It has been shown a strong correlation between the size and its influence on 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 [56]. 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 5,760 blocks. The maximum amount of information that can be Storage Available in the Period (SAP) is described by:

$$SAP = \frac{T}{blockGenTime} \times blockSize \tag{4.2}$$

In the case of a *blockSize* of 1 MB, the SAP considering a *blockGenTime* of 15 seconds 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 [268]. However, such an approach does not reduce by itself the need for storage, requiring snapshot compression and storage at a more efficient media.

4.1.2 Financial Incentives

The CSIRT community currently shares information based on trusted contacts [226], *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 [189]. There are arguments both for and against the use of financial incentives [102, 226]:

- **Against**: changing the current model of how information is shared could impact the trust model among CSIRTs.
- **In 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" [226] 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.

Considering that BC is a transparent platform, in which cooperative entities can define its terms of cooperation (*i.e.*, , incentives and conditions to perform a mitigation service) in its smart contracts, the negotiation of a mitigation service takes place in a transparent manner. Also, with the history of past negotiations open to all members within the alliance, organizations are able to evaluate each participant's mitigation or requester history as well as their terms for performing the service.

4.1.3 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 [88]. 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 [87], 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 [191, 246], 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 [110, 135], 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 [87]. 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.

However, considering security (confidentiality) requirements as a collaborative defense, a deployment private permissionless needs to be considered, *i.e.*, closed to a community. In this sense, information disclosed in this cooperation network should not be disclosed to the general public, but kept within the cooperative network. It is important, within this alliance, that all members participate in the BC consensus on an equal basis, preventing certain members from *e.g.*, having the ability to censor certain transactions.

Finally, BC allows to build a reliable and robust platform for signaling DDoS threats in a transparent and verifiable manner, but it does not cover all the security needs of such platform. For example, while transparency favors a trust-free platform, it is needed to strike a balance with confidentiality requirements of each member in order to securely exchange information.

4.2 Cooperative Signaling Protocol

Figure 4.2 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 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

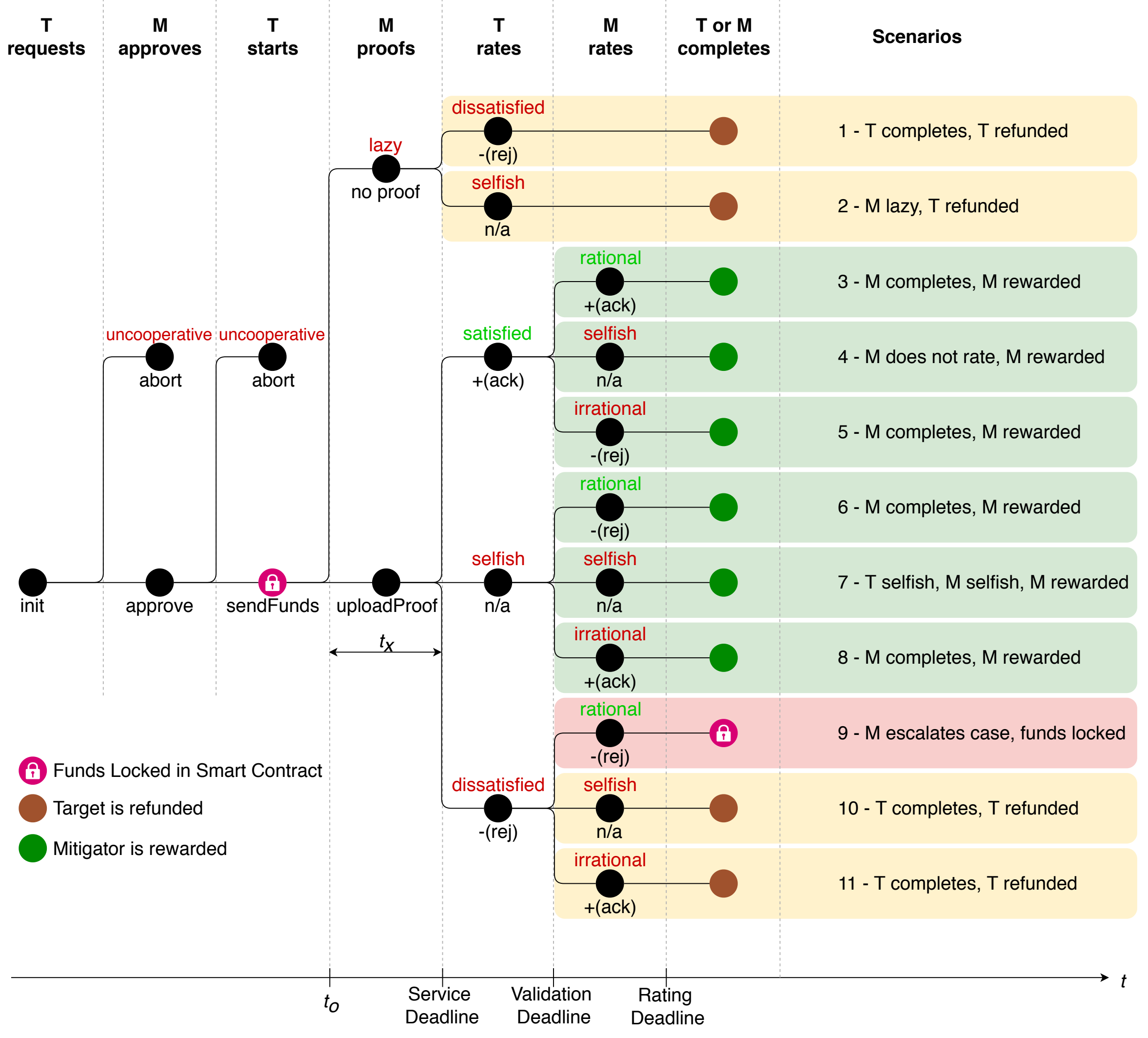


Figure 4.2: On-chain Cooperative Signaling Protocol

SC until completion. *T* writes the attack information (*e.g.*, IP address ranges, packet captures, request headers, notes, and other patterns [197]) to the external off-chain storage (*e.g.*, IPFS [17]). During the service time window (to), *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 Figure 4.3. For example, if *T* misses to acknowledge or reject the proof in response to the delivered service, *M* will be rewarded.

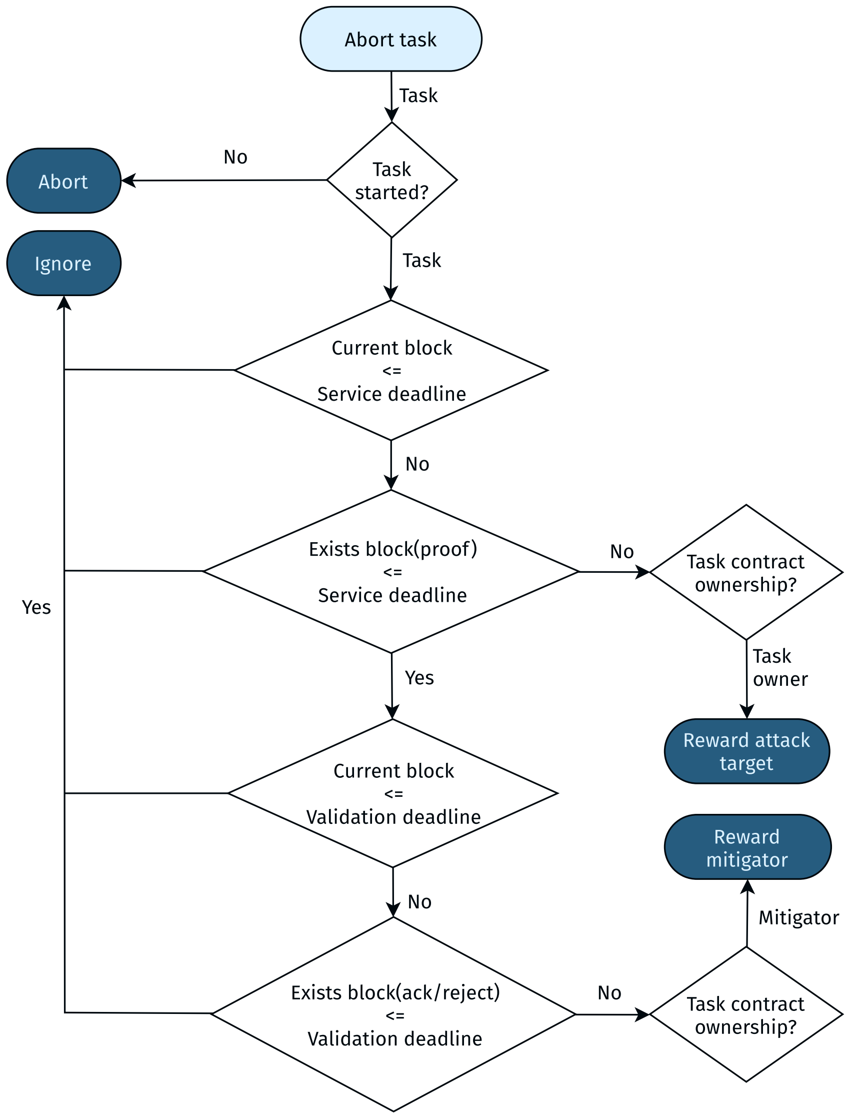


Figure 4.3: Payout Process: T is Refunded, if no Proof was Uploaded and M is Rewarded, if T Does Not Validate in Time

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 Figure 4.3. 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 [30]. 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 [30]. 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.

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.

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*. By storing 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.

4.2.1 Protocol Outcome 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.

4.2.2 Truthfulness of Mitigation Proofs

A crucial element within the cooperative protocol is the ability of an *M* to provide a proof-of-mitigation that satisfies the aspects of reproducibility, tamper-evidence and timeliness. If such a proof is not available right after completion of the mitigation, the other party will withhold the incentive payout and the overall service becomes unusable. Manual verification of a mitigation proof is not feasible, due to the strict time constraints required to provide a mitigation service able to counteract large-scale DDoS attacks. The timeliness aspect does, therefore, include the aspect of being able to automatically verify the proof during the available time-window and excludes any user interaction in order to be efficient.

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 to complete the service 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 [141] and this subsection presents an overview on the challenge of verifying the quality of the mitigation service.

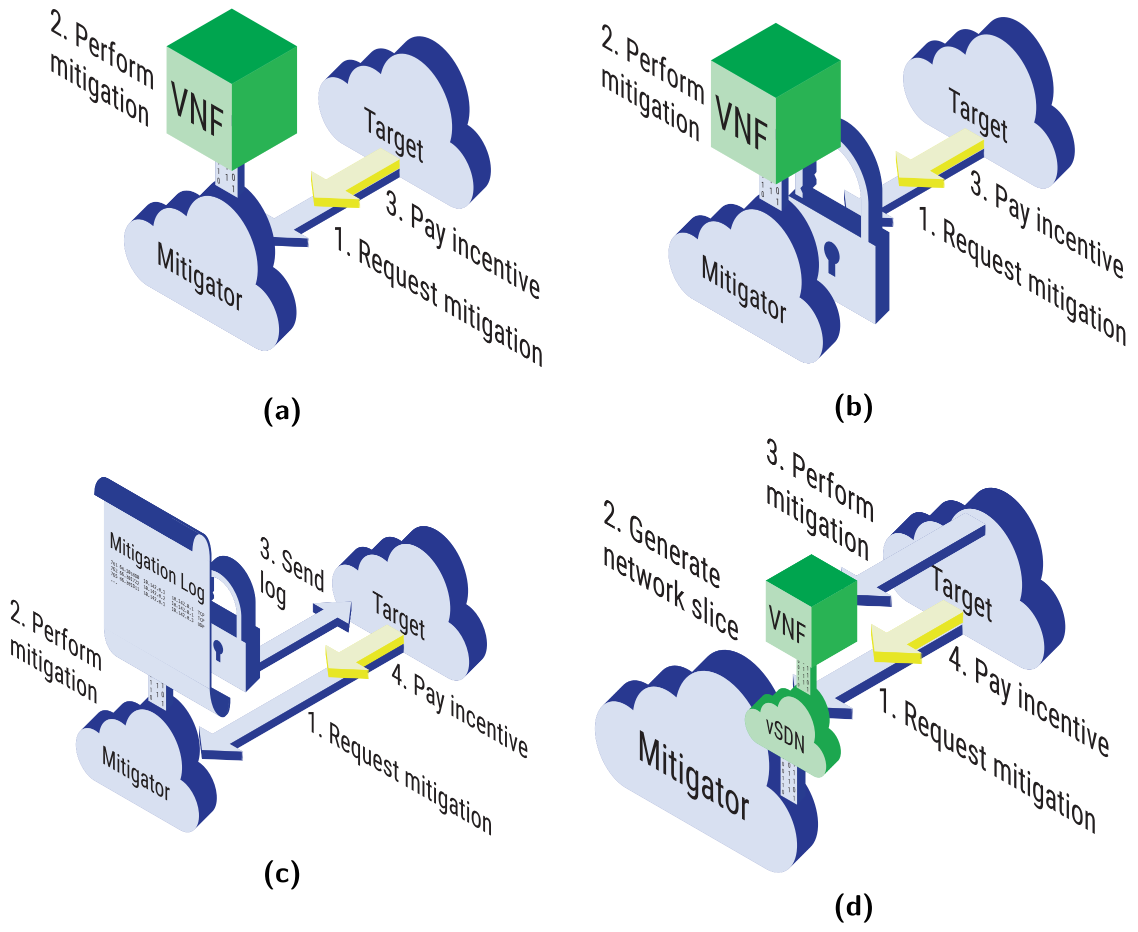


Figure 4.4: Approaches Toward a Verifiable Mitigation Proof [141]: (a) VNF Marketplace, (b) Trusted Platform, (c) Secure Logging, and (d) Network Slicing

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 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. Incombination 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

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.

Truthfulness of Mitigation Proofs

For a qualitative discussion of the individual approaches, metrics are based on the scheme proposed by Zargar *et al.* [265] and focus on the deployment complexity as well as scalability while adding security related metrics not present in [265]. These additional metrics can provide an important overview of the presented approaches since providing a successful proof of mitigation is largely dependent on the security of the system generating the proof as well as the proof itself.

1. **Confidentiality**: Describes how effective rules and measures are to protect both the mitigation proof as well as the mitigation system from unauthorized access.

2. **Integrity**: Defines the level of trustworthiness of the proof generated by the given approach in regards to accuracy and tamper-resistance.

3. **Availability**: Relates to the availability of the mitigation proof to authorized parties.

4. **Reproducibility**: Describes the ability to reproduce the proof through replay by a third party.

5. **Tamper-Evidence**: Discloses the effort necessary for changing the proof of mitigation to reflect an alternate reality.

6. **Timeliness**: Defines the ability to provide an automatically verifiable proof within a pre-specified time-frame while adhering to all security requirements.

7. **Deployment Complexity**: Defines the additional resources required to deploy the approach.

8. **Scalability**: Relates to the adaptability of the approach in regards to a large-scale DDoS defense scenario with high-bandwidth attacks.

9. **Service Model**: Defines the cloud service model the approach most closely relates to.

Table 4.2: Qualitative Comparison of Approaches Toward Mitigation-as-a-Service

	NFV	Trusted Computing	Secure Logging	Network Slicing
Security				
1. Confidentiality	Low	Medium	Low	Low
2. Integrity	Low	High	High	Low
3. Availability	High	Medium	High	Medium
4. Reproducibility	High	High	Low	High
5. Tamper-Evidence	Low	Medium	Medium	Low
6. Timeliness	High	High	Low	Medium
Practicability				
7. Deployment Complexity	Low	High	High	High
8. Scalability	High	Low	Low	Low
Scope				
9. Service Model	SaaS	SaaS	PaaS	IaaS

Table 4.2 shows that no single approach satisfactorily addresses the trade-offs between security and practicability and could, therefore, be used by itself for an independent trustless mitigation service. NFV and Network Slicing have similar characteristics with respect to security due to their virtualized

nature. While NFV virtualizes a single function in the network, Network Slicing aims to deliver a portion of the network infrastructure as a service. To accomplish this, approaches like AutoSlice [23] help to automate the creation of on-demand slices per mitigation request. However, the infrastructure requirements such as the need for SDN based networking for the automatic creation of vSDNs, increases the associated deployment complexity, thus limiting the applicability of this approach.

An NFV-based approach on the other hand, presents a lower deployment complexity. For example, CoFence [195] can create VNFs upon demand to filter attack traffic, however, security aspects of this approach can be easily tampered with to obtain the incentive related to the mitigation service. The Trusted Computing and Secure Logging approaches have high deployment complexity as strict hardware requirements need to be considered. Logging is by default provided by any mitigation tool and therefore has no deployment complexity, but secure logging requires a trusted platform to ensure that the output of a mitigation action has not been tampered with. These deployment complexities directly translate to poor scalability since a large number of TPM as well as Intel SGX [49] enabled systems would be required to use a trusted computing approach at scale.

The service model metric differentiates individual approaches by correlating them with their respective cloud service models. Table 4.2 presents this metric showing NFV and Trusted Computing approaches follow a similar model as Software-as-a-Service (SaaS) since these are individual software packages that provide the proof. In contrast, secure logging only provides logs identical to Platform-as-a-Service (PaaS) cloud models where an interface to a service is provided. The approach with the highest degree of access to the mitigation system is the network slicing approach where, similar to Infrastructure-as-a-Service (IaaS), a complete virtualized networking infrastructure is provided. Most discussed approaches are not clearly separable since they combine similar concepts. However, these combinations lead to an increase in complexity and lack of scalability, which may appear that combining these approaches could lead to a comprehensive solution that addresses all requirements.

4.3 Reputation Tracking

A minimal level of trust needs to be established in a competitive environment. Solely relying on a voluntary contribution (*i.e.*, accepting defense requests) creates a favorable environment for free riding peers (consuming resources without contributing). Hence, a reputation scheme allows contributors and consumers of the network to rate entities that request protection in a cooperative defense. 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 4.3 illustrates three possible mitigation histories and outcomes, depending on how T and M act.

Table 4.3: 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	-

Whenever a M submits proof of mitigation, there is no automated way for other peers to build a consensus on the quality of the service delivered [141]. 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. 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.

It is reasonable to assume that the costs of T inflicted by the attack are higher than the costs of the 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 because of future attacks. A reputation and incentive scheme can incentivize 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 [251].

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, a 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 peer's reputation. There are two types of reputable sources, subjective and objective. While the individual 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 the compute of 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 number of completed tasks over time helps to compare historically with current customer performance.

4.3.1 Target and Mitigator Profiles

The behavior of the different customers (T or M) is summarized in Tables 4.4 and 4.5. These profiles were selected to build a reputation system that reveals freeloaders and false-reporters. Revealing and expelling lazy customers can reduce free-riding. Similarly, exposing untruthful customers could potentially reduce false-reporting. However, this is more difficult than detecting free-riding, because such a fraudulent case needs close examination by a tribunal.

Table 4.4: Attack target (T) strategies: The satisfied T Acknowledges (ack) and Accepts the Service if a Proof was Uploaded. The Dissatisfied T Always Rejects (rej) the Service

Strategy	**Description**	**Strategy**		
		Abort	**Start (Payment)**	**Rate**
Uncooperative	Aborts before payment	✓		
Selfish	Pays but does not leave feedback		✓	
Satisfied	Pays and gives positive feedback if a proof was uploaded		✓	+ (ack)
Dissatisfied	Pays, always gives negative feedback		✓	− (ack)

Table 4.5: Mitigator (M) Strategies

Strategy	Description	Strategy			
		Abort	Approve	Proof	Rate
Uncooperative	Aborts before approval	✓			
Selfish	Delivers no service (no rating allowed)		✓		
Satisfied	Signs contract and uploads proof but never rates		✓	✓	
Dissatisfied	Signs contract, uploads proof and rates accordingly to T's expectations		✓	✓	+/-

4.3.2 Beta Reputation Scores

An analysis of reputation scores based on different profiles for T and M based on a probabilistic reputation engine presented by Jøsang *et al.* in [101] and Schlosser *et al.* in [218]. A Beta density function is typically used to represent the distribution of binary events (α and β) with restriction variables that define their thresholds:

$$f_X(x : \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1} \tag{4.3}$$

Where:

- Probability (p) = $0 \leq p \leq 1$
- α and β = $\alpha \geq 0$ and $\beta \geq 0$

In the case of the cooperative signaling protocol, α and β denote, depending on the final state of the protocol (*i.e.*, outcome of interactions), binary values representing that a target or mitigator is satisfied (positive) or not satisfied (negative). Then, to evaluate the probability for a customer c calculated as the expected value:

$$E(p) = \frac{\alpha}{\alpha+\beta} \tag{4.4}$$

Where:

- α = positive(c) + 1 and
- β = negative(c) + 1

The Beta function outputs reputation scores in a range of $[0, 1]$, where 0.5 is the initial, neutral score of every new customer. To analyze the reputation scores, it was defined a threshold in which no customer (*i.e.*, T or M) is willing to work with a counter-party that has a Beta reputation lower than 0.3—accumulated positive and negative reputation values at one point in time. This value reflects the probability of future positive interaction with customer c, based on its past positive and negative ratings. Reputation points used to rate target and mitigator are preferably not the same because a task owner with a good rating does not necessarily need to be a good mitigator and vice-versa. Therefore, the reputation earned as a mitigator is stored separately from the reputation earned as an attack target or task owner. A simple metric gives the analyzing peers a clear understanding of a peer's reputation. Further, a complex metric might lead to negative feedback loops [68]. The scores earned by ratings from other peers (subjective ratings) and the total amount of transactions (objective metrics) are equally essential to have a good understanding of a peer's trustworthiness [256].

Since reputation is earned in interactions between peers *i.e.*, T and M in BloSS, it can be attached to transactions. BC is useful in building consensus about the reputation of all peers in the system, providing an accurate and verifiable scoring engine. Repeated interactions and a definite interaction history allow for building a reliable foundation of trust within the cooperative defense. Furthermore, with incentives and penalties, users are more likely to behave according to the social norm. The reputation scheme can prevent free-riding (attack targets) and false-reporting (mitigators) in the long run by incentivizing rational behavior. However, reputation itself is not sufficient to discriminate selfish and rational mitigators, because their behavior in terms of service delivery is similar. Selfish customers, nonetheless, can be easily identified by looking at past interactions.

Further, the reward payment provides an incentive structure suited to discipline selfish customers. A reputation scheme is a solution towards a marketplace of mitigation services where their behavior might reward "good" mitigators. Mitigators will remember to apply the final service rating because otherwise, they deprive themselves of payment. Similarly, targets (*i.e.*, domains requesting cooperative mitigation) can also be rewarded by a reputation scheme indicating a "good" payee. The reputation system trades verifiability against customer anonymity since, for every transaction, the reputation claim and owner are published on IPFS. A verifiable but untraceable reputation system can guarantee the validity of each rating or review without explicitly linking the rating and the transaction. Ideally, it should not be possible to infer the owner of the reputation statement (*i.e.*, the rater) from the reputation statement for a particular mitigation service. Vice-versa, knowing the reputation statement's owner, the content of a particular rating should remain unknown. Nevertheless, the reputation system should only store valid ratings.

4.4 Key Observations

This Chapter presented a DDoS signaling protocol for cooperative attacks. The design considers the challenges mentioned in the technical, social, economic, and legal scopes, allowing peers to exchange information and mitigation services confidentially and with incentives that can charge operational expenses. The fact that the design is based on a software component that can be executed on standard servers presents a more significant simplification of deployment and operation, resulting in greater adoption of the cooperative solution. In this sense, members of the defense specify types of mitigation actions possible, and when requested, they can carry out these actions on-demand and at an associated cost.

Since the design is based on contracts between peers, a Mitigator *M* can freely specify an amount for the realization of a mitigation service, which could, for example, cover its operating expenses and possible profits. Similarly, a Target *T* may check whether the service offered by *M* at the given price is attractive. This can be based on an analysis of possible economic losses during periods of service unavailability (due to the DDoS attack). Further, mitigation service prices are not controlled or balanced (*i.e.*, normalized across *M*'s) in BloSS, which allows members to freely set and negotiate prices for mitigation services, as in a marketplace.

Although a pre-established level of trust is necessary to compose the members of the cooperative defense, the design of the cooperative signaling protocol incorporated a reputation system to evaluate actions of both *M* and *T*; henceforth, existing reputation algorithms are incorporated into the collaborative defense protocol to foster cooperation and build trust in the second scenario. Although being a viable approach by eliminating the need for software/hardware mechanisms to verify an attack's mitigation, its sole use does not guarantee that a malicious peer will not perform malicious actions. For instance, to subvert the reputation algorithm's functioning, such as providing negative feedback to peers who correctly execute mitigation requests or not provide feedback at all.

Lastly, the on-chain design's significant benefits are the integrity of the information exchanged between members, which can be verified transparently between alliance members, and the provision of methods for the exchange of incentives and feedback regarding the service of mitigation. These benefits are contrasted with the need to maintain the confidentiality of the information exchanged, even within the alliance. In this regard, Chapter 5 details how this aspect is addressed to ensure that details about the mitigation service are open only to those involved in the contract.

5 Design of the Blockchain Signaling System

This Chapter details the design decisions of the Blockchain Signaling System (BloSS), which is the decentralized system interfaced with the on-chain cooperative protocol presented in Chapter 4. The design considers similar assumptions to the protocol to minimize impacts and requirements on the underlying network system. Thus, BloSS-dApp aims to be deployed on generic computers, providing a web access interface for management and communication interfaces with network management systems.

BloSS decouples the primary defense system logic from the underlying network infrastructure by delegating network-specific tasks to individual modules. To further increase loose coupling between the individual parts of the defense system, data exchange related tasks apart from networking are also separated into a purpose-built module. The modularization allows for the adaptability of the entire system to different computing environments, including networking infrastructures apart from the SDN-based networking (used as a proof-of-concept). Modularizing the data exchange capabilities of the BloSS allows switching to a different data storage backends where the cooperative protocol is deployed.

5.1 Architecture

To better grasp the individual modules' tasks, a metaphor was used as a basis for the naming scheme. Figure 5.1 shows a flower, with each part of the flower representing a vital module of the BloSS. The blossom of the flower represents the core module of the BloSS, which uses other modules to mitigate an attack. 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. This includes a module for access to the Ethereum, a data store module managing data on the InterPlanetary File System (IPFS)[17] and a database module to store statistics on InfluxDB for demonstration purposes.

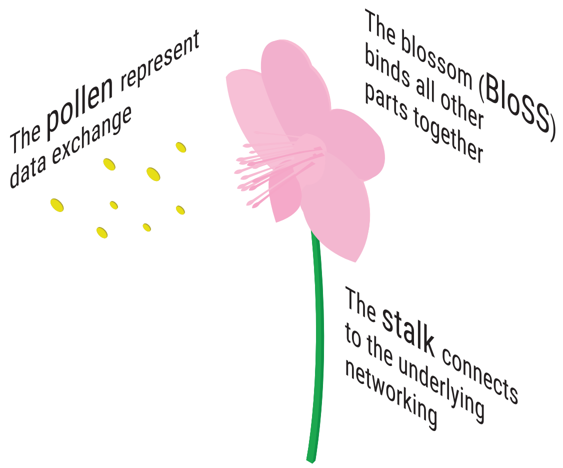

The pollen represent data exchange
The blossom (BloSS) binds all other parts together
The stalk connects to the underlying networking

Figure 5.1: Metaphor for the Naming Scheme of the Individual BloSS Modules

An overview of the BloSS Decentralized Application (dAPP) is provided in Figure 5.2 detailing connections between 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 BloSSmodule, 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

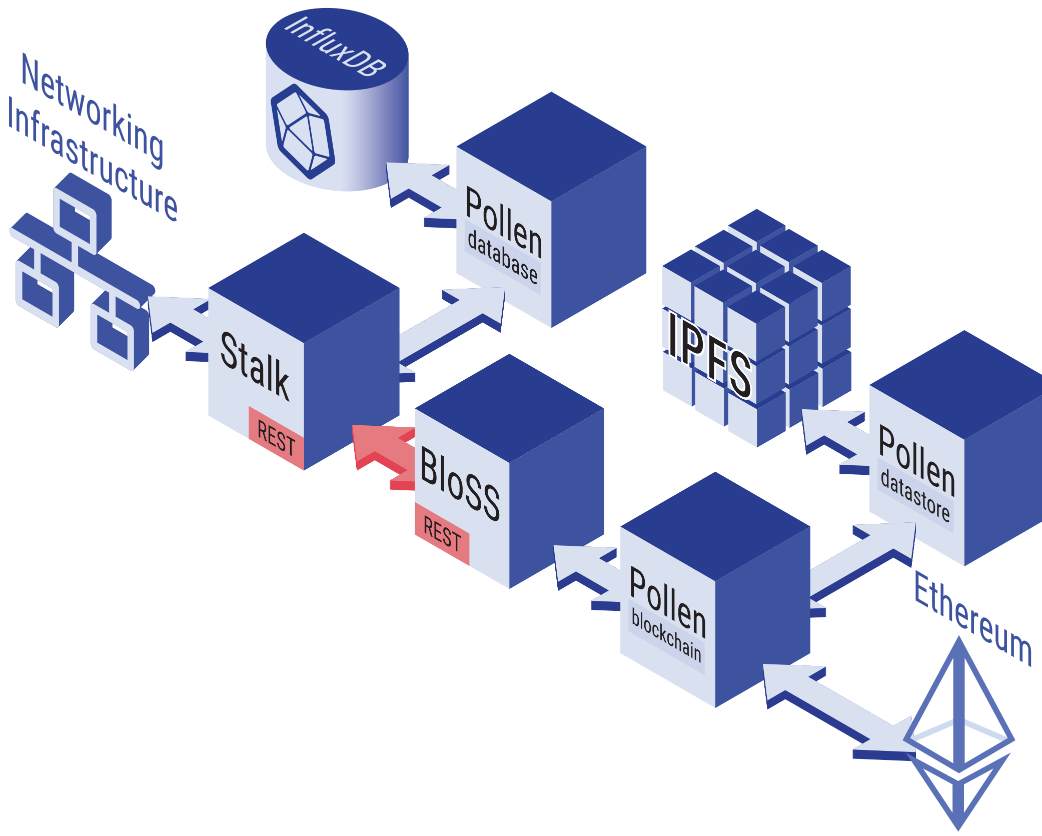


Figure 5.2: Architecture of the Blockchain Signaling System (BloSS). Ethereum Denotes the On-chain Cooperative Network

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 a decentralized and highly scalable storage solution to hold attack information. Each service provider running the BloSS also maintains an IPFS node to enable the decentralized storage. Whenever a new set of attack information is posted, 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 to implicating individuals both as victims of an ongoing DDoS 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 available to 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 block flows from these addresses to the T address specified in the attack information.

5.1.1 BloSS Module

The smallest module of the entire BloSS is the namesake module of the system: The BloSS module. The BloSS module has been decoupled from Stalk and Pollen to allow for a more network-agnostic system that would also facilitate a VNF-based encapsulation of the core parts of the BloSS, consisting of two classes as illustrated in Figure 5.3. The BloSS REST API receives attack reports from the Stalk module running on the same machine and posts these attack reports to the Pollen module. It also maintains an API mapping to receive requests from the Stalk module to set a specific attack report to "block" the to signal that the report has been addressed. Conversely, the manager periodically retrieves attack reports from the and sends the decrypted reports directly to the Stalk REST API to be blocked.

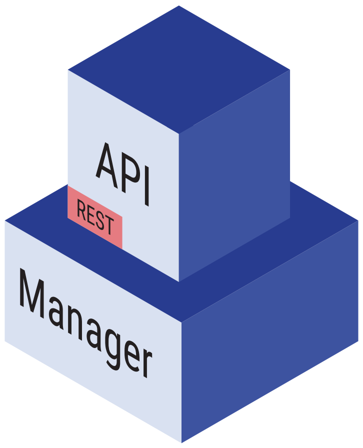


Figure 5.3: Classes of the BloSS Module

5.1.2 Pollen Module

All data-exchange apart from basic networking handled by Stalk is taken care of by the Pollen module. Pollen, therefore, consists of a multitude of specialized classes, as illustrated in Figure 5.4. These classes handle Ethereum access (PollenBlockchain), IPFS storage (PollenDatastore and PollenEncryption), and InfluxDB management (PollenDatabase). An additional helper class to handle attack reports (AttackReporting) provides simple parsing and processing capabilities to make sure attack reports are always formatted correctly and did not yet expire.

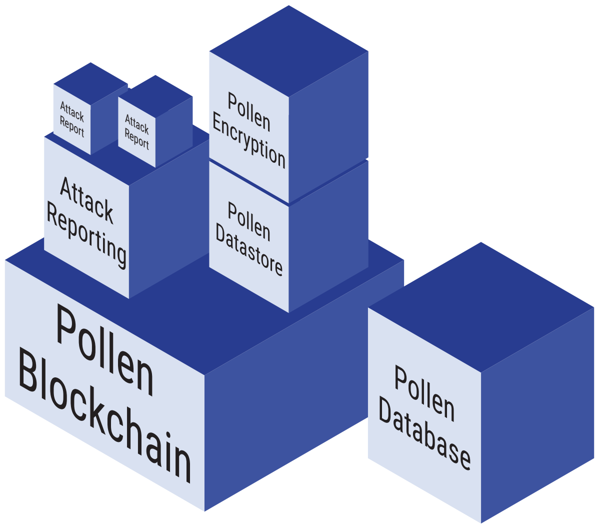


Figure 5.4: Classes of the Pollen Module

The PollenDatabase class is only used to enable the centralized storage of traffic information for visualization in Grafana [119]. It is important to note that PollenDatabase is not an integral part of the Pollen module or the BloSS as a whole and only serves as a debugging and demonstration tool since it would otherwise be complicated to gather the traffic information of multiple BloSS instances to figure out whether and were a problem exists. The centralized aspect of the InfluxDB where traffic information is stored would defeat the goal of building a decentralized and scalable DDoS defense system, which is why it is essential to state the demonstration-focused nature PollenDatabase clearly.

5.1.3 Stalk Module

The stalk module consists of two main classes, directly communicating with the underlying networking infrastructure—the controller and the simple router. While the simple router enables the correct forwarding of packets between ASes through the SDN switches, the controller analyzes flows from the same SDN switch and detects ongoing attacks. Ryu [241] is used to communicate with the SDN networking infrastructure, allowing the prototyping of SDN controllers directly in Python.

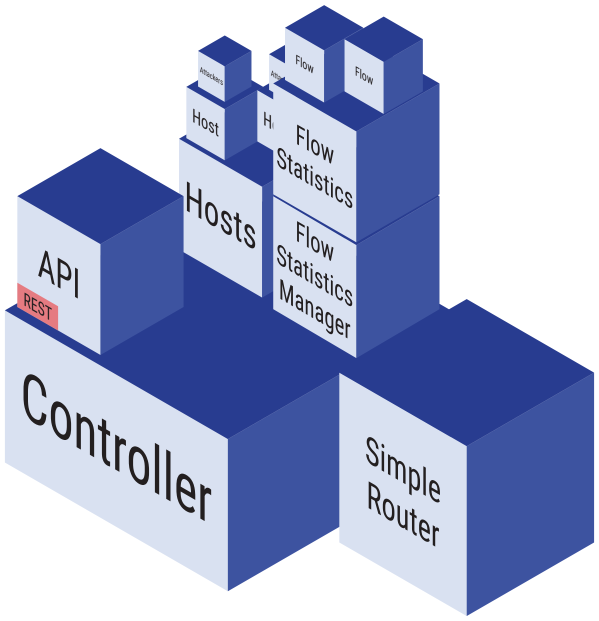


Figure 5.5: Classes of the Stalk Module

Analyzing traffic to find potential attackers is the main task of the Stalk controller. Traffic flows received from the SDN switch are stored in the flow statistics manager. From there, bytes transferred and received for each flow are written to the corresponding source or destination host. The host objects are generated while initializing the Stalk controller and are comprised of all hosts managed by the controller as specified in *config.ini*.

```
def _request_flow_statistics(self):
        ...
        parser = datapath.ofproto_parser
        request = parser.OFPFlowStatsRequest(datapath)
        datapath.send_msg(request)
```

Listing 5.1: Requesting Flow Statistics in Stalk Controller

To receive detailed traffic flow information in the first place, the Stalk controller has to send flow statistics requests continuously as shown in Listing 5.1, which will cause the SDN switch to answer with the needed data. Access to SDN-specific functions works through decorators in Ryu, which are bound to OpenFlow events such as *EventOFPFlowStatsReply* to receive flow statistics replies from the SDN switch.

Whenever traffic volumes transferred or received are written to the corresponding source or destination host, the volume of traffic is checked against a threshold. If the traffic exceeds the threshold, the remote source of the traffic is noted as being a potential attacker.

As soon as the average traffic throughput volume for a host during a specified time window exceeds a threshold, the attackers are once again updated with the current traffic volumes and whether their throughput still exceeds the threshold and the resulting list of attackers is then used to compile an attack report. This attack report is then sent through the Python requests library directly to the BloSS REST API endpoint on the same machine.

The Stalk REST API serves as the connecting link between the BloSS module and Stalk and only consists of a simple Python Flask app maintaining the */api/v1.0/mitigate* mapping to receive requests to block attackers from the BloSS module.

```python
def block_attackers(self, attack_report):
    ...
    ofproto = datapath.ofproto
    if attack_report.action == "blackhole":
        instructions = [
            parser.OFPInstructionActions(
                ofproto.OFPIT_CLEAR_ACTIONS,
                []
            )
        ]
    else:
        instructions = []
    blocking_duration = (
        self._config['INTERVAL']
                    ['MAX_BLOCKING_DURATION_SECONDS']
    )
    mod = parser.OFPFlowMod(datapath=datapath,
                            command=ofproto.OFPFC_ADD,
                            priority=999,
                            idle_timeout=blocking_duration,
                            hard_timeout=blocking_duration,
                            match=match,
                            instructions=instructions)
    datapath.send_msg(mod)
```

Listing 5.2: Blocking Attackers Based on an Attack Report

This mitigation service of receiving attack reports and blocking the contained attacker addresses is the second task of the Stalk controller. Blocking works on a per-flow basis including a source address (the attacker) and destination address (the attack target). Listing 5.2 shows the relevant Python code to block a single flow. This is accomplished by creating a new flow with a very high priority of 999 which specifies to clear all other actions specified for this flow. The old flow specifying the packet forwarding with the corresponding out-port will therefore be overruled with this new flow definition. The blocking flow is however not permanent and will only exist for a pre-defined time window in order to allow the flow to normalize and avoid blocking benign traffic.

5.1.4 PollenBlockchain Module

PollenBlockchain is capable of automatically creating the system contracts upon initialization, as illustrated in Figure 5.6. It first checks whether a system contract address is already defined in *config.ini*, and if this is not the case, it uses the solidity compiler [144] to create the solidity bytecode which is then deployed through PollenBlockchain. After deploying a system contract, it needs to be registered with the relay contract to specify which subnets the system contract is responsible for. This is accomplished with a direct transaction from PollenBlockchain to the relay contract. Ethereum-related transactions go through the Web3 Python library [147], which is connected to the RPC API of the geth Ethereum node running on the controller.

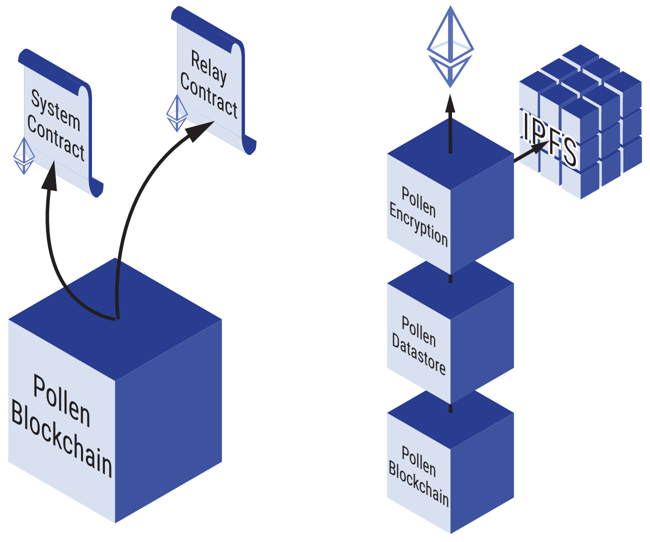


Figure 5.6: Duties of the PollenBlockchain Class. Left: Initialize System Contracts. Right: Retrieval and Storage of Attack Reports.

Access to the RPC API is limited to localhost connections by default, which is why the geth Node needs to run on the same controller as the BloSS instance. Apart from deploying contracts, the PollenBlockchain class posts the public key used to encrypt and sign attack reports to Ethereum. This is the last step in the initialization and ensures that the public key available always represents the correct key set up in the BloSS instance. After the initialization is done, the PollenBlockchain class is a passive entity that provides three main functions to other classes: Reporting and retrieving attack reports and marking attack reports as blocked in the system contract.

5.2 Defense Scenario

Figure 5.7 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 the attack information and encrypting it to post the IPFS hash to the Ethereum later. Attack information hash is connected to a Boolean, indicating whether the information has already been accessed by *M*'s AS to block the attackers. This is done to reduce access to IPFS and Ethereum to access attack information and the public key of *T*'s AS. However, incentive schemes necessary to realize a true Mitigation-as-a-Service (MaaS) offering, as outlined in [141], are out of the scope of this implementation of BloSS.

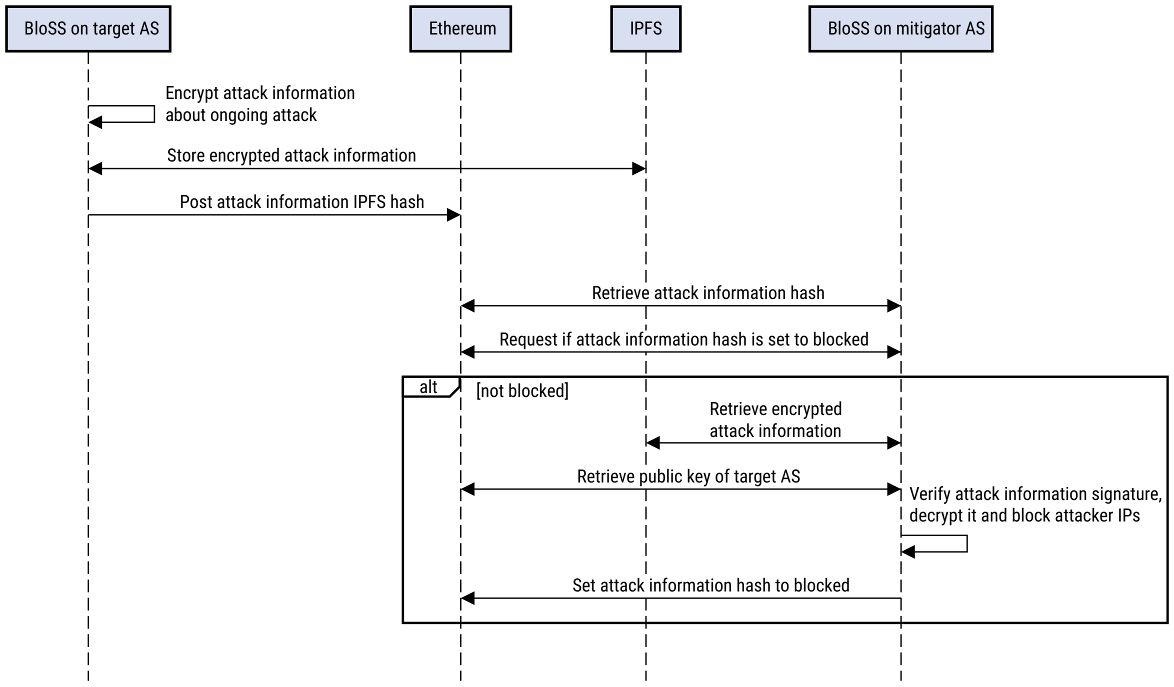

BloSS on target AS
Ethereum
IPFS
BloSS on mitigator AS
Encrypt attack information about ongoing attack
Store encrypted attack information
Post attack information IPFS hash
Retrieve attack information hash
Request if attack information hash is set to blocked
alt
[not blocked]
Retrieve encrypted attack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

Figure 5.7: BloSS Defense Scenario Including a T and an M, ASes

5.2.1 Signaling Attacks

An attack report specifies the network of the attackers. This is used to enable efficient signaling among a large number of ASes in the Ethereum. A single relay contract keeps track of networks participating

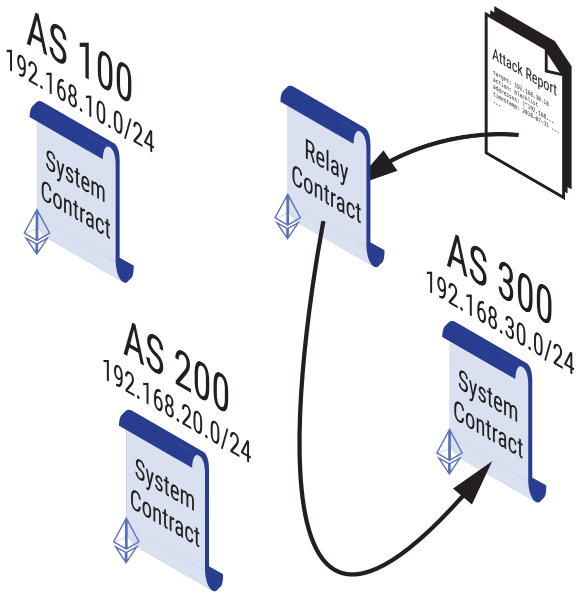


Figure 5.8: Example of an Ethereum Contract Setup with the Relay and Cooperative Signaling Protocol Contracts

in the collaborative DDoS defense and maps each network to the system contract managing that network. As soon as an attack report is posted to the relay contract, it directly sends it to the correct system contract to allow the responsible BloSS instance to act on it.

This relaying of attack reports represents a significant change from the old system, where only a single, smart contract was responsible for attack reports. With the new system, it becomes possible to implement a simple access control scheme where only the owner of a specific system contract can access the attack reports stored in that contract. Due to the openness of the Ethereum, this simple scheme does not deter determined third parties to access the attack reports stored in any contract by examining the individual blocks and finding the point at which the report has been stored. Apart from simple access control, this scheme does, however, also allow to reduce the number of calls necessary for a specific BloSS instance to get the attack reports it is interested in, *e.g.*, the ones mentioning attackers from a network it is managing. The number of networks managed by a single system contract is not limited, and the system contract registers the network for which it wants to receive attack reports with the relay contract.

System contracts also represent an essential part of the encryption scheme since the public keys required to encrypt symmetric keys for the encryption of attack reports and to sign said attack reports are stored directly in a system contract. Since only the creator of the system contract, *i.e.*, the BloSS instance responsible for networks that the system contract was registered for, can change the smart contract's fields, only they can change the public key. This means that building upon the secure ledger inherent in the Ethereum allows us to build a highly decentralized but still secure signaling system.

5.2.2 Attack Report Information

The attack information is the communication payload exchanged between individual BloSS instances, *i.e.*, the file that describes the request for a mitigation action on attackers' IP addresses. It carries the relevant data to indicate the target and the source of an attack, enabling the collaborative mitigation of a large-scale DDoS attack directly at the source. Following data points are part of the attack information (also listed in 5.3):

```python
class AttackReport(object):
    def __init__(self, target, action, timestamp,
                 subnetwork, addresses, hash=None):
        self._target = self.target = target
        self._action = self.action = action
        self._timestamp = self.timestamp = timestamp
        self._subnetwork = self.subnetwork = subnetwork
        self._addresses = self.addresses = addresses
        if (hash is None and target is not None
                and action is not None
                and timestamp is not None):
            self._calculate_hash()
        else:
            self._hash = hash
```

Listing 5.3: Fields of the Attack Report Implemented in BLoSS

- **Target**: The IP address being targeted by a DDoS attack.
- **Action**: Action to take against the attack. In the version of the BloSS outlined in this thesis, this is limited to the "blackhole"[163] action discarding attack traffic directly.
- **Timestamp**: The purpose of the timestamp is to ensure that outdated attack information is no longer considered for blocking, making sure that no unwanted side effects can occur.
- **Network**: The network address in which attacker addresses belong.
- **Addresses**: IP addresses of the attackers.

- **Hash**: A hash computed from the target address, timestamp, network, and action. This hash is used to identify a set of attack information uniquely.

There are two essential considerations in this file. The first is related to the confidentiality of the information in the report, and the second is related to performance, considering that this file can be relatively large, depending on the scale of the attack. A single attack might often produce multiple attack reports since each report represents a specific network of attackers to simplify the attack's mitigation. Based on the network, the attack report can directly be sent to the AS managing the network from which the attack originates.

The JSON [24] format was selected for exchanging object information among BloSS instances. It has a lightweight design to transmit data and allows simplifying operations in the Python ecosystem around BloSS. Listing 5.4 shows an example of a JSON signaling list.

```
{
  target: "192.168.10.5",
  action: "blackhole",
  timestamp: "2019-10-10T08:32:02+00:00",
  subnetwork: "192.168.30.0/24",
  addresses: {
   "123.123.125.123",
    "190.23.65.86",
    "169.56.85.74",
    "69.58.74.85"
  }
}
```

Listing 5.4: Example of Storing IP Address Lists Formatted as JSON Objects

For every host, there is a list of attackers assigned (*i.e.*, blacklisted in the field 'addresses'. The example in Listing 5.4 shows two attacked hosts listed behind the keyword `target` and each of them has a list of four attackers assigned to them.

5.3 Off-chain Data Exchange

Pollen datastore includes an encryption component not directly visible in Figure 5.2 since it is an inherent part of the entire module. 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

ASes since the attack information can be sensitive in regards to implicating individuals both as victims of an ongoing DDoS attack or as the perpetrators of said attack.

Encryption and decryption are built with the Python Cryptography library [61] and the choices for cryptographic algorithms and key lengths and other cryptographic details are based on an article by Colin Percival [184]. 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 [61].

Asymmetric encryption is used for two tasks in PollenEncryption: To encrypt the symmetric key and cryptographically sign the unencrypted attack report with the private key of the sender (*cf.*, Figure 5.9). Signing the attack report with the private key allows the receiver to verify the authenticity of the attack report 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 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 2,048 bit [184].

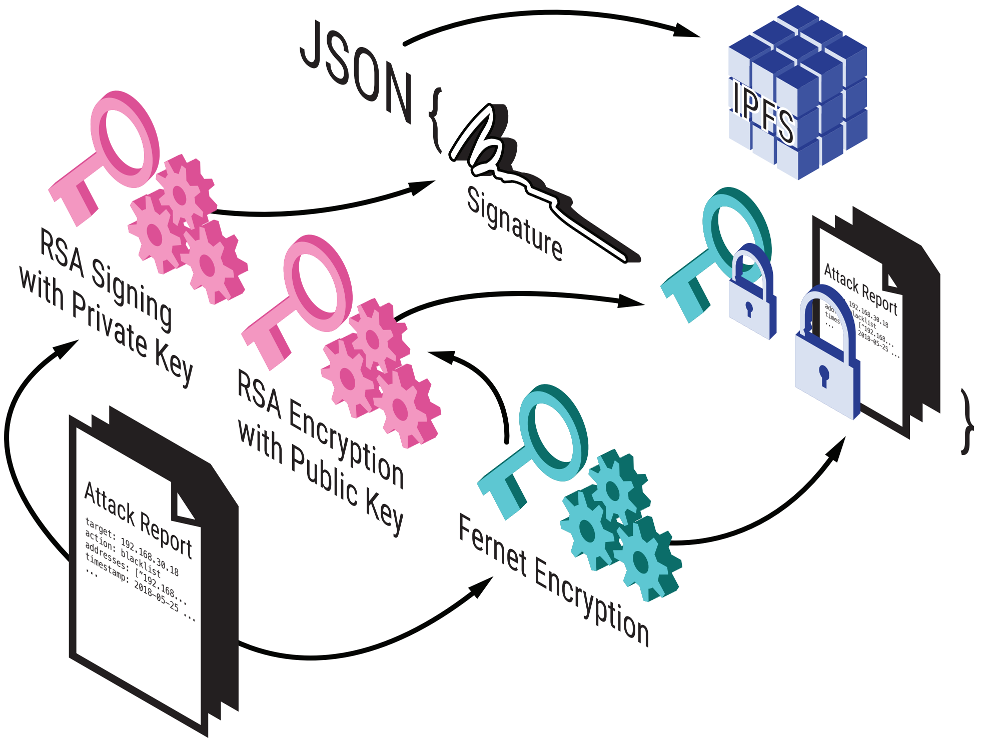


Figure 5.9: Encryption Procedure for Off-chain Data Transfer via IPFS

After encrypting the attack report and symmetric key and 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 straightforward handling through the Stalk and BloSS REST APIs. In addition to encrypting each set of attack information when posted to IPFS, communication between the Pollen datastore module and IPFS is encrypted with the libp2p-secio [120] 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 impossible to ascertain which AS accessed which attack information when merely looking at the communication between IPFS and AS. This added anonymity is an additional factor contributing towards increased confidentiality.

Communications between individual Ethereum nodes are also encrypted as detailed in the DEVp2p [253], which contributes to 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.

5.4 BloSS Management Dashboard

In a context where ASes rely on cybersecurity specialists to make critical decisions regarding threats, it is necessary to structure and categorize data such that visualization "makes sense" to the analyst [230]. As a cooperative defense involves multi-disciplinary concepts and the decision-making process usually requires a low response time from the user, selecting an appropriate type of graphical representation and flow of interaction is not a straightforward task [126].

As mentioned in the Bloss Module Section 5.1.1, the system is modular based on management API's providing interaction either via terminal or a Web-based interface. In this regard, Figure 5.10 to enable communication with the front-end dashboard by collecting and encapsulating information from the various BloSS components. The Express REST API communicates with two components to feed the dashboard: Stalk and BloSS. While BloSS is the core component which receives mitigation requests and signals ongoing attacks through Pollen and stops attack traffic, Stalk monitors the network traffic of the underlying networking infrastructure.

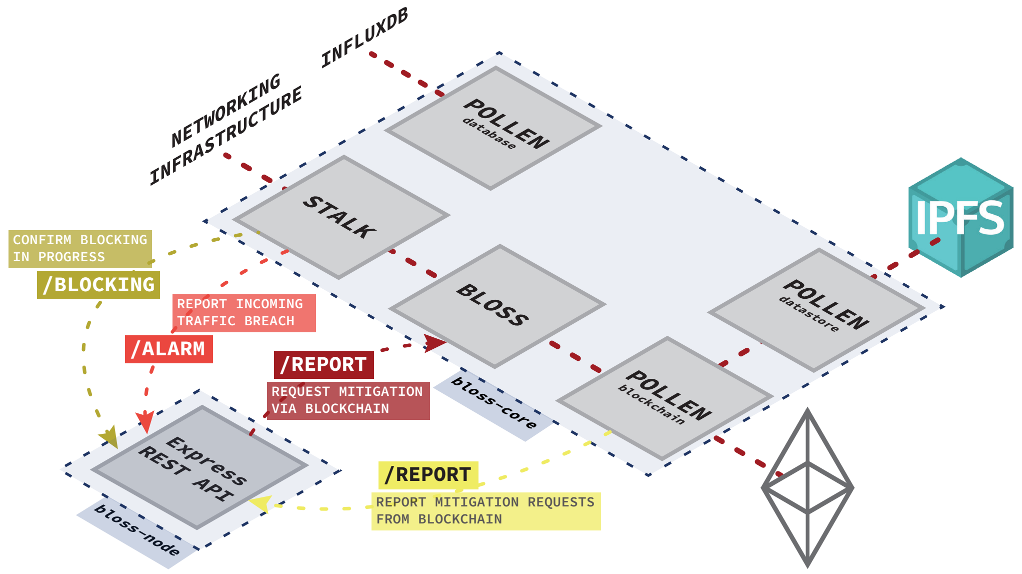

INFLUXDB
NETWORKING INFRASTRUCTURE
POLLEN database
STALK
BLOSS
POLLEN datastore
POLLEN blockchain
IPFS
bloss-core
CONFIRM BLOCKING IN PROGRESS
/BLOCKING
REPORT INCOMING TRAFFIC BREACH
/ALARM
/REPORT
REQUEST MITIGATION VIA BLOCKCHAIN
Express REST API
bloss-node
/REPORT
REPORT MITIGATION REQUESTS FROM BLOCKCHAIN

Figure 5.10: Communication of BloSS Core Components (bloss-core) with the REST API

Then, the BloSS node facilitate the relaying of information to the dashboard, whose challenge is to reduce the complexity of information provided to a network operator to support decision-making without adding the overhead of unnecessary information. In the presented use cases (*cf.*, Chapter 6), a new alarm is displayed to and evaluated by the human operator. An alarm can be classified as a valid threat and whether mitigation should be requested, or the alarm can be ignored. Thus, the complexity is reduced and based on a known layout from software development (*e.g.*, the Kanban board [173]), leading to a low entry barrier of understanding the user interface. The dashboard remains predictable as the main layout does not change, and empty states (*e.g.*, no new MREQ) indicate where new elements have to appear in the layout.

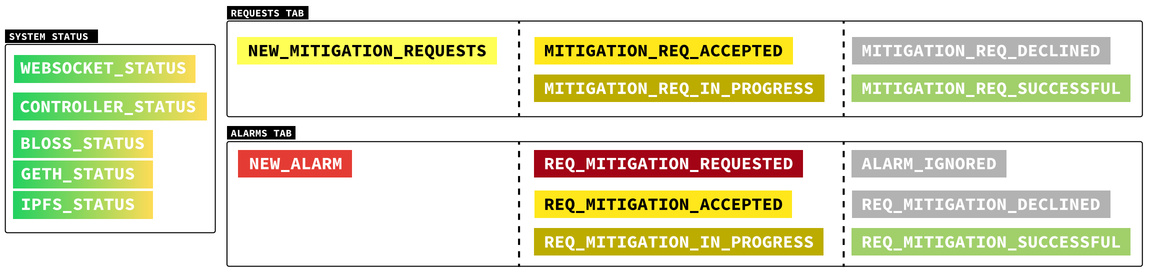


Figure 5.11: Schematic View of the Network Operator's Dashboard

Figure 5.11 illustrates the three main sections, the status of the system components, the visualization of requests for collaborative defense on the Mitigator's (M) side, and the alarms tab allowing the request for collaborative help on the Target's (T) side. An AS may act either as M or T, so these tabs are available to the network operator. These tabs display three columns with progressing states from left to the right, in which the left row contains new events [173]. The middle row comprises events in progress and updates them accordingly to changes in their status. The right row represents a log of elements finished or declined. The *REQUESTS_TAB* contains incoming mitigation requests. The *ALARMS_TAB* contains alarms that were triggered as soon as a pre-defined inbound traffic threshold breach occurs. The *REQUESTS_TAB* contains incoming mitigation requests.

Combining meaningful naming and appropriate coloring of states enhances the user experience for the analyst. It is crucial to standardize the use of colors in security visualization [75] to enhance rapid information processing. As already indicated in Figure 5.11, the colors are chosen according to their meaning in the process context. The state management, therein the naming and coloring of states is another crucial part of security visualization indicating the perspective of the AS.

5.4.1 Target Interaction Flow

The state diagram is triggered by a DDoS attack by nodes from other networks, operated by other ASes. In short, the AS of the attacked target will issue an attack report to the BC, which will be retrieved by the AS of the attacking nodes, henceforth called Mitigator (M). First, a network monitoring system sends traffic data to BloSS, determining whether the pre-defined traffic thresholds are breached. For each breach, an attack report is sent to the BC and persisted with an initial status of ● `NEW_ALARM` if the same attack report hash has not been persisted yet.

The lifecycle of an attack report starts with the state ● `NEW_ALARM` as soon as a traffic breach occurs. The analyst of the target AS has to decide whether the attack should be reported or not (hence ignored). If the analyst ignores the alarm, the attack report changes state to ● `ALARM_IGNORED` and the lifecycle of the attack report ends. Otherwise, it is possible to request the cooperative mitigation and the state changes to ● `REQUEST_MITIGATION_REQUESTED`. This means that the attack report is submitted to the BC and retrieved by the Mitigator ASes. It is important to note that at this point, an attack report with the state of ● `NEW_MITIGATION_REQUEST` is created on the Mitigator's side (as soon as it was retrieved from the BC), hence starting the state machine in Figure 5.13.

Further, the Mitigator decides whether or not to accept the request for mitigation. If the request is declined, the state changes to ● `REQ_MITIGATION_DECLINED` and the lifecycle of the attack report ends. Otherwise, the state changes to ● `REQ_MITIGATION_ACCEPTED`, meaning that the mitiga-

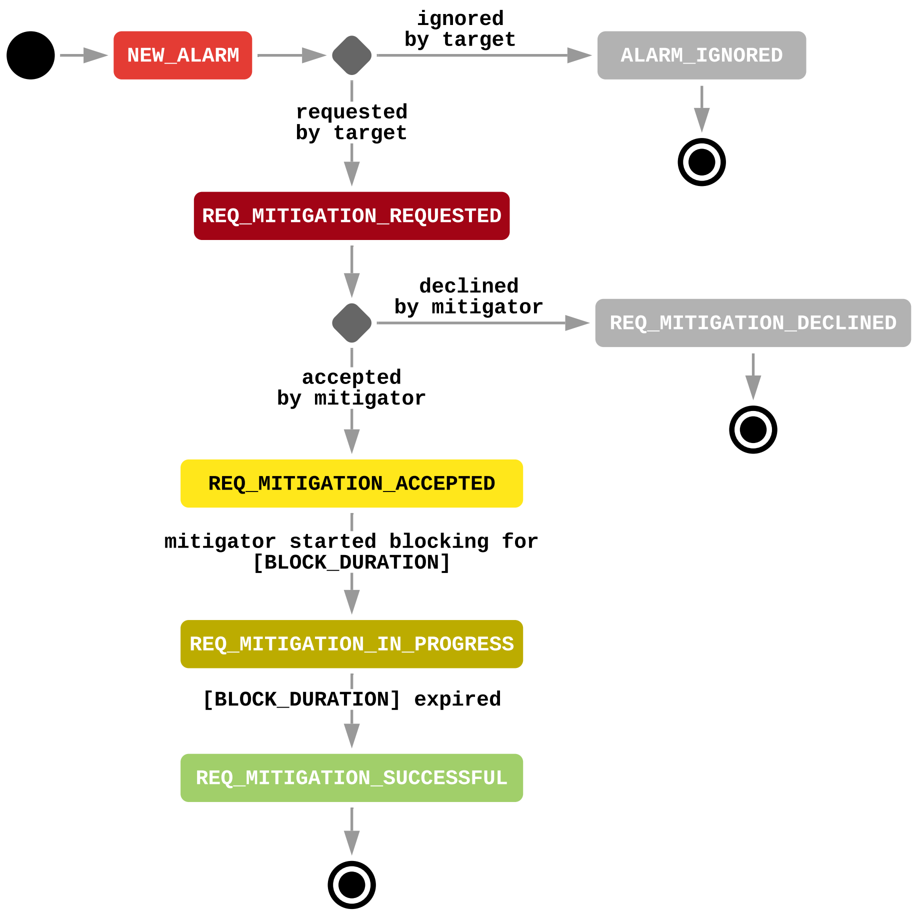


Figure 5.12: State Diagram to Request Mitigation

tor AS will block any harmful traffic. Therefore, from the moment that the mitigator starts blocking traffic, the state changes to ● `REQ_MITIGATION_IN_PROGRESS`. After the blocking time expires, the mitigation successfully ends, and the lifecycle of the attack report changes to the state of ● `REQ_MITIGATION_SUCCESSFUL`. Then, the AS of the attackers' origin will retrieve the attack reports and decide whether to accept the requests. From here on out, the process from the target's perspective is finished.

5.4.2 Mitigator Interaction Flow

An AS with the role of mitigator periodically checks for attacks signaled *i.e.*, reported on its SC. For example, when an AS is under attack, it can report the attackers directly on the SC of ASes responsible for the networks, from where the identified hosts originated. In this regard, BloSS instances

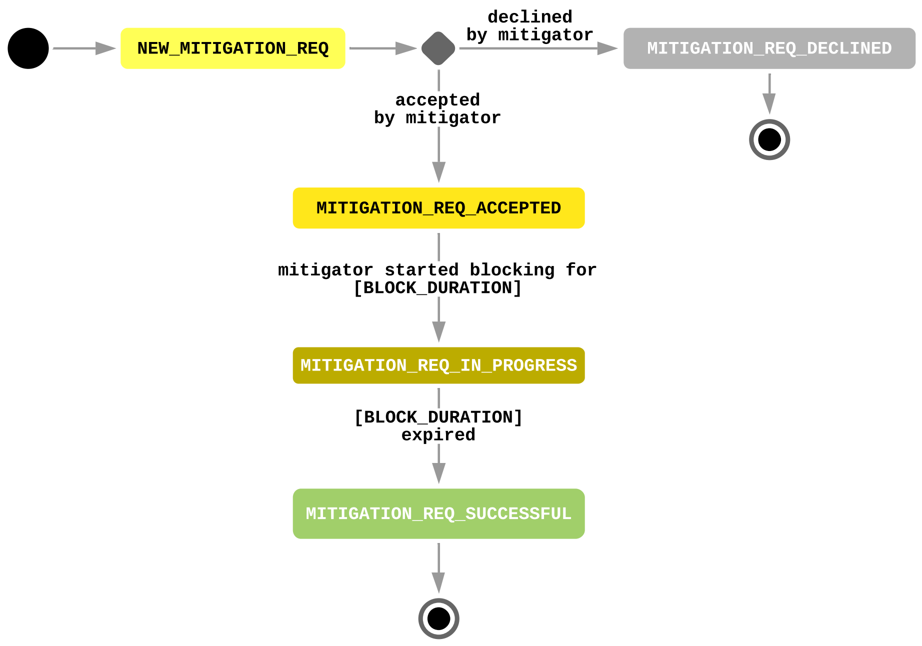


Figure 5.13: State Diagram to Decide on Mitigation Requests

are configured to periodically check for changes in the states of their Smart Contracts, *i.e.*, whether there are requests, and further creating events on the dashboard and updating whenever their status is changed. As soon as an attack report is reported and retrieved via IPFS, it is persisted in BloSS and displayed in the dashboard with an initial status of ● `NEW_MITIGATION_REQUEST` if the same attack report hash has not been persisted yet.

The analyst on the mitigator M side has to decide whether the mitigation should be accepted. If the analyst declines, *e.g.*, provided incentives are not attractive, or the AS is unavailable to perform the requested mitigation; the attack report state is modified to ● `MITIGATION_REQ_DECLINED` and the lifecycle of the attack report ends. Otherwise, the state changes to an intermediary state of ● `MITIGATION_REQ_ACCEPTED`.

As soon as the AS performs the requested mitigation, traffic of reported attackers (hence mitigating the attack), the state changes to ● `MITIGATION_REQ_IN_PROGRESS`. When the mitigation is confirmed and relayed to the dashboard, the mitigation successfully ends, and the lifecycle of the attack report changes to the state of ● `MITIGATION_REQ_SUCCESSFUL`.

5.5 Key Observations

This Chapter introduced the decentralized BloSS application, which interacts with the cooperative signaling protocol deployed on-chain. While the on-chain protocol regulates the interaction between alliance members, BloSS is responsible for the interface with the protocol and the network management system, which, as a proof of concept, was based on SDN. Hence, BloSS is responsible for storing specific configurations for each member, such as the definition of reputation thresholds or legal specifications concerning which members in certain regions the cooperation can be made. Also, amounts are determined for calculating the incentive required to be paid in the event of attacks and the amount for possible mitigation services.

The field of DDoS defense system contains an increasing number of competing approaches to solve the multi-domain DDoS mitigation problem. The BloSS is best comparable with the decentralized hybrid mechanisms as presented in Chapter 3. These approaches mainly differ in the communication mechanism they employ.

While the DOTS architecture pioneered by the IETF is built on top of a purpose-built communication protocol specifically designed for the use in DDoS defense signaling, other approaches such as DefCOM base their communication mechanisms on existing overlay network technology to enable signaling in a peer to peer manner [169]. Compared to these two approaches, BloSS is more akin to DefCOM than DOTS since it also builds on an existing system in the form of the Ethereum blockchain instead of developing a new communication approach from scratch. Leveraging the highly distributed nature and secure ledger aspects of blockchains allows the BloSS to securely and easily scale to the demand of a modern distributed DDoS defense system based on SDN (underlying management system) or NFV (scaling mitigation actions on attacking traffic). DefCOM on the other hand relies on complex peer to peer message exchanges that are more prone to failure than the consensus-based blockchain system utilized by the BloSS.

By building the demonstration implementation of the BloSS with SDN and NFV-capability in mind, advantages of quick deployment inherent to competing systems like CoFence [195] or Bohatei [69] are already part of the re-engineered BloSS. With the modular aspect of the BloSS in mind, BloSS is not limited to SDN based networking infrastructures or only being able to provide the BloSS as a NFV-based solution, but can adapt Stalk (*i.e.*, the networking module of the BloSS) to various infrastructures while still maintaining the simple, yet efficient RESTful interface between the BloSS module and Stalk.

Lastly, BloSS leverages the BC capabilities to define individual agreements or conditions in a smart contract to perform a mitigation service in cooperative network defense. Conditions to perform a mitigation service are described in an individual smart contract, which expresses the networks managed by a peer and the conditions (*e.g.*, cost to operate the infrastructure) to perform a mitigation service. In a mitigation request, service and validation timeouts are agreed between the peers to establish deadlines for mitigation service completion and the payout of the reward (cost) upon submitting a mitigation proof.

6 Evaluation

BloSS was evaluated in different stages (*i.e.*, locally and globally) and scopes (individual components) to emphasize its simplicity of deployment and operation for the signaling of DDoS attacks. Considering these different stages and scopes, an organization of the these different evaluations scopes is presented in the beginning of the Chapter. Further, the summary of achievements and their respective discussions concerning how cooperative defenses tackled challenges is presented at the end of the Chapter.

Local evaluations aims at assessing the functionality and correctness of BloSS components for subsequent global evaluations, mainly for assessing the on-chain cooperation protocol that rejects the operation of BloSS. Global evaluations relied on the configuration of an Ethereum-based blockchain network based on Virtual Machines deployed with BloSS instances to evaluate performance aspects (end-to-end latency) of each possible outcome of the on-chain cooperative protocol, as well as the latency for o off-chain exchange data via IPFS.

6.1 Roadmap of BloSS Evaluations

Several evaluations were performed during the development and refinement of the system to achieve the version presented in this thesis. This Section outlines their specific goals toward reaching the overall thesis goals.

1. **BloSS Functionality and Correctness (Section 6.2)**: evaluations performed at the local prototype and deployed in the cluster depicted in Figure 6.1 to evaluate overall correctness and functionality (*e.g.*, signaling addresses, mitigating attacks, interaction with Ethereum).

2. **Dashboard Usability (Section 6.3)**: evaluation performed in the local cluster improving the previous system with a focus on improving usability. Thus, a dashboard was designed and deployed in a modular version of BloSS in order to simplify its usage.

3. **Off-chain Signaling Latency (Section 6.12)**: evaluation performed both locally and globally, in which the goal was to evaluate the performance in terms of latency of the off-chain communication channel based on an encrypted channel.

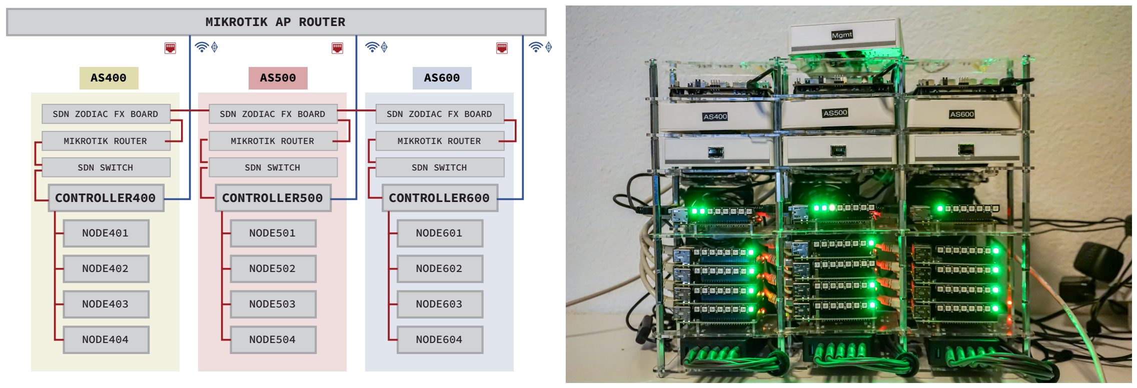
4. **Reputation Scores (Section 6.5)**: performed locally based on a emulation environment to create multiple contracts of target's and mitigator's. The goal was to evaluate the behavior of beta reputation thresholds and the interaction of peers over time (*e.g.*, how many interactions needed to accept or deny interactions). Summarized results of the experiments are presented within the section and expanded results are presented in Appendix D.

5. **Cooperative Signaling Protocol Latency (Section 6.6)**: evaluation performed locally and globally. The goal was to assess the correctness of the protocol in local assessments (a simulation based on Truffle and based on the Rinkeby Ethereum testnet). The goal of the global evaluation was to assess the signaling performance (in terms of latency) in all possible outcomes of the protocol, as well as its difficulty of deployment and operation in VM's across the world.

6. **Smart Contract's Vulnerabilities (Section 6.7)**: based on local experiments using different automated vulnerability assessment tools. The objective was to identify, analyse and compare the findings in terms of vulnerabilities in the BloSS SCs. Results of the analysis are summarized within the section based on the list of vulnerabilities available in Appendix E.

6.2 BloSS Functionality and Correctness

BloSS was deployed on a physical single-board computer cluster (*cf.* Figure 6.1). Three isolated and identically configured ASes were built: AS 400, AS 500, and AS 600 with each AS consisting of four host nodes used to initiate the attack traffic, and two controllers, which host the BloSS as well as the Ethereum BC and IPFS [17] nodes. Hosts are based on Raspberry Pi Model B (RPi), and controllers use ASUS Tinker Board devices, which provide greater computational capacity than RPIs.



(a) Schematic Representation **(b)** BloSS Hardware

Figure 6.1: BloSS Schematic View and Hardware Prototype

The schematic representation (*cf.* Figure 6.1) at the left-hand side (a) shows the network configuration in the BloSS hardware (b). The management network is configured via wireless between the three routers, the two auxiliary controllers, and the gateway. Furthermore, additional MikroTik routers and switches are necessary since the Zodiac FX switches only provide four ports, which is not sufficient to connect all hosts of an AS. The SDN controller responsible for the Zodiac FX switches is directly specified in the Zodiac FX Web interface.

6.2.1 Configuration

BloSS has been evaluated by utilizing the iperf network bandwidth measurement tool [240]. An instance of iperf is installed on the last compute node of AS 600 with IP address 192.168.30.18 and is listening for incoming iperf connections on UDP port 5000. To start an attack, the following command is issued to all hosts from AS 400 and AS 500:

```
# iperf -c 192.168.30.18 -u -b 20m -t 10 -i 1 -p 5000
```

The remaining three hosts from AS 600 (192.168.30.15, 192.168.30.16 and 192.168.30.17) are idle throughout the attack since it is impossible to block traffic flowing from them to the target host 192.168.30.18. Only traffic between ASes goes through the SDN switches and can, therefore, be affected by the SDN controller. Intra-AS traffic does however go through the switch and router to which all compute nodes of an AS are connected. This is based on the limited number of Ethernet ports on the Zodiac FX switches.

6.2.2 Signaling Latency

This is an important aspect to show that BloSS is capable of mitigating attacks regardless of whether the attack volume accounts to more than eight times the available throughput of the target system (in the case of the experiment with 100 MBit/s bandwidth per attacker) or only around 80% of the available throughput. To evaluate the delay from the start of an attack to the attack being entirely blocked by the cooperative defense, traffic statistics from InfluxDB were used. PollenDatabase, the database component of the Pollen module, posts traffic volume information every second to the InfluxDB. To get the delay, only database entries with a bandwidth value of over 1 Mbit/s were considered since these correspond to the entries written during the attack. Since traffic information is written every second, a select statement was performed to count the number of returned rows corresponding to the amount of time passed from beginning to end of the attack.

This does only work if only a single attack in terms of volume is present in the database. To ensure this was the case, the database was routinely cleared before each attack to eliminate old attack volumes. Table 6.1 shows delays recorded for 4 different bandwidths and over 10 attacks for each bandwidth. The bandwidth is set per compute node, which means at a bandwidth of 10 Mbit/s, a total attack volume of 80 Mbit/s is created and routed toward the target compute node.

Table 6.1: Delay Until Attacks with Different Bandwidths are Blocked

Bandwidth / Delay [Mbit/s]	**10 Runs [s]**										**Average [s]**
10	34	34	28	20	28	27	26	37	23	19	27.6
20	32	18	34	18	28	31	32	31	16	32	27.2
40	34	25	39	24	31	35	25	29	31	24	29.7
100	34	26	40	29	29	24	35	28	36	35	31.6

It is important to note that one of the most significant contributing factors to the delay is the block period of 5 seconds. After sending an attack report, 5 seconds pass until the attack report becomes available to all BloSS instances. Since attack reports are based on subnetworks, a minimum of two reports need to be sent out to cover the two attacking subnetworks 192.168.10.0/24 and 192.168.20.0/24. If one of the attacking hosts is detected with a delay, another attack reports need to be filed, which consumes another 5 seconds. The average mitigation time of around 29 seconds overall experiments shows that the BloSS is a fast-acting mitigation solution capable of quickly diminishing even high-bandwidth threats.

6.2.3 CPU Overhead

The additional processing resources required to run the BloSS are a vital metric to decide whether it is worth tolerating the added strain on the CPU to mitigate DDoS attacks that could otherwise not be handled due to their distributed nature. The analysis consists of full mitigation, including eight attack hosts and one target host with a total attack volume of 160 MBit/s. The experiments are split into mitigations with encrypted attack reports and mitigations with encryption turned off entirely. With this differentiation, the added confidentiality provided through the encrypted attack reports can be contrasted with possibly increased CPU usages.

Table 6.2: CPU Usage Statistics [%]

CPU Usage	**AS 400**		**AS 500**		**AS 600**	
	E	***P***	***E***	***P***	***E***	***P***
Minimum	5.8%	4.8%	8.2%	6.6%	8.8%	8.4%
Maximum	21.2%	18.2%	27.8%	24.6%	34.4%	34.6%
Average	13.2%	10.5%	17.2%	14.8%	16.3%	15.8%
Median	13.4%	10.3%	16.9%	17.8%	15%	14%

Legend: Encrypted (E), Plain-text (P)

In order to better understand the variations in CPU usages between the two scenarios with and without encryption, minimum, maximum, average, and median CPU usage across all ASes and all runs per experiment have been compiled into two tables. Each entry represents the average over five runs for the specific statistical indicator, and AS. Table 6.2 shows statistics for the five runs with encryption turned on (E - Encryption) and off (P - Plain-text). Apart from the ability of the BloSS to quickly mitigate attacks, it should not consume too many resources while doing so. The ten experiments suggest that the highest degree of CPU usage occurs at the point in time when attacks are reported to the BC or retrieved by the mitigators.

An interesting aspect lies in the statistical metrics that show over 5 experiment runs each, that the difference in CPU usage between the BloSS instances entirely relying on encryption is only around 2% higher than the other half of the experiments with encryption turned off. This is a clear indicator that encryption does not add considerable overhead to the BloSS and can, therefore, safely be left turned on, especially considering that encryption provides a high degree of confidentiality and allows to ensure the integrity of the attack reports through the verification of cryptographic signatures.

6.2.4 Discussion of CPU Overhead and Signaling Delay

By collecting CPU usage information for a short period before and after the attack occurs, it can also be shown that the attack itself does not contribute to a considerable increase in CPU usage, with the exceptions of the short bursts to post-attack reports and retrieve them from the BC. The baseline CPU usage of around 15% has to be attributed to the Python programming language, which is an interpreted language, creating a small overhead when running programs written in that language. Apart from this, the BloSS requires to periodically analyze traffic information and request attack reports from the BC, which both contribute to the baseline CPU usage.

Since 15% of average CPU usage is not a negligible amount of processing resources consumed, this value can be further improved by reducing the aforementioned periodic tasks' frequency. However, this would result in increased delays to mitigate ongoing attacks. Since the delay is an essential factor in enabling an efficient incentive scheme, later on, this trade-off does not seem reasonable, and keeping the frequencies at the current high rate is advisable.

Finally, the test environment considered a SoC (System-On-a-Chip) computer like Raspberry Pi that has constrained resources in contrast to standard servers. Besides, as BloSS is a software-based approach, the deployment and operation of BloSS are simplified while the performance regarding the latency for the propagation of attack signaling information is sufficient for ASes to react to current DDoS attacks.

6.3 Dashboard Usability

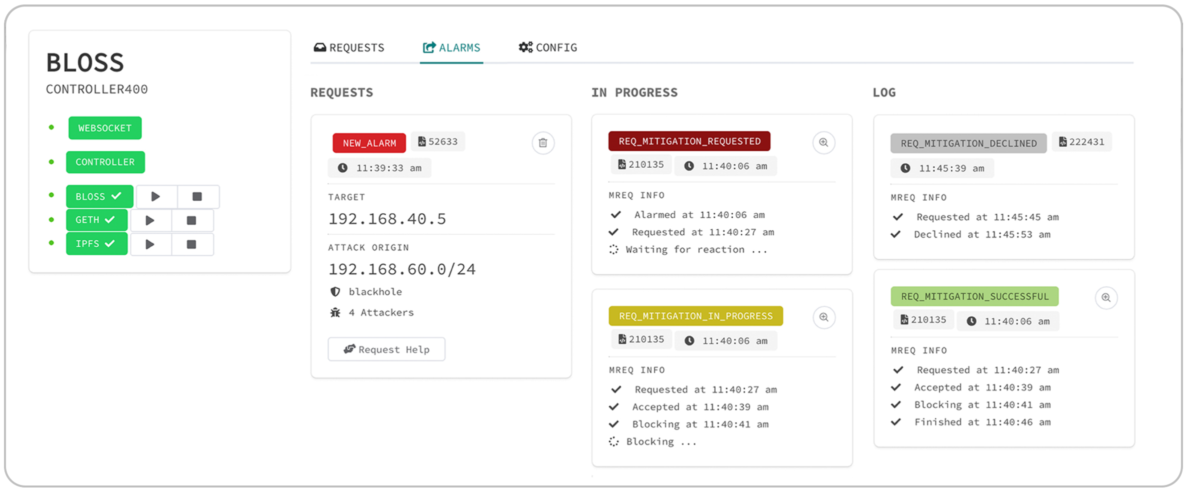
The dashboard implemented in the BloSS prototype was analyzed based on use cases, while displaying the flow of information in the dashboard during experimental attacks. The system configuration is detailed in Subsection 6.3.1, Use Case 1 (UC1) presenting the target's view of a request for mitigation or alarm ignore is described in Subsection 6.3.1, and Use Case 2 (UC2) presenting the mitigator's view to accept or decline a mitigation request is presented in Subsection 6.3.1.

6.3.1 Configuration

Hosts connected to AS 400 (4 hosts), AS 500 (4 hosts), and AS 600 (three hosts) will start a flood-based DDoS attack on a single host on AS 600. Controllers of all ASes are configured with a static inbound traffic threshold to determine whether there is an ongoing attack. Based on this, alarms on the AS 400 dashboard will show a warning about the attack, and the operator can decide to request the cooperative mitigation or the ignore alarm (UC1). If cooperative mitigation is requested, the dashboard on AS 600 will display the mitigation request (UC2).

UC1 - Request Mitigation or Ignore Alarm

The precondition for UC1 is that all the BLoSS services are active and operating correctly (*cf.*, Figure 6.2). In this regard, the left-hand side of the dashboard shows the interface (BloSS AS 400) that presents services' status, which can have its services activated or deactivated modules based on a click. As soon as inbound traffic breaches the pre-defined threshold (*i.e.*, a DDoS attack is detected), alarms are sent to the dashboard, and the operator has to decide on whether to request or ignore these alarms. Then, the dashboard displays a message ● `NEW_ALARM`, which is seen in the Requests column in Figure 6.2, and the security analyst can decide on whether to request for collaborative defense or ignore the alarm.



REQUESTS

NEW_ALARM 52633
11:39:33 am
TARGET
192.168.40.5
ATTACK ORIGIN
192.168.60.0/24
blackhole
4 Attackers
Request Help

IN PROGRESS

REQ_MITIGATION_REQUESTED
210135 11:40:06 am
MREQ INFO
✓ Alarmed at 11:40:06 am
✓ Requested at 11:40:27 am
Waiting for reaction ...

REQ_MITIGATION_IN_PROGRESS
210135 11:40:06 am
MREQ INFO
✓ Requested at 11:40:27 am
✓ Accepted at 11:40:39 am
✓ Blocking at 11:40:41 am
Blocking ...

LOG

REQ_MITIGATION_DECLINED 222431
11:45:39 am
MREQ INFO
✓ Requested at 11:45:45 am
✓ Declined at 11:45:53 am

REQ_MITIGATION_SUCCESSFUL
210135 11:40:06 am
MREQ INFO
✓ Requested at 11:40:27 am
✓ Accepted at 11:40:39 am
✓ Blocking at 11:40:41 am
✓ Finished at 11:40:46 am

Figure 6.2: `bloss-dashboard` UI Running on AS 400. `ALARMS_TAB`

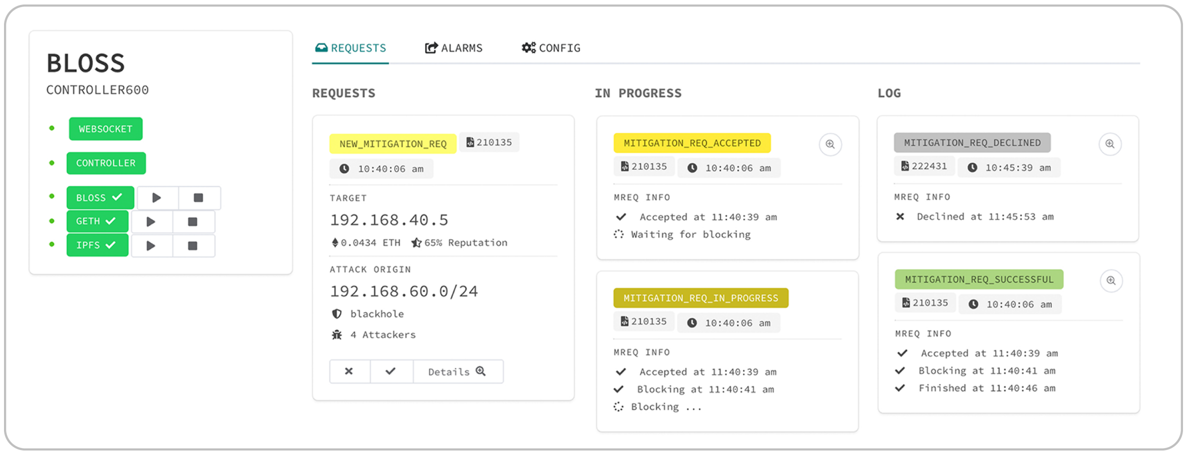
In the following, the security analyst should decide whether to request the cooperative mitigation or ignore the alarm. If mitigation is required, a request is sent to BloSS, which submits a transaction to the BC, and the request is moved to the column "In Progress" on the dashboard with status ● `REQUEST_MITIGATION_REQUESTED`. Then, the target operators on AS 500 and AS 600 can either decline the request for mitigation and the status of the attack_report changes to ● `REQ_MITIGA-`

TION_DECLINED or accept the request for mitigation and the status of the attack_report changes to ● REQ_MITIGATION_ACCEPTED.

As soon as the involved mitigator involved starts blocking, *i.e.*, applying a mitigation action such as blackholing traffic or blocking hosts listed as attackers, the attack_report is also ● REQ_MITIGATION_IN_PROGRESS. After the expiration of the maximum block duration, the attack_report is completed, and thus, ends in the status ● REQ_MITIGATION_SUCCESSFUL. The history of requests, besides registered in the BC (not disclosing the details, *e.g.*, blacklisted addresses), is available to all members of the alliance. The events involving each domain are also registered and grouped in the "Log" column and can have their details visualized on demand.

UC2 - Accept or Decline Mitigation Request

This use case, in contrast to UC1, considers the mitigator's perspective available in the "Requests" tab (*cf.* Figure 6.3). Mitigation requests are periodically retrieved from the BC, which are relayed to bloss-dashboard to display to the operator with the status ● NEW_MITIGATION_REQUEST. Incoming requests can be grouped depending on the number of requests, and the operator can click on specific events for more details.



NEW_MITIGATION_REQ 210135
10:40:06 am
TARGET
192.168.40.5
0.0434 ETH 65% Reputation
ATTACK ORIGIN
192.168.60.0/24
blackhole
4 Attackers

MITIGATION_REQ_ACCEPTED
210135 10:40:06 am
MREQ INFO
✓ Accepted at 11:40:39 am
Waiting for blocking

MITIGATION_REQ_IN_PROGRESS
210135 10:40:06 am
MREQ INFO
✓ Accepted at 11:40:39 am
✓ Blocking at 11:40:41 am
Blocking ...

MITIGATION_REQ_DECLINED
222431 10:45:39 am
MREQ INFO
✕ Declined at 11:45:53 am

MITIGATION_REQ_SUCCESSFUL
210135 10:40:06 am
MREQ INFO
✓ Accepted at 11:40:39 am
✓ Blocking at 11:40:41 am
✓ Finished at 11:40:46 am

Figure 6.3: bloss-dashboard UI Running on AS 600. REQUESTS_TAB

At this point, an operator can decide to deny the request (and the attack_report's status changes to ● MITIGATION_REQ_DECLINED), or to accept the request (and the attack_report's status changes to ● MITIGATION_REQ_ACCEPTED). Once requests are accepted or denied, the dashboard forwards the request to BloSS, which submits the transaction to the BC. Then, the mitigation service has a deadline to be completed, which can be visualized in the time stamps registered on the dashboard.

This service deadline is required for the mitigator to upload a proof of completion of the requested mitigation task, ensuring a cooperative mitigation service's effectiveness.

Traffic from attacking hosts is mitigated, the attack_report's status changes to ● `MITIGATION_-REQ_IN_PROGRESS`. At this point, the mitigator can act rationally and upload a proof or miss the upload. However, it is not possible to verify the truthfulness of the quality of the (proof of) service performed by M (an issue discussed in [143]). Even if the BC can preserve a transparent audit trail for all transactions, it cannot compensate for lack of ground-truth. This holds for the uploaded proof of service and user-defined, subjective ratings, in which there is no automated way to determine the truthfulness of proof or rating fully.

Once the service is completed, and the proof is uploaded (*e.g.*, log showing a mitigation action), or the mitigation deadline is expired, the service is considered complete, and the status is changed to ● `MITIGATION_REQ_SUCCESSFUL`. Similarly to UC1, all mitigation service events involving each domain are registered and grouped in the "Log" column, and its details can be visualized on demand. The transparency of actions recorded in the BC and their details locally on the requester and the mitigation on logs is useful in cases where a mitigation service does not have its effectiveness proven by the uploaded proof.

6.3.2 Discussion of Use Cases Usability

The interactive dashboard facilitates the visualization and management of a collaborative defense by a network operator, and enables the management of these events. The bloss-dashboard includes features to visualize the status of mitigation requests from the perspective of the mitigator as well as the requester.

Tasks involving the detection and mitigation of a DDoS attack are not trivial, hence, the analysis in a collaborative environment grows in complexity. The challenge of this dashboard here, however, is to reduce the complexity of information provided to a network operator to support decision-making without adding overhead of unnecessary information. In the presented use case, a new alarm is displayed to and evaluated by the human operator. An alarm can either be classified as a valid threat and whether mitigation should be requested, or the alarm can be ignored. The complexity of the this bloss-dashboard is minimal and based on the known layout from software development (*e.g.*, the Kanban board [173]), leading to a low entry barrier of understanding the user interface. Also, the bloss-dashboard remains predictable as the main layout does not change and empty states (*e.g.*, no new MREQ) indicate where new elements have to appear in the layout.

6.4 Off-chain Signaling Latency

Attack information posted to the BC is not directly stored on the BC due to limited block sizes. Thus, IPFS (InterPlanetary File System) [17] is used as a decentralized and scalable storage solution to hold attack information. Each AS running the BloSS also maintains an IPFS node to enable 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 BC. 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 BC.

Two important aspects concerning this thesis challenges are relevant at the evaluation of the off-chain signaling: (1) confidentiality and integrity of information, and (2) performance. While is essential to (1) maintain the secrecy and integrity of data shared between a *T* and *M*, it is also important (2) to have such data being exchanged in a timely manner. Local and global experiments were conducted to evaluate the signaling delay of BloSS. Node configuration for both experiments are described in Section 6.4.1. First, local experiments were performed at the University of Zürich (UZH) network to tune BloSS configurations and estimate its performance in a controlled environment. These experiments are described in Section 6.4.2. Second, global experiments were performed to assess the performance of BloSS in a geographically distributed and heterogeneous environment. Section 6.4.3 describes these experiments.

6.4.1 Configuration

Instances used to perform the BloSS measurements are described in Table 6.3. Local instances were configured with different Virtual CPU (vCPU) and RAM capacities ranging from 1 to 12 vCPUs and 1 GB to 32 GB of RAM. Instances marked with an asterisk processed parallel workload from other services, such as Graphical User Interfaces (GUI).

Table 6.3: Configuration of Instances Used During the Experiments

Provider	Local	Local	Contabo	Amazon EC2	Amazon EC2	Microsoft Azure	Microsoft Azure	DigitalOcean	DigitalOcean
Type	UZH IfI*	UZH Irchel*	VPS M*	t2.micro	t2.small	B1S	B1MS	small	medium
vCPU	4	12	2	1	1	1	1	1	2
RAM	4 GB	32 GB	6 GB	1 GB	2 GB	1 GB	2 GB	1 GB	2 GB

Table 6.4: Round Trip Time (RTT) Between Deployed Instances [ms]

From \ To	Australia	Brazil	Germany	USA	Switzerland
Australia	-	344	315	248	351
Brazil (AWS)	337	-	229	130	220
Brazil (Azure)	366	6	224	116	204
Germany	333	227	-	103	21
Asia	197	326	279	227	260
Japan	153	266	334	171	170
USA	247	128	104	-	96
Switzerland	347	221	20	94	-

Local instances (UZH IfI and UZH Irchel) were configured with more processing power (vCPUs) and RAM than other nodes due to the availability of resources in the UZH infrastructure. However, nodes used in the global experiment, such as AWS, Azure, and DigitalOcean, were configured with similar specifications (1 GB of RAM for Amazon t2.micro, Microsoft Azure B1S, and DigitalOcean small instances). However, as observed during the experiments, due to low RAM in experimental runs instances were adjusted to 2 GB of RAM. Furthermore, while AWS and Azure were configured with 1 vCPU, DigitalOcean was configured with 2 vCPUs and thus, being able to handle more processes simultaneously. It is important to note that a vCPU is not comparable to a hardware processor and describes the claim of using a physical core for a specific time. Thus, a so-called CPU-credit is used for regulating the CPU usage in these instances. Also, it is important to note that real traffic was not generated in the experiments. Thus, different files with random IPv4 addresses were produced, and requests were submitted to the BC simulating the signaling of an attack. Figure 6.6 presents the distribution of the time required for this process during the experiments.

6.4.2 Local Experiments

Local experiments measured the performance of BloSS to exchange black-listed addresses with different file sizes. Six instances were deployed inside the UZH network: two at the Institute of Informatics (IfI) and the remainder at the UZH Irchel campus. The distance between these instances is 3 km, and the available bandwidth was 200 Mbit/s.

Located at the Institute of Informatics are two local machines attached to the network over Wireless LAN. At the server room four instances were running. They are virtualized using LXD (Linux Containers) and connected to each other over GBit Ethernet. Figure 6.4 depicts the elapsed time for a transaction inside the UZH in four experiments. For each test, 100 measurements (blue dots) were performed, and the file size varied from 10 kB to 10 MB. The red line represents the median value of the corresponding experiment set. Also, results do not account for the constant 15 seconds necessary to create a block in the Ethereum Rinkeby BC.

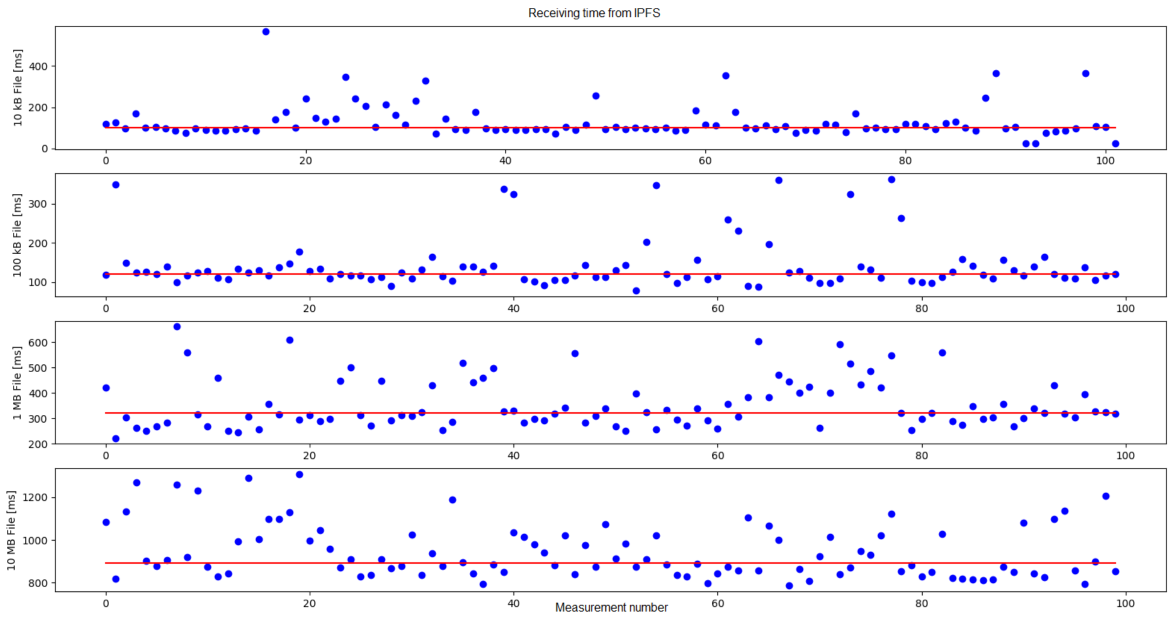


Figure 6.4: Elapsed Time for a Transaction Within the UZH Network With Different File Sizes. Bottom to Top: 10 MB, 1 MB, 100 kB, 10 kB.

The average to perform the signaling of a ten kB file was around 100 ms (first chart on top), and for a 100 kB file is 120 ms (second chart on top). For a 1 MB file, the delay was approximately 320 ms. Thus, a baseline time of 75 ms (processing of addresses) and an additional 25 ms per 100 kB could be observed to complete the signaling of an attack.

Also, it is important to note that the maximum block-size defined in IPFS is 1 MB. Thus, in the case of large-scale attacks where a potential list of blacklisted addresses is above 1 MB a file is segmented into different blocks, which increases the signaling time. During local experiments, a file with 10 MB was created to evaluate the impact of such segmentation. Using a traditional file transfer method (*wget*), it was observed an average delay of 3,200 ms (10 times 320 ms) to complete. For instance, an address list of 10 MB needs to be divided into ten blocks of 1MB, using traditional methods (*e.g.*, HTTP), would take 3200 ms (10 times 320 ms) to complete.

However, IPFS took approximately 900 ms due to the decentralization throughout multiple instances using IPFS. An additional experiment included ten files of 1 MB size simultaneously to IPFS, submitting the hash (generated when the blacklist is added to IPFS) value to the same Smart Contract (SC), and retrieving this file on the mitigator instance. The measured time was 950 ms on average, which is slightly above the transmission time of a regular 10 MB file (900 ms).

Finally, to upload a hash value to the SC and broadcast the transaction to the Ethereum BC was, in general, required between 40 ms and 100 ms. During local experiments, IPFS required between 70 and 140 ms to publish the file, which includes the processes of copying addresses file locally to the IPFS directory, calculating the hash value of the file, splitting it into blocks, and updating the hash table. On the sending side, between 40 ms and 100 ms are required to upload the IPFS hash value to the SC and broadcast the transaction to the Ethereum BC. In local experiments, IPFS required between 70 and 140 ms to publish the file, which includes the processes of copying addresses file locally to the IPFS directory, calculating the hash value of the file, splitting into blocks and updating the hash table.

6.4.3 Global Experiments

To evaluate BloSS in a geographically distributed environment, 8 instances in different countries were deployed to measure the data transfer time among themselves. Table 6.5 details the specification of such instances.

Table 6.5: Specifications of Instances Used for the Geographically Distributed Evaluation

Location	Australia (AU)	Brazil (BR1)	Brazil (BR2)	Germany (DE)	Singapore (SG)	Japan (JP)	USA (US)	Switzerland (CH)
Type	AWS t2.small	Azure B1MS	AWS t2.small	Contabo VPS M	Azure B1MS	Azure B1MS	DigitalOcean Medium	Local
Bandwidth	350 Mbit/s	225 Mbit/s	180 Mbit/s	90 Mbit/s	400 Mbit/s	180 Mbit/s	170 Mbit/s	310 Mbit/s

Two instances were located in Europe, Switzerland, and Germany. In Asia, two other instances were deployed in Singapore and Japan. Three separate instances were situated in America, one in the USA, close to New York, and two instances were located in Brasil near São Paulo. At the time of the experiment, these main cloud providers do not provide instances in Africa. Finally, instances were hosted by different cloud providers (AWS, Azure, DigitalOcean, and Contabo) and configured with similar compute and network capacities (*cf.*, Figure 6.5).

Bandwidth was measured using the package `speedtest-cli` from the Ubuntu repository and measurements performed based on the instance with the lowest latency from the list of `speedtest.net`. Values represent the approximated minimum values upload and download times in a batch of 10 measurements. The lowest bandwidth measured is from Contabo with 90 Mbit/s, which restricts

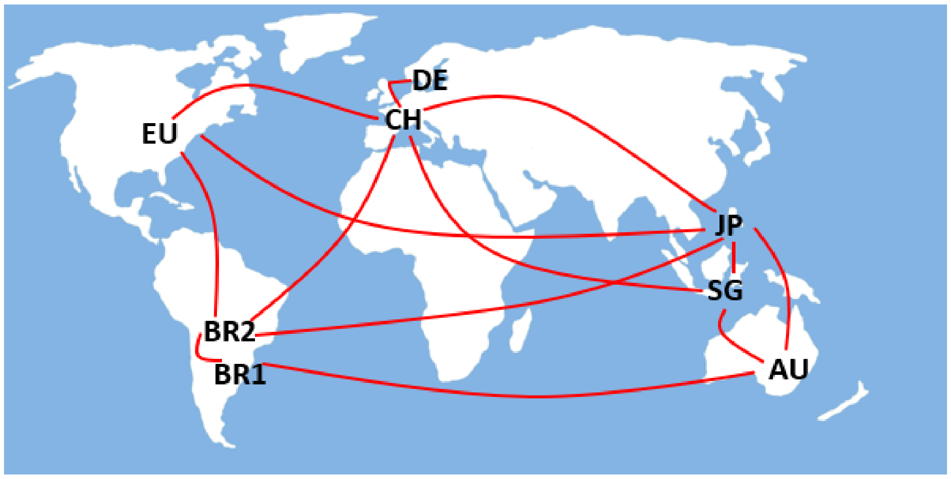


Figure 6.5: VM Instances Used in the Experiment Configured as in Table 6.5

available bandwidth to a maximum of 100 Mbit/s. All other instances were configured with at least a bandwidth of 170 Mbit/s. Differences in available bandwidth were not significant for lists of addresses of under 200 kB in the experiments.

The Round Trip Time (RTT) provides an estimated lower bound for the time needed to transfer a file between two instances. The measurements presented in Table 6.4 represents an average value of 100 requests at the interval of 10 seconds. It is worth mentioning that Microsoft Azure does not allow incoming ICMP traffic. Therefore, ping requests could not be answered by instances deployed in this cloud provider. The measurements shows approximately symmetric results, *i.e.*, the RTT time from Brazil to Switzerland is approximately the same as from Switzerland to Brazil.

Signaling Latency

The latency between a request of mitigation service (signaling) and the mitigation is an important metric in a cooperative defense system. Experiments consisted in executing the sequence of steps in Figure 5.2 to evaluate the delay from the beginning of an attack until the attack has been mitigated as well as the CPU usage of each BloSS instance throughout the entire mitigation.

More than 95% of the measured values are between the 15 to 35 seconds range. The lowest signaling time measured was 14.1 seconds, while the highest was 3 minutes. Considering the measured values and that a block is created at every 15 seconds in the Ethereum Rinkeby BC, most transactions were mined in the first or second block after a transaction was submitted. Observing long-lasting transactions, it was noticed that failures on receiving instances caused the transactions to be delayed

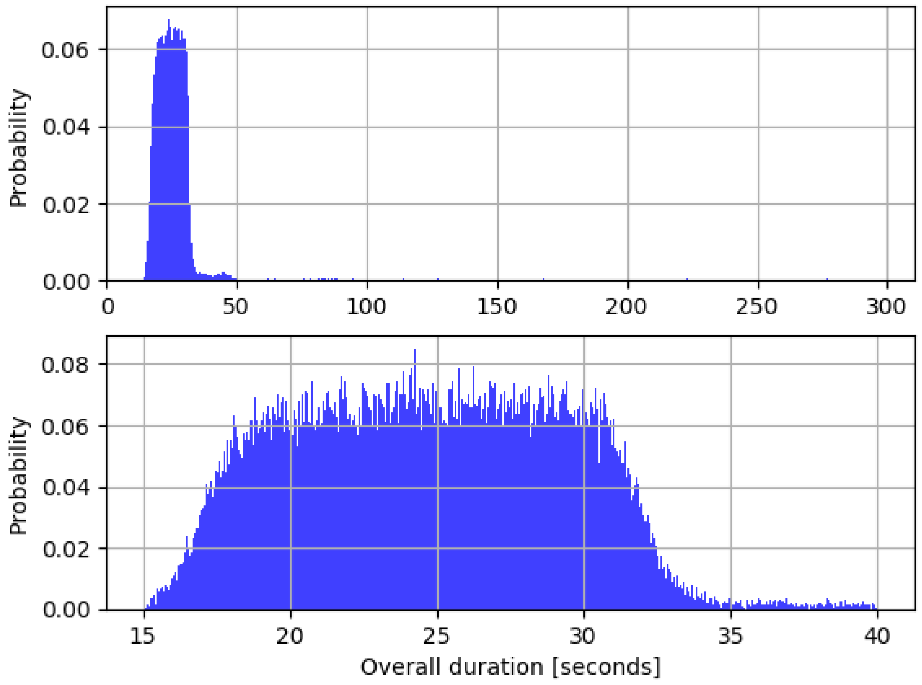
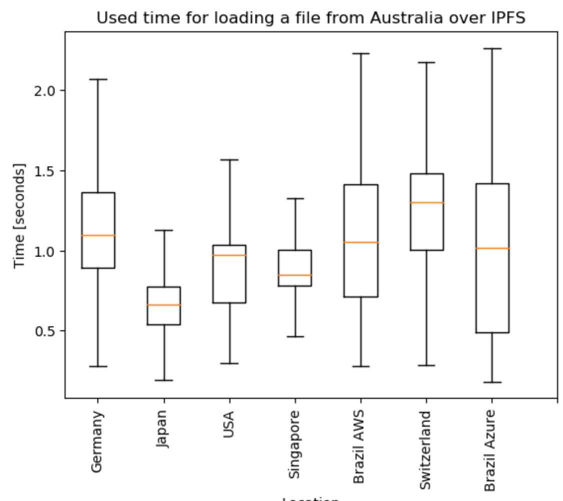


Figure 6.6: Probability Distribution of Overall Elapsed Time (Upper Part), and Range Between 15 and 40 Seconds Zoomed (Bottom Part) [197]

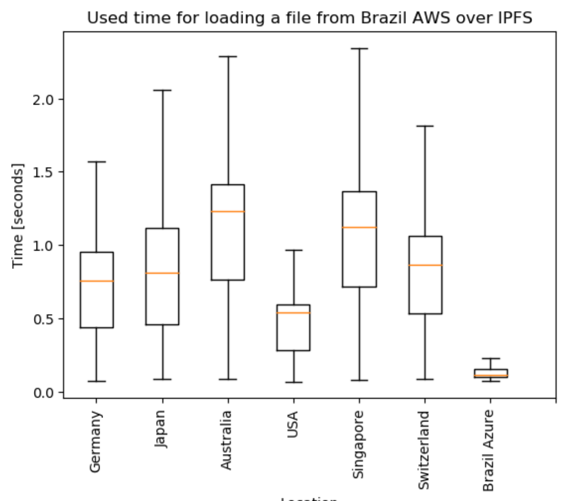
mostly due chain reorganization. A measurement of more than 300 seconds is not possible because the value stored in the contract would be overwritten by the subsequent transaction of instances still reporting addresses. Other errors were encountered due to the restart of the operating system and the instance after adding the instance in Brazil on the Azure Cloud.

Figures 6.7 and 6.8 depicts the results of the measurements of the time required for successfully transferring a file from a sender to receivers. Every file in the experiment contains 10,000 randomly generated blacklisted IPv4-addresses, resulting in a file size of 173 kB. Files were published by BloSS-enabled instances, with an attacked network assigned and other BloSS instances retrieving the file to support the mitigation of the attack.

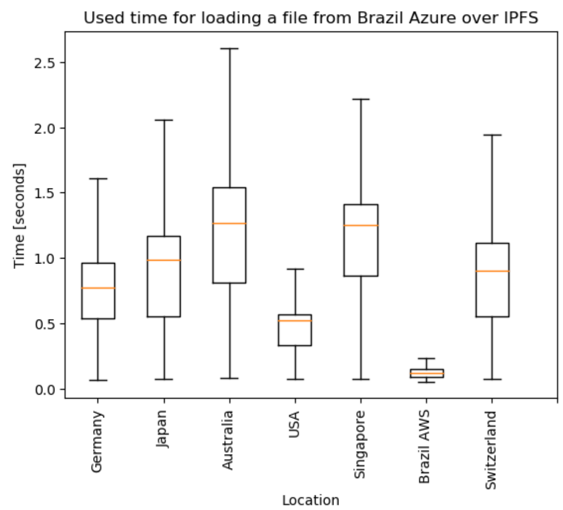
BloSS instances into a SC deployed in the Ethereum Rinkeby BC and all other instances listening to the contract can retrieve the signaled file. This represents a BloSS-enabled instance, with an attacked network assigned to as the publisher of the file and the other instances of BloSS retrieving the file to support the mitigation of the attack. In general, proportions between transfer times are distributed as shown in RTT measurements Table 6.4. Transmission times of near-located instances is, in general, lower than between farther distanced instances, but not symmetric in all cases.



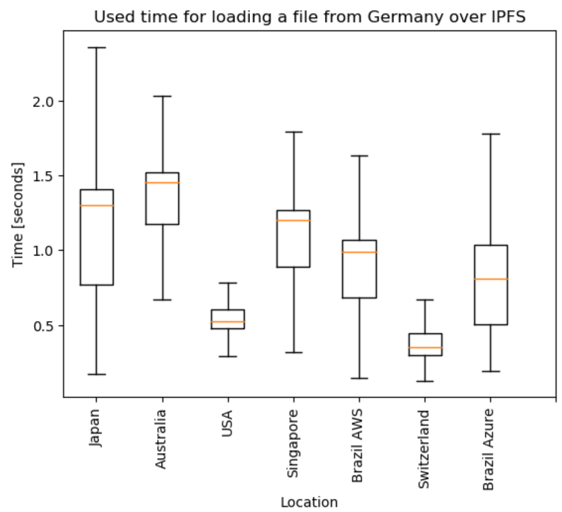
(a)



(b)

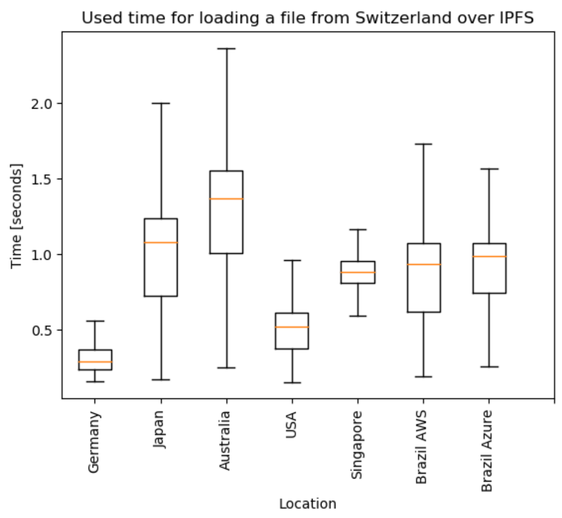


(c)

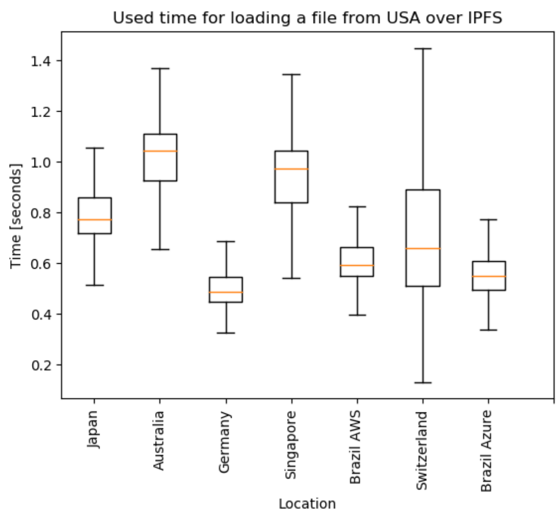


(d)

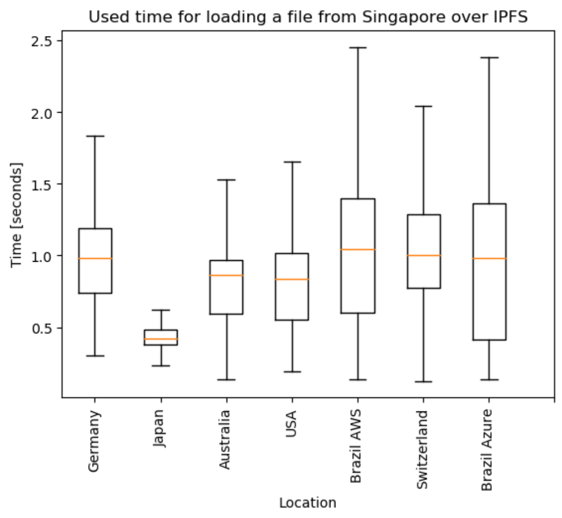
Figure 6.7: (a) Australia, (b) Brazil AWS, (c) Brazil Azure, and (d) Germany



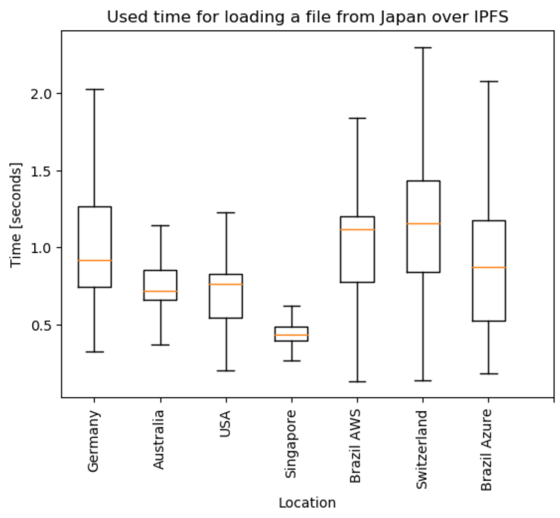
(a)



(b)



(c)



(d)

Figure 6.8: (a) Switzerland, (b) USA, (c) Singapore, and (d) Japan

Differences in observed transfer times mostly relates to the geographical proximity of instances and the availability of content hosted in the IPFS nodes. For example, during the experiments some instances closely located (*e.g.,* , Switzerland-Germany, Brazil-Brazil) resulted - as expected - in low signaling delays in contrast to others. Furthermore, some instances could benefit from a direct link connection between the same cloud provider (*e.g.,* , Azure instances Brazil-Japan).

A significant variation in data transference times (delay between instances) was observed. The maximum delay observed was 2.6 seconds while the average values by instance regardless varied between 0.9 and 1.2 seconds. The fastest transmission between instances was in Brazil where, due to their proximity, the transference time was below 200 ms. In general, to retrieve signaled addresses represents the overall time an instance uses to receive a block from the BC, retrieve the file from the DHT (Distributed Hash Table) and its own operations to decrypt the file.

Large Dataset

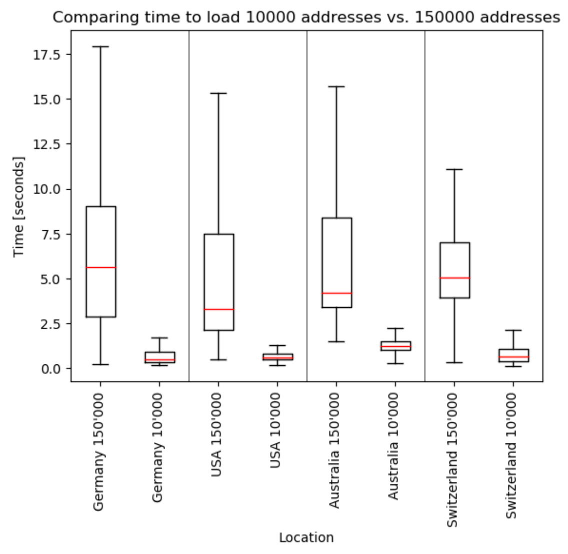
In large attack occasions, such as the DDoS attack on the French provider OVH in 2016 where over 150'000 IoT devices were involved [174], larger files have to be handled. Figure 6.9 (a) shows the measured values of four instances in comparison of 150'000 to 10'000 IP addresses as in the previously mentioned tests.

Figure 6.9 (a) presents mean values for the larger file, these values are between four and ten times higher for transmitting data 15 times larger. On one hand, files of over 1 MB size delivers the advantages of the decentralized approach of IPFS by splitting these files into multiple hosts. For example, for a blacklist published in Australia, the instance in Switzerland can retrieve the first block from the USA, which already has loaded the block, some blocks from Australia, and other blocks from Germany. This behavior allows to minimize the usage of distant instances and interacts with close-by instances more often.

On the other hand, the spread of measurements is higher. In the best case, blocks are available from a close-by instance and the transfer time will be relatively low. All instances can try to receive the data at the same time from one instance which gets congested at this moment and can serve only two requests simultaneously. In this case, some instances have to wait until the resources of the sender are available. As a result, the used time increases strongly.

HTTP Comparison

To compare the results of IPFS as the data channel to exchange blacklisted addresses measurements using the data transmission over HTTP were performed. Figure 6.9 (b) depicts the time taken to



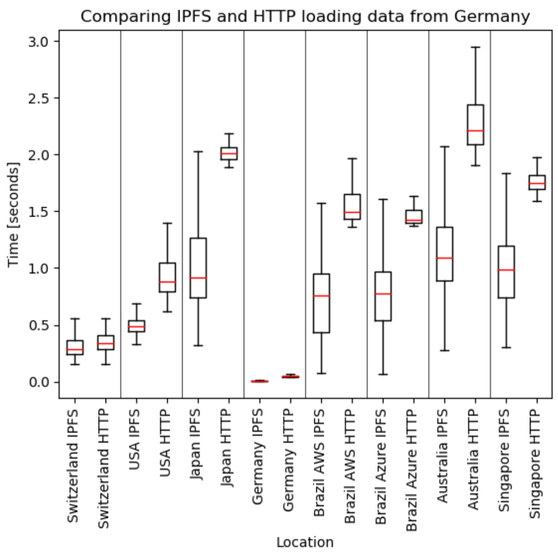


Figure 6.9: Large Dataset; HTTP Comparison.

load the data using HTTP and IPFS. Measurements were performed using the instance in Germany as file provider, in which instances requested the target file 120 times. In general, the transmission over HTTP was slower than over IPFS. The difference between the time taken over HTTP and IPFS is increased by the distance between the two instances. The difference in the spread of the two instances in Brazil is noticeable probably caused by the different overseas cable of the providers.

6.4.4 Discussion of Off-chain Performance

While local experimentation helped to create the basis of the system and reveal first aspects of configuration, the global evaluation exposed the system to real-world problems. During the experiments, it can be seen that the off-chain tasks involving file management, have a more significant influence on transfer latency than the steps in the BC. This influence varies according to the file size and distance between IPFS gateways, being able to observe a transfer time increased to 20 seconds in the global experiments with large datasets (150'000 IPv4 addresses) in contrast to the payload of 10,000 IPv4 addresses.

Also, during the local experiments, it was possible to transfer files up to 4 GB size and ten files simultaneously without failures. However, in the global test with instances running on cloud servers from various providers some issues were found. Firstly, it is necessary to observe that BloSS uses IPFS and the Ethereum BC for signaling attacks. Thus, it is required an available communication channel to send/answer requests during an attack. Secondly, the use of relatively new technologies such as IPFS and BC may result in failures during its operation. For example, IPFS experienced problems of excessive memory consumption in instances running on Amazon EC2 t2.small, Microsoft Azure B1S and DigitalOcean small instances with 1 GB of RAM, as shown in Table 6.3, which needed to be upgraded to Amazon EC2 t2.medium, Microsoft Azure B1MS and DigitalOcean medium instances providing 2 GB of RAM.

The distribution of the used memory (besides the portion allocated to the operating system) is spread in the following proportion: two parts for the IPFS daemon and one part for the Ethereum client with a deviation of up to 15% of this ratio. During block indexing periods, the Ethereum client allocated more memory for this process.

6.5 Reputation Scores

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, incentivizing peers to contribute with relevant information and establishing fairness among peers. BC technology not only offers new possibilities in attack signaling but also emerges as a trustworthy and distributed solution for reputation management. Since reputation is earned in interactions between peers, it can be attached to transactions preventing arbitrarily manipulations or gamming attempts. This Section presents the evaluation of different customer profiles (T or M) in the cooperative signaling protocol.

6.5.1 Configuration

The simulation experiments comprised two identically equipped Dell XPS machines, with Intel Core i7 (4 Cores, 3.40 GHz) CPUs and 8GiB of RAM each. One machine hosted an Ethereum bootnode and one mining peer. The second machine hosted the second, non-mining `geth` instance and the Node.js simulator script. In addition, an Ethereum network statistics dashboard (eth-netstats) has been deployed for monitoring purposes on this second machine. The simulator uses the Web3 Ethereum JavaScript API and runs in two different modes, the test mode used to verify the correct behavior of the simulated customer strategies and the acceptance mode simulates an environment close to reality.

1. **Test Mode**: Allow customers to complete a mitigation contract with each other only once.

2. **Acceptance Mode**: Simulates continuous DDoS attacks and the creation of mitigation tasks and allows the simulated customers to make decisions based on past reputation values.

6.5.2 Customer Strategies

Tables 6.6 and 6.7 present the numerical analysis of possible strategies, which match their analytical specification in Figure 4.2. These two tables show the state of the simulated world after all customers complete a mitigation contract with each other once. The simulation function takes as an input one mitigator strategy and one target strategy. After the simulation halts, the output analyzed in this evaluation here is the end state of the mitigation tasks in the simulated world (Table 6.6) and the reputation of the customers (Table 6.7). In this way, all 16 possible combinations of customers (4 targets $\times$ 4 mitigators) have been evaluated, to ensure the correct behavior of these customer strategies.

Table 6.6: Task Configurations and End States

Input: Customer Strategy			
Target	**Mitigator**	**Output: End State**	**Task ID**
Uncooperative	Uncooperative	Completed	0
	Lazy		1
	Selfish		2
	Rational		3
Selfish	Uncooperative		4
	Lazy	Started (funds sent)	5
	Selfish	Proof uploaded	6
	Rational	Completed	7
Satisfied	Uncooperative		8
	Lazy		9
	Selfish		10
	Rational		11
Dissatisfied	Uncooperative		12
	Lazy		13
	Selfish		14
	Rational	Rejected	15

Table 6.6 lists all possible combinations of mitigator-target strategies and the mitigation contract ID. Completed tasks are either aborted before payment or paid out successfully. Because selfish and lazy customers never rate, tasks 5 and 6 cannot be completed by either party. Compared to the lazy customer, the selfish one does upload a proof but never rates, being the reason why the end states for task 5 and 6 are different. Task 15 is the typical escalation case, where a dissatisfied target and a rational mitigator would argue about the truthfulness of the proof of service. In this case, no payment is made, and end state is "rejected".

Varying time-windows were chosen for the deadlines, such that they might lead to situations, where customers miss to meet a deadline. For example, the code that simulates a mitigator uploading a proof with the minimum service deadline of three blocks might not be executed fast enough,

Table 6.7: Reputation Values and Total Number of Interactions

Input:	**Output: Ratings**			**Output:**
Customer Strategy	**Positive**	**Neutral**	**Negative**	**Interactions**
Satisfied T	1	3	0	4
Selfish M	1	2	1	4
Rational M	1	2	1	4
Uncooperative T	0	4	0	4
Uncooperative M	0	4	0	4
Selfish T	0	3	1	4
Dissatisfied T	0	3	1	4
Lazy M	0	2	2	4
Total:	**3**	**23**	**6**	**32**

because the block time of the BC increases faster due to other transactions being processed. The service, validation and final rating/escalation deadlines are sampled randomly for each task. Service deadlines are chosen in a range of $[3, 13]$ blocks and validation deadlines in range of $[17, 27]$ blocks. Final rating deadlines are sampled in the range of $[32, 42]$ blocks. Before the new mitigation contract is created, all peers are funded with 10 Ethers each. The contract price was fixed to one Ether for every mitigation task.

Overall, Beta reputation scores for attack targets (Ts) allow identifying a satisfied T, because in comparison with the undesired T strategies it develops the highest average reputation value (*cf.* Figure 6.11). From Figure 6.11 it also becomes evident, that dissatisfied and selfish Ts are downgraded continuously and will eventually no longer receive help from mitigators (Ms), due to their lousy rating. It is not desirable in all circumstances that satisfied Ts crowd out dissatisfied Ts, since this will incentivize Ts to accept also poorly (or even severely) delivered mitigation services. Because of the current design, no third party checks that a rating indeed matches the quality of the delivered service, this problem remains yet unsolved.

After the first few mitigation contracts, M reputation values in Figure 6.11 also allow to clearly distinguish a constant rational M from the lazy M. However, unlike for the Ts, it is difficult to distinguish the rational M from the selfish and uncooperative M. The uncooperative M usually shows low positive and negative reputation, because it aborts mitigation tasks early, never takes any action, thus receives little feedback and its reputation stagnates at 50%, not performing particularly good or bad. Hence, chances of picking an uncooperative M for a transaction is diminished the most by considering raw reputation values, especially the amount of positive and negative ratings. Because both, the selfish and rational M, upload proofs, the only difference between the two strategies is that the selfish M never rates. Selfish M behavior is an irrational strategy, since the selfish M (deliberately or

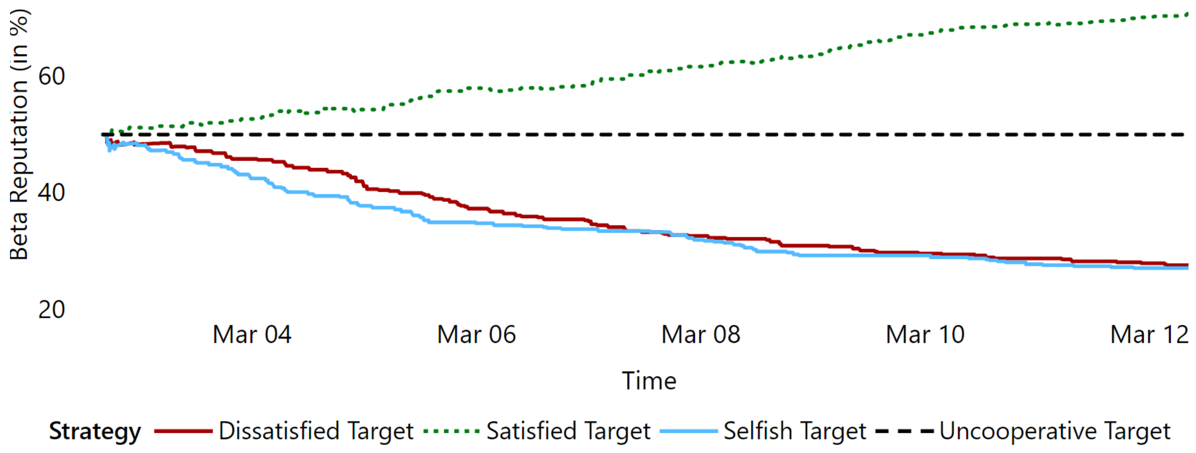


Figure 6.10: Average Beta Reputation for Attack Targets [81, 82]

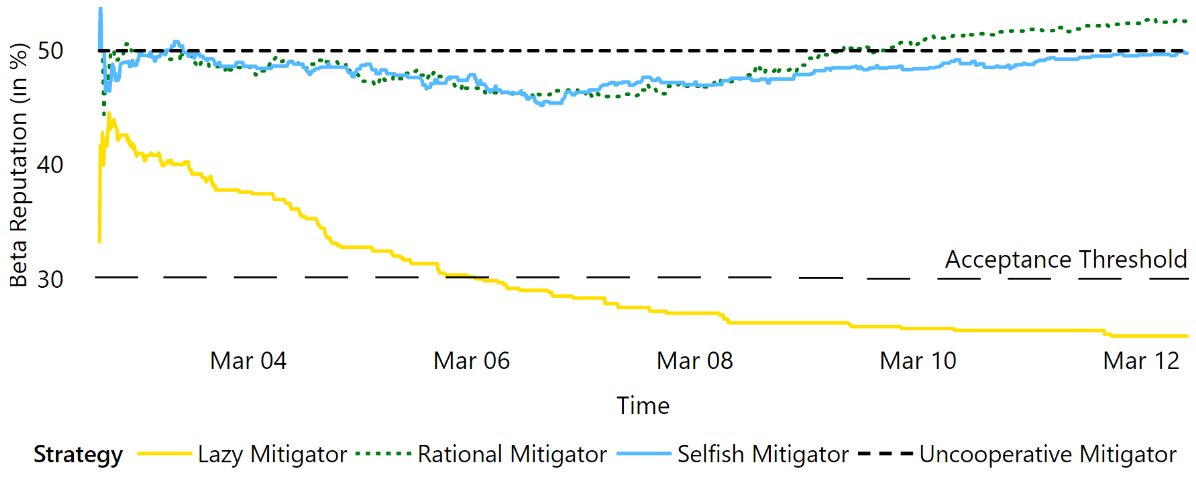


Figure 6.11: Average Beta Reputation for Mitigators [81, 82]

not) deprives itself of payment. Assuming that M rarely forgets to rate services at the end (which is the definition of selfish behavior), one can also assume that there exists more rational than selfish Ms. This leads to the conclusion that chances of picking a rational M compared to a selfish M with the same Beta reputation score are higher, due to the incentive.

6.5.3 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 [81]. Table 6.8 summarizes the analysis, which is followed by a discussion on each type of fraud.

Table 6.8: 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

- **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).

- **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.

6.5.4 Discussion of Reputation Scores and System Robustness

Overall, Beta reputation scores for attack targets (Ts) allow identifying a satisfied T, because in comparison with the undesired T strategies it develops the highest average reputation value. It also becomes evident, that dissatisfied and selfish Ts are downgraded continuously and will eventually no longer receive help from mitigators (Ms), due to their lousy rating. Hence, it is not desirable in all circumstances that satisfied Ts crowd out dissatisfied Ts, since this will incentivize Ts to accept also poorly (or even severely) delivered mitigation services. Because of the current design, no third party checks that a rating indeed matches the quality of the delivered service, this problem remains yet unsolved.

After the first few mitigation contracts, M reputation values in Figure 6.11 also allow to clearly distinguish a constant rational M from the lazy M. As depicted in the Figure 6.11, the rational M would always upload a proof as soon as the mitigation service is complete whereas a lazy M would eventually miss a deadline depending on the random waiting time configured in the experiment to simulate their lazy behavior. However, unlike for the Ts, it is difficult to distinguish the rational M from the selfish and uncooperative M. The uncooperative M usually shows very low positive and negative reputation, because it aborts mitigation tasks early, never takes any action, thus receives little feedback and its reputation stagnates at 50%, not performing particularly good or bad. Hence, chances of picking an uncooperative M for a transaction is diminished the most by considering raw reputation values, especially the amount of positive and negative ratings.

Because both, the selfish and rational M, upload proofs, the only difference between the two strategies is that the selfish M never rates. Selfish M behavior is an irrational strategy, since the selfish M (deliberately or not) deprives itself of payment. Assuming that M rarely forgets to rate services at the end (which is the definition of selfish behavior), one can also assume that there exists more rational than selfish Ms. This leads to the conclusion that chances of picking a rational M compared to a selfish M with the same Beta reputation score are higher, due to the incentive.

Lastly, the reputation system prevents Sybil and collusion attacks by mapping customer accounts to real-world identities, preventing customers from creating several identities to manipulate reputation scores. However, ballot stuffing and bad-mouthing are not prevented, but discouraged due to the cost to deploy a mitigation contract only to manipulate reputation scores. Hence, a customer can boost its reputation by creating mitigation contracts with itself. As long as the system is not deployed on a public ledger, customers can be prevented from maintaining multiple account pseudonyms on the BC and transacting between them to inflate reputation (Sybil attack).

6.6 Cooperative Signaling Protocol Latency

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 [198], this evaluation discloses the global evaluation results.

6.6.1 Configuration

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 6.9 lists Round Trip Time (RTT) results executed on AWS instances in Zürich, 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.

Table 6.9: 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	-

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*.

6.6.2 Simulation Experiments

The first step in the prototyping of the Cooperative Signaling Protocol is to experiment it in a simulated environment. A simulation based on Ganache [235] allows for rapid experimentation in terms of correctness, performance, and integration before deploying the contract in a real BC environment. Especially the *gas usage* and *performance* results, independently from the block times, are considered in the evaluation on Ganache. The reason for disconsidering block times is that, in Ganache, block times are reduced or even eliminated for rapid prototyping reasons. Thus, waiting times to simulate a lazy or selfish actors were subtracted to achieve comparable results within the different test cases and to measure only the processing times in this first step.

Also, registering a mitigator and retrieving the mitigator's address from the Register follows the same approach in any of the scenarios. However, when a mitigator is found, a second search for the same mitigator will not return a different address of the previous and thus, the registration process as well as the searching for a specific mitigator by a name is executed only once after deploying the Signaling Protocol.

Table 6.10: Ganache Processing Times [s] for Protocol

Scenario	Average Processing Time [s]	Standard Deviation [s]
1	0.312	0.027
2	0.277	0.020
3	0.417	0.028
4	0.358	0.028
5	0.398	0.025
6	0.377	0.025
7	0.335	0.020
8	0.366	0.033
9	0.404	0.022
10	0.426	0.037
11	0.405	0.022
Average	0.370	0.026

The overall average processing time of each scenario with $n = 20$ is approximately 0.37 seconds, and the average standard deviation is 0.026 seconds, which is achievable by eliminating block times on

Ganache. These results can not be reproduced by using a testnet (*e.g.*, , Rinkeby) since its block times are dependent on the PoA algorithm and have an average duration of 15 seconds. Table 6.10 shows that without including block times, the average processing time for the Signaling Protocol would be close to 0.37 seconds.

6.6.3 Local Experiments

By executing the performance tests on a public testnet *e.g.*, Rinkeby, real-world block times are included. Not only does Infura serve as a proxy to the Rinkeby BC and, therefore, allows testing with defined accounts through the Infura Rinkeby endpoint ID, it also enables to execute the same test scenario scripts that were used for the Ganache evaluation. Executing the same scripts leads to comparable results and a more reliable testing cycle. By only adding the option to the Truffle command for using the Rinkeby network, truffle automatically starts deploying the Register and Signaling Protocol with the predefined accounts specified in the *truffle-config.js* file.

Running the performance tests locally from the Infura node on the same computer returns the same gas usages as the described registration gas usages in Table 6.14. This also means that the transaction fee, while setting the same gas price on Ganache and Rinkeby, results in the same sum. Executing the performance tests, however, showed that for each changing state one block time is to be awaited for the state to be changed on the BC, allowing the Cooperative Signaling to proceed to a final *endstate*.

Table 6.11: Local Rinkeby Processing Times [s] for Protocol

Scenario	Average Processing Time [s]	Standard Deviation [s]
1	90.270	0.542
2	88.811	0.855
3	88.188	0.889
4	106.304	0.471
5	89.500	0.492
6	102.518	0.657
7	118.151	0.528
8	105.062	0.863
9	87.842	0.722
10	105.418	0.601
11	89.471	0.629
Average	97.412	0.659

Since the average block time on Rinkeby is defined as 15 seconds and each outcome scenario of the cooperative signaling protocol includes six to eight blocks, the time to finish each scenario can be estimated by multiplying the number of blocks with the average block time. However, by running the performance tests locally on Rinkeby, the average processing time was shorter than the estimated times in 7 out of 11 scenarios. On average the overall processing signaling time is 97.412 seconds and the average standard deviation testing the scenarios on Rinkeby via Infura is 0.659 seconds.

6.6.4 Global Experiments

A synchronized node can utilize Geth in order to interact with a deployed SC. Instead of relying on one full node that is offered by Infura with two accounts, the tests with the Geth client are executed on two synchronized nodes in Zürich and Ohio. A control condition is used between the nodes in Tokyo and São Paulo such that the results from Zürich-Ohio can be compared with either a private BC, a proxy instead of a full node and finally Tokyo-São Paulo (*cf.*, Figure 6.12 shows AWS instances deployed worldwide). The focus on these global tests is to measure the *gas usage* and the *performance*, and whether the Register and Signaling Protocol are able to be used by more than one node. Moreover, in order to test the behavior on two instances, a different script was implemented for a pseudo-real target to simulate its information as well as the mitigator's side which is to be registered.

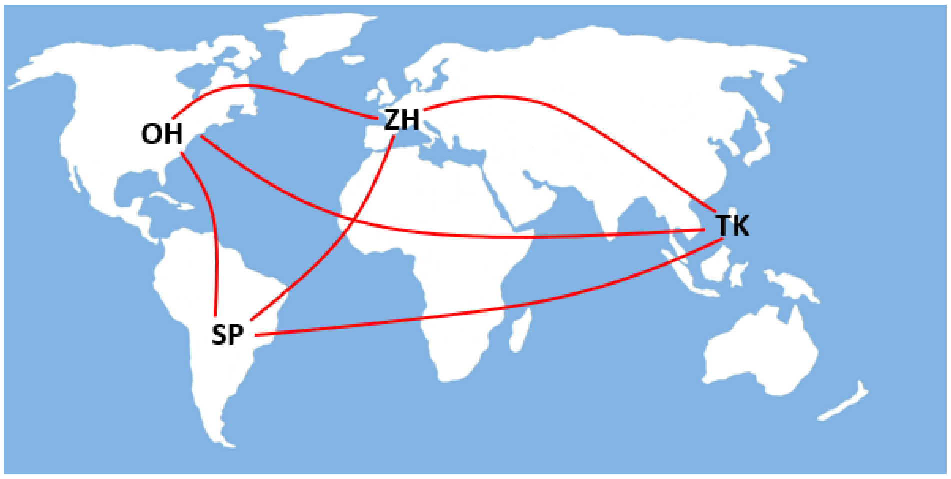


Figure 6.12: AWS Instances Used in the Experiment. Acronyms: OH - Ohio, TK - Tokyo, SP - São Paulo, ZH - Zürich

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.

Table 6.12: 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

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 6.12). Since the control condition has been tested and 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 6.12 and 6.13, while the nodes were not synchronized at all times due to timeouts, *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.

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 6.9 the 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.

Table 6.13: 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

Figure 6.13 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.

6.6.5 Deployment and Operation Costs

A critical factor to retrieve from the execution of all scenarios, besides the *performance*, is the Ethereum gas cost. As every transaction or change of a contract state costs gas, it is important to evaluate the *gas usage* of the scenarios in order to determine whether the costs are feasible or not. Note that in order to calculate the total cost of a transaction, *gas used* is multiplied with the price per Gwei. The *gas price* per Gwei is adjustable and, therefore, offers slower or faster transaction speed *i.e.*, , a lower or higher probability to be added to the next block, correlating positively with the gas price.

$$TransactionFee[\$] = \frac{GasUsed \times GasCost \times ETHPrice}{10^9} \tag{6.1}$$

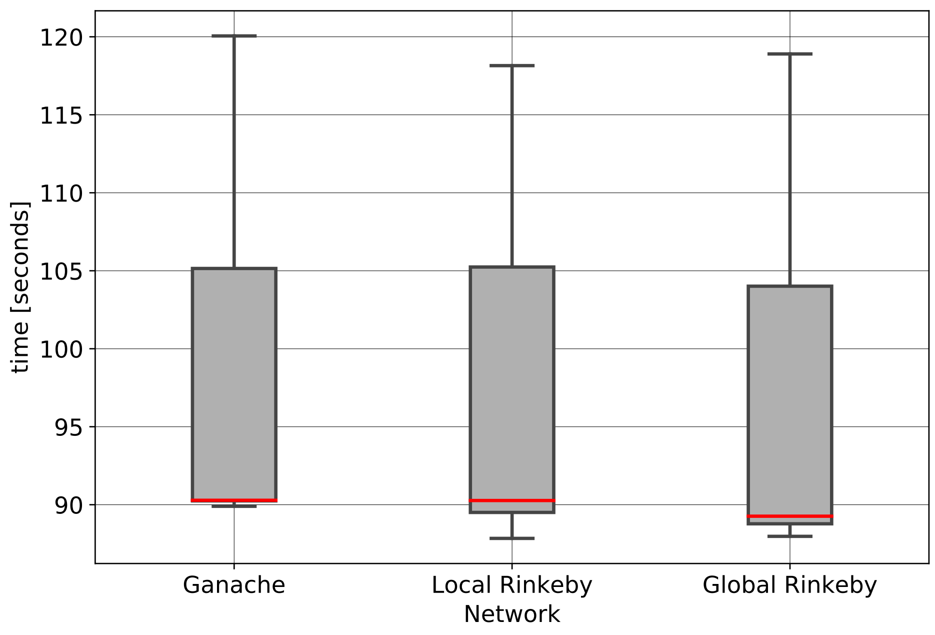


Figure 6.13: Comparison of Global Average Processing Times [s] on Ganache and Local Rinkeby

Since the gas used does not directly show the transaction fee costs in fiat currency *e.g.*, US dollar, the formula in 6.1 can be used to determine the transaction fee of the deployment or the interactions afterwards.

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 6.14.

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 6.15 with a conversion rate of 216.00 US$/Ether).

Table 6.14: 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

Table 6.15: Total Average Global Rinkeby Costs 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 [Gwei]	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

*Conversion [Gwei]/[US$] considering 216 US$ per Ether (August 26, 2019).

6.6.6 Discussion of Signaling Performance

The average processing times showed that under perfect conditions in the Simulated environment *i.e.*, , close to zero block times and no network latency, the average processing time for the Signaling Protocol execution takes 0.498 seconds. Local test scenarios using the global Rinkeby setting showed, however, that on average across all scenarios, 96.943 seconds should be expected for the Signaling Protocol to process in a realistic scenario. This difference of 96.444 seconds results mostly from the average block time of 15 seconds which is added to the total processing time, in every change of state. In addition to the main factor of the average block time, the latency between Zürich and Ohio also adds 0.1192 seconds to the total processing time of the Signaling Protocol.

Averaging every scenario over the tested results shows, by using Ganache with block times set to 15 seconds as well as the test results from Rinkeby in Section 6.6.3 and 6.6.4, that the results are indeed similar to each other. While the processing tests on the private local BC Ganache were deviating 0.37

seconds on average, local Rinkeby showed 0.81 seconds and global Rinkeby 1.172 seconds for the same task. In addition to the standard deviation, the block time of Rinkeby is averaging 15 seconds, which also leads to processing times in the local and global tests to be slightly shifted towards shorter periods of time. Every test in the Evaluation used for the comparison represents worst case scenarios for the cooperative signaling process, since only full block times are considered. 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.

6.7 Smart Contract's Vulnerabilities

The analysis of vulnerabilities had as goal to conduct an assessment of BloSS contracts (*cf.*, Appendix B) by applying five tools for automated security audit, and a manual inspection of BloSS contracts. Subsection 6.7.1 describe the conditions and tools used for the automated analysis. Subsection 6.7.2 presents the outputs of the analysis and a comparison between the tools. Lastly, Subsection 6.7.3 discuss the findings on vulnerabilities.

6.7.1 Configuration

Five automated tools were used for an automated security audit of BloSS SCs (*cf.*, description in Chapter 2, Section 2.7): Mythril, Securify, Securify, Remix, and Manticore. It is important to note that not all listed tools were used in the evaluation due to their own outdated dependencies (in the case of Oyente and MAIAN), implying that contracts could not be compiled in their environment. For example, the latest Solidity version supported by Oyente was 0.4.19, while most contracts in BloSS require 0.5.8. Oyente's Web GUI had even older dependencies, where the last supported Solidity version is 0.4.17. In addition, this analysis was complemented with a manual code check based on the list of vulnerabilities for SCs listed in Appendix E.

6.7.2 Analysis of BloSS Smart Contracts

BloSS was analyzed in terms of general BC vulnerabilities. Due to irreversible nature of blockchain's transactions, deployed SCs and transactions become immutable after deployment. This property brings both advantages and disadvantages. On one side, attackers cannot modify contracts of tamper with transactions (*e.g.*, revoke the payment for a mitigation action). On the other side, SC developers are not able to change contracts if they are already deployed. Thus, to fix a bug or improve a contract, developers have to terminate it and create a new one. Thus, testing and security audit play a crucial role in the development of smart contracts [190].

Another important BC property is sequential execution. The order of SCs execution is determined by a consensus mechanism and is the same for all users. As a result, only a limited number of contracts can be executed per second. This would lead to a performance bottleneck and allowing attackers to stall the network by using a contract, which would take a lot of time to execute. However, BloSS deployment aims at a permissioned network composed by limited and pre-trusted members, thus such vulnerability would not affect its scalability. Lastly, code complexity and human errors are typical problems for SCs. The technology is still relatively new, and many developers do not have sufficient knowledge and experience in this area yet. Table 6.16 demonstrates the number of their findings for different smart contracts in BloSS.

Table 6.16: Number of Findings per Security Audit Tool

Contract	**Mythril**	**Securify**	**Securify 2.0**	**Remix**	**Manticore**
Enums	-	-	-	-	-
Migrations	1	2	8	1	-
Register	2	5	9	5	N/A
Protocol	3	18	N/A	64	N/A
Total	**6**	**25**	**17**	**70**	**-**

*"N/A" = Not Available, "-" no findings

The cooperative signaling protocol (termed Protocol) provides the main functionality of BloSS (*i.e.*, regulates the interaction between a target and a mitigator), and is, therefore, the most complex SC. This explains why the tools found the highest number of vulnerabilities and bugs. However, two tools were not able to perform the analysis of Protocol. Securify 2.0 successfully performed the analysis of Register contract after Enums contract was added directly to the code. Found vulnerabilities were categorized in Table 6.17 and discussed in Subsection 6.7.3.

6.7.3 Analysis of Smart Contract Security Tools

It is important to note that our definition of a vulnerability type does not always completely match the one implemented in the tools (as described in Section 2.7 of Chapter 2, the novelty of the topic and different BC environments result in distinct view by authors on vulnerabilities and their definitions).

Table 6.17: Classification of the Security Audit Tools Findings

Vulnerability	Mythril	Securify	Securify 2.0	Remix	Manticore
Reentrancy	-	2	-	-	-
Transaction ordering	-	2	-	-	-
Block timestamp dependency	1	-	-	7	-
Exception handling	-	-	1	34	-
Unrestricted Write	2	14	3	-	-
Non-validated arguments		1	4	-	-
Greedy contract	-	1	-	-	-
Overspent gas	-	-	-	25	-
External call	3	3	-	-	-
Erroneous visibility	-	-	6	-	-
Division	-	2	-	-	-
Uninitialized state variable	-	-	1	-	-
Solidity naming convention violation	-	-	1	-	-
Complex Solidity version pragma statement	-	-	1	-	-
Variables with similar names	-	-	-	1	-
Bytes and string length	-	-	-	1	-
Tool's internal error during the audit	-	-	-	2	-
Total	**6**	**25**	**17**	**70**	**0**

For example, Mythril identifies two external call vulnerabilities in the line 42 of the Protocol (*cf.*, Listing 6.1):

```
reg = Register(RegisterAddress);
// cast  mitigator  address  from  address  to  payable  address
Mitigator = address(uint160(reg.getMitigator(address(this), _name)));
```

Listing 6.1: Snippet Protocol Contract

As per Mythril's findings report, the first vulnerability occurs because the output of the external call is read. The second is caused by writing this output to a variable. However, according to the definition of external call (*cf.*, Appendix E), they refer to the same external call and, therefore, identify only one external call vulnerability in this snippet. Besides, the last seven vulnerabilities in the table do not match any of vulnerability types defined in Appendix E. Securify allows to check if a contract used division in any calculations. Its second version examines state variables, the compliance with Solidity naming convention and Solidity compiler's version statements. Finally, Remix reviews variable names and the usage of bytes and string length. Moreover, there are two messages about an internal error included into the security report for Protocol contract.

After the findings of the security audit tools were classified, a manual analysis and validated findings against true and false positives was performed. As a result, it was identified overall 85 false positives. Almost half of them belong to the exception handling vulnerability type. In particular, 34 findings by Remix contain a suggestion to consider using *assert(x)* instead of *require(x)*. However, the main purpose of using *require(x)* in BloSS is to check that certain conditions are met during the execution and not to handle internal errors. Thus, according to Solidity documentation [67], the choice of the function is correct.

Securify 2.0 identified one exception handling vulnerability, which was considered to be a false positive. According to the tool's output, the return value in the line 22 in Migrations needs to be explicitly checked for an error (*cf.*, Listing 6.2). However, no special error handling is required in this case because *setCompleted()* only assigns an argument to a public variable and is not error prone.

```
function upgrade(address new_address) public restricted {
  Migrations upgraded = Migrations(new_address);
  upgraded.setCompleted(last_completed_migration);
}
```

Listing 6.2: Snippet Migrations Contract

In the same line, Mythril and Securify identified an external call. However, this is a recursive call of Migration contract by itself. This contract does not contain any critical functions or data. For this reason, this finding is not relevant from a security point of view. Another vulnerability with a large number of false positives is gas overspent (identified only by Remix). In overall 25 cases, Remix suggests adding a gas requirement limit to functions. As neither of the functions contain gas costly patterns it is unnecessary to add gas limits to them.

There are also eight cases of unrestricted write identified by Securify which are considered to be false positives. For example, Securify claims that there are no write restrictions for the line 55 in Protocol. However, as the listing 6.3 demonstrates, only a target can call this function and, therefore, perform these write operations. Besides, it is only possible to do this in three states of the process and only if a mitigator is selected.

```
//Line 55 of the Protocol Contract
owner.transfer(address(this).balance);
```

Listing 6.3: Transfer of Funds in Protocol Contract

Besides, Securify identified two cases of division in Protocol and suggests being careful about them because of the integer rounding by division which can influence the computation result. However, no division operations were found during the manual review. Thus, these findings are considered false positives. Securify also claims that the order of transactions can influence who is the receiver of the funds and the amount of the transferred funds in Listing 6.4 from Protocol:

```
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));
  }
```

Listing 6.4: Initialization of Protocol Contract

However, due to the implemented system of different states (Request, Approve, Funding, among others), it is only possible to perform the transfer in states Rate_T or Rate_M: after the rating evaluation by a target and, if necessary, by mitigator is completed. Thus, the receiver and the amount of funds are not affected by the transaction order, but by the participant's ratings and whether the proof of work was uploaded on time. However, it is essential to note that the Protocol does not check the uploaded proof's content.

In the same line of the code, Securify identified an external call with a target that attackers can potentially manipulate. However, this finding is not a vulnerability because the target of the call (*owner*) is provided by the *evaluate()* function and can be only the target or the mitigator. Moreover, Securify marked the Register contract as greedy. According to the tool, the funds can be received by the contract but then locked because there is no opportunity to extract them. As Register does not support any Ether transfer, this finding is considered to be a false positive.

Finally, Remix found a vulnerability that is not in the list of Appendix E. According to Remix and Solidity documentation [67], when a *string* is converted to *bytes*, its length is calculated in bytes and not in characters as it might be expected, which can lead to misreadings. However, in the Listing 6.5 from Protocol, it is only checked if the proof is empty, which would mean that both lengths in characters and bytes would equal zero.

```
if(bytes(Proof).length ==0){
    return  endProcess ();
}
```

Listing 6.5: Check for Empty Proof in Protocol Contract

Thus, although it is acknowledged that developers should be careful when converting a *string* to *bytes*, in the current example, it is not considered it as a vulnerability in BloSS. Other vulnerabilities did not match any vulnerabilities but were considered to be true positives. For example, Remix finds a warning when there are two variables in the Protocol with similar names (*Target* and *_target*). Securify 2.0 claims that the Solidity pragma version statement in Migrations is too complex (*e.g.*, pragma solidity >=0.4.21 <0.6.0;). Moreover, Securify 2.0 suggests renaming *last_completed_migration* variable in Migrations because it does not comply with the Solidity naming convention as all of these findings are correct and evaluated as true positives.

Table 6.18 contains an overview of results.

Table 6.18: Overview of True Positives and False Positives in the Findings of the Security Audit Tools

	Mythril	**Securify**	**Securify 2.0**	**Remix**	**Manticore**
True Positives	5	10	16	8	0
False Positives	1	15	1	60	0
Total	**6**	**25**	**17**	**68**	**0**

It is important to note that two findings of Remix were removed from the overview because they were related to an internal error message in Remix and, therefore, could not be considered neither as true positives nor as false positives. As a result, Remix has only 68 instead of 70 findings in Table 6.18. Also, Table 6.18 demonstrates that the number of findings does not necessarily correlate with the number of true positives. Even though Remix has by far the highest number of findings, it has a rather low number of true positives. In contrast, Securify 2.0 with relatively few findings has 16 true positives and only one false positive. It was able to identify the highest number of security audit findings and has, therefore, the lowest number of false negatives.

Altogether, the novelty of the area in auditing SC vulnerabilities and their tools is demonstrated by the differences between the security tools. Tools are based on different types of analysis (*cf.*, Chapter 2) and, therefore, use various analysis strategies with different vulnerability types in scope. In the concrete example of BloSS smart contracts, Securify and Securify 2.0 performed the best. Although Remix had the largest number of findings, most of them were evaluated as false positives during the manual security analysis. Mythril demonstrated good results regarding the ration of true and false positives. However, as it focuses only on finding four types of vulnerabilities, it has a high number of false negatives.

6.8 Key Observations

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 [198]). It enables a flexible and efficient DDoS mitigation solution across multiple domains based on a permissioned PoA Ethereum [253], 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 stated above and in [183, 265].

The experiments and evaluations performed during the thesis had, in principle, the objective of refining the architecture and services and validating core functionality of BloSS. In this sense, local experiments are both based on simulators (*e.g.*, Mininet [123] and Ganache [235]) and hardware prototypes implemented in the BloSS cluster were of fundamental importance to outline the initial architecture and operation of the service, offering rapid prototyping and verification of results. Global evaluations on virtual machines spread across the world to evaluate performance characteristics and the simplicity of deployment and operation of the prototype. Therefore, being of fundamental importance for reducing technical complexities and reaching the first objective outlined for this thesis.

The evaluation conducted in Section 6.2) was fundamental from a correctness point-of-view to validate the design of both the protocol (*cf.*, Chapter 4) implemented in the SCs and the various BloSS instances connected to the Ethereum network (*cf.*, Chapter 5). From the purely software based design where BloSS acts as a VNF connected to the SDN controller, the instantiations could be easily ported to VMs across the world in the global experiments in Section 6.4 (*i.e.*, off-chain signaling performance) and Section 6.6 (on-chain signaling performance).

Overall, the technical, social, economic, and legal aspects of BloSS have been achieved. BloSS is an effective solution as a defense strategy in cases of DDoS large-scale attacks, such as the Memcached attack on GitHub servers surpassing 1.35 TBit/s [117] and DynDNS, which peaked 1.2 TB/s resulting in the unavailability of significant Internet services [192]. In such cases, traditional centralized defenses can be easily overloaded, and cooperation between organizations is useful to reduce 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.

By designing the BloSS dApp with SDN and NFV-capabilities, the critical advantage of a quick deployment was engineered into BloSS, inherent to competing systems like CoFence [195] or Bohatei [69]. 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.

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. Also, it is possible to create circles by connecting different BloSS instances through interoperability between BCs. This could be achieved by employing a Notary-based agnostic solution such as [215, 216].

It is also important to note that there is a dichotomy in using reputation assessment mechanisms within a cooperative alliance with pre-trusted members. If, on the one hand, there is the need to prevent free-riding and abuses in the use of cooperative services, on the other hand, the pre-trust requirement can reduce or even eliminate the need to add these mechanisms based on trust between members. However, it is understood that the great asset of collaborative defense against large-scale DDoS attacks lies in its ability to distribute mitigation points. Therefore, in situations where there are transitive trust relationships as such a particular member *A* trust another *B*, and *B* trust a member *C*, then A can trust C.

The need for trust between different organizations is a major social challenge whose mere use of technology does not provide trust. For example, the leakage of information about attacks (*e.g.*, frequency, downtime) in a cooperative defense can generate impacts from a commercial point of view, and with this, competitors can be exploited. In this sense, BC has the function of acting as a tool capable of increasing trust levels between members who already have certain levels of trust with each other - enough to collaborate in a cooperative defense. The addition of a BC-based reputation system whose ranking can be calculated transparently allows such transitive trust relationships to be encouraged.

As incentives already stipulate social behavior, by providing a platform to exchange incentives, BloSS creates a new scenario for mitigation defenses [81]. By extending cloud-based protection services, a decentralized marketplace for protection services based on BCs is made possible based on this design. It is essential that, in addition to trust, there are incentives that are based on currencies or agreements for the mutual exchange of mitigation services for a cooperative defense to be successful. 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 [71], as selecting a suitable protection provider based on specified requirements can optimize costs.

Lastly, 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 each operator's legal settings, possibly restricting the participation or interaction with particular regions or operators.

7 Summary, Conclusions, and Future Research

This Chapter concludes the thesis summarizing the achievements, contributions, and future work, which resulted in several contributions related the posed Research Questions (RQs).

7.1 Summary

The distributed nature of DDoS attacks suggests that a distributed and coordinated defense is necessary for a successful defense. Although the advantages of distributed defenses are recognized in the literature, still does not exist a widespread deployment of such systems because of their lack of effectiveness and implementation complexities. In addition to technical complexities involving how to share attack signaling information, the lack of incentives for third parties to actively participate in mitigating DDoS attacks by using their infrastructure makes the operation often not attractive. Thus, as major outlines of Chapter 2 in this thesis, among the challenges of existing approaches are the high complexity of operation and coordination, the need of trusted communication, and lack of incentives for the service providers to cooperate.

Building upon those challenges this this thesis has proposed and investigated a novel technique for implementing a decentralized cooperative DDoS defense based on an immutably and verifiable structure – Blockchain (BC). BloSS enables a flexible and efficient DDoS mitigation solution across multiple domains based on a permissioned BC, in which only pre-selected operators participate in the cooperative defense. 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 introduces a service model in which costs can be shifted from the AS operators to potential customers subscribing to a purpose-built Mitigation-as-a-Service (MaaS) offering. This works by directly using the fees paid by the customer as incentives between the operator of the AS the customer resides in and the operators of the ASes where the attack stems from.

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. Hence, the reputation mechanism is highly relevant to foster trust by allowing a mutual evaluation between providers of mitigation service and customers (*i.e.*, mitigator and target) in a transparent manner. The contract stipulated in BloSS maps all possible alternatives for interaction between the two parties by stipulating deadlines so that both parties provide input regarding the service and the rating of the service (rating).

Overall, the main achievement and advantages reached with the design and prototypical implementation and the evaluation of BloSS include *(a)* the use of an existing 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.

The use of a purely software-based approach with well-defined interfaces (API and Web) and intuitive use, allows BloSS to be easily deployed and interconnected with standard network management systems. As demonstrated in the evaluation Chapter 6, the prototype developed and evaluated locally was ported and deployed in a simple and efficient way in different virtual machines across the world without the imposition of hardware and special registrations in the underlying network infrastructure. BC fits with an ideal platform for data replication among all members of the collaborative defense if and only if based on a consensus protocol allowed to network members, which must be selected according to pre-established criteria for alliance. In this sense, as discussed in Chapter 4, the use of a consensus mechanism based on Proof-of-Work (PoW) in a collaborative defense setting is unnecessary due to the strict need for confidentiality and performance.

BloSS balances confidentiality and performance characteristics using an off-chain communication channel based on IPFS. In this way, on-chain communication is limited to the signaling process without revealing details of the mitigation service negotiation to other members of the BloSS-network. In the design described in the Chapter 5, the data encryption strategy is presented through a combination of asymmetric cryptography through RSA with 2048 bit keys and symmetric cryptography through Fernet. In this way, blacklisted or whitelisted address lists are transferred off-chain via IPFS while that of the terms of service is placed on-chain. In this regard, global experiments demonstrated (*cf.*, Chapter 6) that the protocol has an average execution time of 96.9 seconds to be completed in all scenarios (different outcomes of the protocol).

7.1.1 Major Contributions

The main contributions of this thesis are the design of the approach and its proof of concept, as well as the analysis of the literature providing an updated view of the state-of-the-art. Other contributions are derived from these major contributions, which are, for instance, related to the prototyping of the modules that belongs to BloSS duly documented in their related scientific publications.

- **Fostering Trust in a Cooperative Network Defense**. This is a fundamental point to leverage the use of cooperative tools involving the exchange of critical information (*e.g.*, information in relation to attack and respective standards, impacts in terms of downtime, among others). In addition to technical challenges concerning the signaling of attacks and the exchange of information between participants, trust between members in order to avoid information misuse or malicious actions that has fundamental relevance. In this sense, although there is no purely technical approach to guarantee the existence of trust, BloSS offers, through its blockchain-based architecture, a decentralized and transparent tool serving as platform for signaling attacks and verifying actions of members.

- **Design and Proof-of-Concept of a Blockchain-based Collaborative Signaling Approach**. To the best of the author's knowledge, this is the first work that combines in a cooperative DDoS signaling system attacks BC concepts to provide incentives and reputation management in this context. Henceforth, the main contribution of this thesis was the conception of architecture and a system as a proof of concept showing that, while it is possible to simplify the deployment and operation of collaborative defenses, it is also possible to include aspects related to incentives, confidentiality, and legal aspects within the same system.

- **Platform for the Creation of On-demand Mitigation Services**. BloSS' design enable a platform for mitigation services in a decentralized fashion. As highlighted in Chapter 2, there is DDoS defense mechanism incorporating incentive mechanisms in exchange for cooperative mitigation services, the realization of which impacts operating costs. The BloSS design based a state-machine smart contract where the outcome of each step is possible to be verified in a transparent way for all members of the alliance, allows members in a BloSS-network to rent their infrastructure based on incentives to cover these operating expenses.

- **Analysis on How Cooperative Defense Challenges are Tackled in the State-of-the-art**. This thesis complements existing surveys in the area of collaborative defenses, analyzing the current state-of-the-art, the challenges pointed out by these surveys. Namely, the challenges raised in this thesis in the technological, social, economic, and legal areas. This updated view of the state of the art allows us to observe how relatively recent technologies such as SDN, NFV, and current BC within this context, providing greater efficiency in signaling and mitigating attacks. Hence, this thesis approached over 70 related references, organizing them based on the action of the mechanism on the network (source, destination, network, or hybrid).

7.1.2 Review of Research Questions

Based on those contributions made in this thesis, the four research questions can be answered. Those questions are reviewed, and the answers to them are provided as below:

RQ1: Can a BC-based cooperative system reduce operation and deployment complexities?
This is one of the main aspects considered in the design of BloSS to minimize the impact on the underlying networking infrastructure. Hence, based on a software-based approach, it was possible to observe a simplification of deployment and operation with BloSS, being used both in virtual machines and in systems based on ARM (Raspberry Pi and ASUS Tinkerboard). On the one hand, software-based approaches facilitate deployment and operation; on the other hand, they can have a significant impact on the solution's performance. Therefore, the system's evaluation was of fundamental importance to observe how the system behaves in different environments (simulation, local, and global). Results presented in Chapter 6 show that BloSS presents a latency for signaling attacks that allows a reaction insufficient time in the parties involved (*i.e.*, based on average duration times for large-scale DDoS attacks, which is the scenario targeted for BloSS). Another decisive factor towards simplifying the operation is the BloSS focus on signaling attacks without involving specific demands regarding the method of mitigating attacks. Therefore, a mitigation service between two parties is specified, which must mutually agree with the terms, including aspects related to the way of mitigating the attack.

RQ2: How to balance transparency and privacy in a cooperative system, increasing trust among cooperative members? While transparency is a crucial aspect of establishing trust, confidentiality, and integrity of information concerning DDoS attacks is essential. BloSS handles this trade-off with a mix between information traded on-chain and the data needed to mitigate the attack being sent off-chain to maintain the confidentiality and integrity of the information. The cooperative signaling contract stipulates that on-chain information is limited to which networks an AS operates, the stipulated incentive to carry out a mitigation service, and its reputation scores as a mitigator (M) and target (T). Once a contract is initiated between a T and an M, data is sent off-chain according to the scheme described in encryption procedure (*cf.*, Figure 5.9). Also, BloSS is suitable for alliances with a pre-established level of trust, in which it is possible to foster an increase in their trust levels through the transparent and verifiable data structure. However, by using BloSS, it is impossible to establish trust (and cooperation) between members who do not trust each other initially. Lastly, while BloSS guarantees the confidentiality and integrity of information in transit (*i.e.*, data transmission), aspects related to trust, which permeate human relationships, are often subjective and dependent on non-technical aspects.

RQ3: How to provide financial incentives to foster cooperative behavior among its members? Through the BC-based structure, BloSS establishes the exchange of incentives for performing straightforward mitigation services. However, more important than just enabling the exchange of incentives is the approach described in the protocol in order to prevent fraud once there are no ways to guarantee proof of the effectiveness of a mitigation service (*i.e.*, proof-of-mitigation). In this sense, the cooperative signaling protocol (*cf.*, Chapter 4) establishes stages of mutual evaluation between T and M so that both interacting parties evaluate each other. Also, the cooperative signaling protocol foresees in its extreme design cases where T and M do not reach consensus with each other (*cf.*, Scenario 9 in Figure 4.2), and the situation needs to be escalated for off-chain evaluation. In these cases, an evaluating committee within the participating network can verify the outcomes of each stage in the protocol and evaluate the best consensus decision. However, it is noted that there is no automated (*i.e.*, on-chain) way to resolve conflicts in Scenario 9.

RQ4: How to ensure compliance across different jurisdictions? It is a significant concern in the legal area but not the technical sphere since different countries and regions have different laws, as well as companies, have internal policies to preserve both consumer information and their public image. From a technical perspective, such alliances can be formed between members with a minimum level of pre-established trust members and/or regions with similar legislation. Hence, such points can be checked by determining local parameters (*i.e.*, at the BloSS dApp) concerning the participation of the collaborative information network.

7.2 Conclusions

Based on contributions and responses to research challenges, it is possible to conclude certain general aspects concerning cooperative defenses, outlined as below:

- A BC-based cooperative defense can foster trust, but it is not able to create trust between, initially, unknown members. Trust is a major aspect of a cooperative defense, and there are no technical approaches to create trust. However, through a transparent and verifiable structure, while sensitive information is sent confidentially off-chain, it is possible to foster an increase in trust in the cooperative network.

- Limiting functions in a cooperative system can be an important step in expanding its applicability. Thus, the inclusion of mitigation functions or a design strongly coupled with a specific type of technology or hardware can negatively affect the participation in the collaborative defense. In this sense, the software-based approach presented serves as evidence of wide applicability in different architectures focusing only on signaling attacks but not specifying how a mitigation service should be performed.

- Cooperative signaling and mitigation of DDoS attacks are a near real-time instead of real-time services. Hence, it is not a service that needs to be performed in the temporal order of milliseconds to seconds, but the space of minutes-hours. In this sense, performance is of secondary importance in collaborative defense, with other elements such as trust (confidentiality, integrity), incentives, and legal aspects being most relevant to cooperation.

- Aspects related to incentives to perform tasks to detect or mitigate DDoS atteacks are often overlooked in cooperative defenses. This is a conclusion derived from Chapter 3, in which it is possible to verify that no work listed mentions the financial aspect as flawless in the operation of collaborative defense. However, tasks related to detection and mitigation imply operational and capital expenses, which must be observed to encourage participation.

- Although challenges in the legal sphere are relatively easy to resolve from a technical point of view, the complexity of internal and external legislation impairs the information sharing. For example, in many situations, the sharing of IP addresses (which can be correlated to individuals), implies a violation of personal rights. Therefore, sharing between different blacklisted IP addresses organizations may violate legislation, although it is possible to guarantee the confidentiality and integrity of the information in transit.

7.3 Future Research

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 analyzing how actors (especially Ts and M s) are impacted by different incentive values required to perform a mitigation service, thus, fostering the development of DDoS protection markets. Then, such markets can define different tokens based on the economic and geopolitical conditions of the alliance members, in which a token can represent a certain value of a fiat currency and be used to exchange mitigation services. The value of a token unit can be defined not only by the operational cost of maintaining the mitigation of the attack, but also by a series of economic, political, and ethical factors, involving alliance members to whom they can influence the value. the token and the cost of implementing a mitigation service.

Considering a technical perspective in the current implementation of BloSS, improvements are also possible. Thus, 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 M or the T, the BloSS register can be extended to enable a ranking and separate positively rated actors from negatively rated ones.

Future runs in a real DDoS scenario are still to be tested. Depending on the scenario, deadlines intervals can be improved to enable DDoS mitigation. When testing the cooperative signaling protocol, it might not be apparent during the initialization of how long the mitigation process may take. Therefore, it may be the case that a different mechanism should be implemented to predict how long actual mitigation takes or the possibility to be extended such that a M that is actively defending will still be rewarded, instead of missing a deadline. Also, instead of storing raw names and strings in the Register (*i.e.*, the *rendez vous* to start a cooperative protocol), hashes of the data or even hashes of the storage address could be stored on the BC. Based on this, ratings of M or the T could be extended in the Register to enable the possibility of ranking and separating positively rated actors from negatively rated ones.

Based on the experimental nature of the BloSS concept, it is envisioned its integration and deployment in research projects cybersecurity at a nation-wide level. For example, the Horizon 2020 CONCORDIA project to which related tasks and various discussions influenced design decisions related to providing incentives. In this sense, since the project is composed of different industry and academic organizations, its deployment in an experimental nature could reveal further insights for improving the BloSS prototype in a realistic setup and possibly based on the signaling of traces of real cases.

It is also essential to note that the approach independent of specific underlying technologies contributes to this thesis. Thus, several aspects of implementation can be optimized so that the evaluation chapter's results can be enhanced. In this context, it is possible to develop a BC-based structure specially crafted to exchange information in the context of collaborative defense, simplifying the definition of the contract on a native platform and the exchange of off-chain data through encryption mechanisms.

Bibliography

[1] Zakaria Abou El Houda, Abdelhakim Senhaji Hafid, and Lyes Khoukhi. Cochain-SC: An Intra-and Inter-domain DDoS Mitigation Scheme based on Blockchain using SDN and Smart Contract. *IEEE Access*, 7:98893–98907, 2019.

[2] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *The 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, page 41–52, New York, NY, USA, October 2006. Association for Computing Machinery. ISBN 1595935614. doi: 10.1145/1177080.1177086. URL https://doi.org/10.1145/1177080.1177086.

[3] Akamai. How to Protect Against DDoS Attacks - Stop Denial of Service, 2020. URL https://www.akamai.com/us/en/resources/protect-against-ddos-attacks.jsp.

[4] Akamai. The State of the Internet, 2020. URL https://www.akamai.com/us/en/resources/our-thinking/state-of-the-internet-report/global-state-of-the-internet-security-ddos-attack-reports.jsp.

[5] Maher Alharby and Aad van Moorsel. Blockchain Based Smart Contracts: A Systematic Mapping Study. *Computer Science & Information Technology (CS & IT)*, August 2017. doi: 10.5121/csit.2017.71011. URL http://dx.doi.org/10.5121/csit.2017.71011.

[6] Sidney Amani, Myriam Bégel, Maksym Bortin, and Mark Staples. Towards Verifying Ethereum Smart Contract Bytecode in Isabelle/HOL. In *The 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2018, page 66–77, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355865. doi: 10.1145/3167084. URL https://doi.org/10.1145/3167084.

[7] Michele Amoretti, Giacomo Brambilla, Francesco Medioli, and Francesco Zanichelli. Blockchain-Based Proof of Location. In *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 146–153, Lisbon, Portugal, July 2018. IEEE.

[8] David G. Andersen. Mayday: Distributed Filtering for Internet Services. In *The 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, page 3, Seattle, WA, USA, March 2003. USENIX Association.

[9] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the Mirai Botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Santa Clara, California, USA, August 2017.

[10] Katerina J Argyraki and David R Cheriton. Active Internet Traffic Filtering: Real-Time Response to Denial-of-Service Attacks. In *USENIX Annual Technical Conference, General Track*, volume 38, Anaheim, California, USA, April 2005.

[11] Hadi Asghari, Michel JG van Eeten, and Johannes M Bauer. Economics of Fighting Botnets: Lessons from a Decade of Mitigation. *IEEE Security & Privacy*, 13(5):pp. 16–23, 2015.

[12] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A Survey of Attacks on Ethereum Smart Contracts (SoK). *The 6th International Conference on Principles of Security and Trust*, pages 164–186, March 2017. doi: 10.1007/978-3-662-54455-6_8.

[13] Audun Josang and Roslan Ismail. The Beta Reputation System. In *The 15th Bled Electronic Commerce Conference*, volume 5, pages 2502–2511, Bled, Slovenia, June 2002. 01484.

[14] Maria Bada, Sadie Creese, Michael Goldsmith, Chris Mitchell, and Elizabeth Phillips. Improving the Effectiveness of CSIRTs, 2014. URL `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3659982`.

[15] Hammi Badis, Guillaume Doyen, and Rida Khatoun. A Collaborative Approach for a Source based Detection of Botclouds. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 906–909, Ottawa, Canada, May 2015. IEEE.

[16] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Consensus in the Age of Blockchains. *arXiv preprint arXiv:1711.03936*, 2017.

[17] Juan Benet. IPFS-Content Addressed, Versioned, P2P File System. *arXiv preprint arXiv:1407.3561*, 2014. URL `https://arxiv.org/abs/1407.3561`.

[18] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof-of-Activity: Extending Bitcoin's Proof-of Work via Proof-of-Stake. *ACM SIGMETRICS Performance Evaluation Review*, 42(3):34–37, 2014.

[19] Karthikeyan Bhargavan, Nikhil Swamy, Santiago Zanella-Béguelin, Antoine Delignat-Lavaud, Cedric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, Natalia Kulatova, Aseem Rastogi, and Thomas Sibut-Pinote. Formal Verification of Smart Contracts: Short Paper. *The 2016 ACM Workshop on Programming Languages and Analysis for Security. 2016*, pages 91–96, October 2016. doi: 10.1145/2993600.2993611.

[20] T. Bocek, M. Shann, D. Hausheer, and B. Stiller. Game Theoretical Analysis of Incentives for Large-Scale, Fully Decentralized Collaboration Networks. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, Washington, DC, USA, April 2008. doi: 10.1109/IPDPS.2008.4536195. 00020.

[21] Thomas Bocek and Burkhard Stiller. Smart Contracts–Blockchains in the Wings. In *Digital marketplaces unleashed*, pages 169–184. Springer, 2018.

[22] Giovanni Bottazzi and Gianluigi Me. The Botnet Revenue Model. In *The 7th International Conference on Security of Information and Networks*, pages 459–465, Glasgow, Scotland, September 2014.

[23] Zdravko Bozakov and Panagiotis Papadimitriou. Autoslice: Automated and Scalable Slicing for Software-Defined Networks. In *The 2012 ACM Conference on CoNEXT Student Workshop*, pages 3–4, Nice, Paris, December 2012. ACM.

[24] Tim Bray. The JavaScript Object Notation (JSON) Data Interchange Format, 2017. URL `https://tools.ietf.org/html/rfc8259`. [Internet Engineering Task Force (IETF) - RFC 8259].

[25] Richard Brown. On Distributed Databases and Distributed Ledgers. URL: `https://bit.ly/3rB2Oop`, 2016.

[26] Yulia Brun. A Security Audit of the Blockchain Signaling Systems Protocol, Aug 2020. URL `https://files.ifi.uzh.ch/CSG/staff/rodrigues/extern/theses/ba-yulia.pdf`.

[27] Ryan Brunt, Prakhar Pandey, and Damon McCoy. Booted: An Analysis of a Payment Intervention on a DDoS-for-Hire Service. In *Workshop on the Economics of Information Security (WEIS)*, pages 1–12, Fairfax, United States of America (USA), April 2017.

[28] Ethan Buchman. Tendermint: Byzantine Fault Tolerance in the Age of Blockchains, 2016. URL `https://atrium.lib.uoguelph.ca/xmlui/bitstream/handle/10214/9769/Buchman_Ethan_201606_MAsc.pdf`.

[29] Joao Cabrera, Lundy Lewis, Xinzhou Qin, Wenke Lee, Ravil K Prasanth, B Ravichandran, and Raman K Mehra. Proactive Detection of Distributed Denial-of-Service Attacks using MIB Traffic Variables-a Feasibility Study. In *IEEE/IFIP International Symposium on Integrated Network Management (IM 2001)*, pages 609–622, Seattle, USA, May 2001. IEEE.

[30] Yuanfeng Cai and Dan Zhu. Fraud Detections for Online Businesses: A Perspective from Blockchain Technology. *Financial Innovation*, 2(1):20, December 2016. ISSN 2199-4730. doi: 10.1186/s40854-016-0039-4. URL `https://bit.ly/3q3MqN5`.

[31] Davide Carboni. Feedback based Reputation on top of the Bitcoin Blockchain. *Arxiv*, abs/1502.01504, February 2015. 00008.

[32] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. In *Proocedings of the Third Symposium on Operating Systems Design and Implementation (USENIX OSDI)*, 1999.

[33] Brian Caswell, James C. Foster, Ryan Russell, Jay Beale, and Jeffrey Posluns. *Snort 2.0 Intrusion Detection*. Syngress Publishing, 2003. ISBN 1931836744.

[34] ChainSecurity. Securify. URL: `https://securify.chainsecurity.com/`, 2020.

[35] Huashan Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu. A Survey on Ethereum Systems Security: Vulnerabilities, Attacks and Defenses, 2019.

[36] T. Chen, X. Li, X. Luo, and X. Zhang. Under-optimized Smart Contracts Devour Your Money. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER 2017)*, pages 442–446, Klagenfurt, Austria, February 2017.

[37] Ting Chen, Xiaoqi Li, Ying Wang, Jiachi Chen, Zihao Li, Xiapu Luo, Man Ho Au, and Xiaosong Zhang. An Adaptive Gas Cost Mechanism for Ethereum to Defend Against Under-Priced DoS Attacks, 2017.

[38] Xiao-Fan Chen and Shun-Zheng Yu. CIPA: A Collaborative Intrusion Prevention Architecture for Programmable Network and SDN. *Computers & Security*, 58:1–19, 2016.

[39] Yu Chen, Kai Hwang, and Wei-Shinn Ku. Collaborative Detection of DDoS Attacks over Multiple Network Domains. *IEEE Transactions on Parallel and Distributed Systems*, 18(12): 1649–1662, 2007.

[40] Tommy Chin, Xenia Mountrouidou, Xiangyang Li, and Kaiqi Xiong. An SDN-supported Collaborative Approach for DDoS Flooding Detection and Containment. In *IEEE Military Communications Conference (MILCOM 2015)*, pages 659–664, Tampa, Florida, USA, October 2015. IEEE.

[41] Jin-Hee Cho, Kevin Chan, and Sibel Adali. A Survey on Trust Modeling. *ACM Computing Surveys (CSUR)*, vol. 48:pp. 28–40, 2015.

[42] Christopher Georgen. Topl, Empowering Growth by Enabling Investment, 2017. URL `https://github.com/Topl/whitepaper/blob/master/Whitepaper.pdf`. 00000.

[43] CloudFare. CloudFlare Advanced DDoS Protection, 2020. URL `https://www.cloudflare.com/static/media/pdf/cloudflare-whitepaper-ddos.pdf`.

[44] CloudFare. What is a DDoS Attack?, 2020. URL `https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/`.

[45] Adrian Colyer. SoK: Consensus in the Age of Blockchains, Aug 2018. URL `https://bit.ly/3p5L5Ux`.

[46] ConsenSys Diligence. MythX. https://mythx.io/about/, 2020.

[47] ConsenSys Diligence. Smart Contract Weakness Classification Registry. URL: https://swcregistry.io/, 2020.

[48] Evan Cooke, Farnam Jahanian, and Danny McPherson. The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets. *SRUTI*, 5:pp. 6–6, 2005.

[49] Victor Costan and Srinivas Devadas. Intel SGX Explained. *IACR Cryptology ePrint Archive*, 2016.

[50] Patrick Cousot and Radhia Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages*, pages 238–252, Madrid, Spain, January 1977. doi: 10.1145/512950.512973.

[51] Flaviu Cristian, Houtan Aghili, Ray Strong, and Danny Dolev. Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement. *Information and Computation*, 118(1):158–179, 1995.

[52] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On Scaling Decentralized Blockchains. In *International Conference on Financial Cryptography and Data Security (FC'16)*, pages 106–125, Accra Beach, Barbados, February 2016. Springer.

[53] CWE. Common Weakness Enumeration. URL: https://cwe.mitre.org/, 2020.

[54] Richard Cziva and Dimitrios P. Pezaros. Container Network Functions: Bringing NFV to the Network Edge. *IEEE Communications Magazine*, 55(6):24–31, 2017.

[55] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. PBFT vs Proof-of-Authority: Applying the Cap Theorem to Permissioned Blockchain, 2018.

[56] Christian Decker and Roger Wattenhofer. Information Propagation in the Bitcoin Network. In *IEEE P2P 2013 Proceedings*, pages 1–10. IEEE, 2013.

[57] Dd B. DeFigueiredo and Earl T. Barr. Trustdavis: A non-exploitable online reputation system. In *E-Commerce Technology, 2005. CEC 2005. Seventh IEEE International Conference On*, pages 274–283, Munich, Germany, July 2005. IEEE. 00057.

[58] Sergi Delgado-Segura, Cristian Tanas, and Jordi Herrera-Joancomartí. Reputation and Reward: Two Sides of the Same Bitcoin. *Sensors*, 16(6):776, May 2016. doi: 10.3390/s16060776. URL http://www.mdpi.com/1424-8220/16/6/776.

[59] Mieso K. Denko. Detection and Prevention of Denial-of-Service (DoS) Attacks in Mobile Ad Hoc Networks Using Reputation-Based Incentive Scheme. *Journal of Systemics, Cybernetics and Informatics*, 3(4):1–9, 2005.

[60] R. Dennis and G. Owen. Rep on the Block: A next Generation Reputation System Based on the Blockchain. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 131–138, Londong, United Kingdom, December 2015. doi: 10.1109/ICITST.2015.7412073. 00006.

[61] Cryptography Developers. Python Cryptography Library. `https://cryptography.io/en/latest/`, 2017.

[62] A. Dika and M. Nowostawski. Security Vulnerabilities in Ethereum Smart Contracts. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 955–962, 2018.

[63] Wesley Dingman, Aviel Cohen, Nick Ferrara, Adam Lynch, Patrick Jasinski, Paul E. Black, and Lin Deng. Defects and Vulnerabilities in Smart Contracts, a Classification using the NIST Bugs Framework. *International Journal of Networked and Distributed Computing*, 7:121–132, 2019. ISSN 2211-7946. doi: https://doi.org/10.2991/ijndc.k.190710.003. URL `https://doi.org/10.2991/ijndc.k.190710.003`.

[64] Christos Douligeris and Aikaterini Mitrokotsa. DDoS Attacks and Defense Mechanisms: Classification and State-of-the-art. *Computer Networks*, 44(5):pp. 643–666, 2004.

[65] Lukas Eisenring. Performance Analysis of Blockchain Off-chain Data Storage Tools, Aug 2018. URL `https://files.ifi.uzh.ch/CSG/staff/rodrigues/extern/theses/ba-eisenring.pdf`.

[66] Meryam Essaid, DaeYong Kim, Soo Hoon Maeng, Sejin Park, and Hong Taek Ju. A Collaborative DDoS Mitigation Solution Based on Ethereum Smart Contract and RNN-LSTM. In *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–6, Matsue, Japan, September 2019. IEEE.

[67] Ethereum. Solidity Documentation. Control Structures. URL: `https://solidity.readthedocs.io/en/latest/control-structures.html`, 2020.

[68] F. Randall Farmer and Bryce Glass. *Building Web Reputation Systems*. O'Reilly, Sebastopol, CA, 1st ed edition, 2010. ISBN 978-0-596-15979-5. 00131 OCLC: ocn460060167.

[69] Seyed K Fayaz, Yoshiaki Tobioka, Vyas Sekar, and Michael Bailey. Bohatei: Flexible and Elastic DDoS Defense. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 817–832, Anaheim, California, USA, April 2015.

[70] M. Felici, N. Wainwright, S. Cavallini, and F. Bisogni. What's New in the Economics of Cybersecurity? *IEEE Security and Privacy,*, vol. 14:pp. 11–13, may 2016. ISSN 1540-7993. doi: 10.1109/MSP.2016.64.

[71] Muriel Franco, Bruno Rodrigues, and Burkhard Stiller. MENTOR: The Design and Evaluation of a Protection Services Recommender System. In *15th International Conference on Network and Service Management (IEEE CNSM 2019)*, pages 1–7, Halifax, Canada, October 2019.

[72] Jérôme François, Issam Aib, and Raouf Boutaba. FireCol: a Collaborative Protection Network for the Detection of Flooding DDoS Attacks. *IEEE/ACM Transactions on Networking*, 20(6): 1828–1841, 2012.

[73] Getoar Gallopeni. Botnet Command-and-Control Traffic Analysis, January 2020. URL `https://bit.ly/2ZoHgoY`.

[74] Diego Gambetta et al. Can We Trust Trust? *Trust: Making and Breaking Cooperative Relations*, vol. 13:pp. 213–237, 2000.

[75] J. Garae, R. K. L. Ko, and M. Apperley. A Full-Scale Security Visualization Effectiveness Measurement and Presentation Approach. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 639–650, Aug 2018. doi: 10.1109/TrustCom/BigDataSE.2018.00095.

[76] Thomer M Gil and Massimiliano Poletto. MULTOPS: A Data-Structure for Bandwidth Attack Detection. In *USENIX Security Symposium*, pages 23–38, Boston, Massachusetts. USA, June 2001.

[77] K Giotis, A Pavlidis, L Anagnostou, M Dimolianis, T Tsigkritis, D Kalogeras, N Kostopoulos, I Kotinas, and V Maglaris. Blockchain-based Federation of Network Providers for Collaborative DDoS Mitigation. In *3rd Symposium on Distributed Ledger Technology, Gold Coast, Australia, November*, Gold Coast, Australia, November 2018.

[78] Kostas Giotis, Maria Apostolaki, and Vasilis Maglaris. A Reputation-based Collaborative Schema for the Mitigation of Distributed Attacks in SDN Domains. In *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, pages 495–501, Istanbul, Turkey, April 2016. IEEE.

[79] James N Gray. Notes on Data Base Operating Systems. In *Operating Systems*, pages 393–481. Springer, 1978.

[80] Peter Gregory. *CISSP Guide to Security Essentials*. Cengage Learning, Boston, MA, USA, 2nd edition, 2015. ISBN 9781285060422.

[81] A. Gruhler, B. Rodrigues, and B. Stiller. A Reputation Scheme for a Blockchain-based Network Cooperative Defense. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2019)*, pages 71–79, April 2019. Washington, United States of America (USA).

[82] Andreas Gruhler. A Reputation and Reward Scheme for a Cooperative, Multi-domain DDoS Defense. Master's thesis, UZH, Zürich, Switzerland, July 2018. URL `https://files.ifi.uzh.ch/CSG/staff/rodrigues/extern/theses/ma-andreas.pdf`.

[83] Guofei Gu, Vinod Yegneswaran, Phillip Porras, Jennifer Stoll, and Wenke Lee. Active Botnet Probing to Identify Obscure Command and Control Channels. In *2009 Annual Computer Security Applications Conference (ACSAC '09)*, pages 241–253, Honolu, Hawaii, USA, December 2009. IEEE.

[84] Zhang Guangsen, Manish Parashar, et al. Cooperative Defence Against DDoS Attacks. *Journal of Research and Practice in Information Technology*, 38(1):69, 2006.

[85] Sufian Hameed and Hassan Ahmed Khan. SDN based Collaborative Scheme for Mitigation of DDoS Attacks. *Future Internet*, 10(3):23, 2018.

[86] Biao Han, Xiangrui Yang, Zhigang Sun, Jinfeng Huang, and Jinshu Su. OverWatch: A Cross-plane DDoS Attack Defense Framework with Collaborative Intelligence in SDN. *Security and Communication Networks*, 2018.

[87] Florian Hawlitschek, Benedikt Notheisen, and Timm Teubner. The Limits of Trust-Free Systems: A Literature Review on Blockchain Technology and Trust in the Sharing Economy. *Electronic commerce research and applications*, 29:pp. 50–63, 2018.

[88] D Henshel, MG Cains, B Hoffman, and T Kelley. Trust as a Human Factor in Holistic Cyber Security Risk Assessment. *Procedia Manufacturing*, 3:pp. 1117–1124, 2015.

[89] Cormac Herley and Dinei Florêncio. Nobody Sells Gold for the Price of Silver: Dishonesty, Uncertainty and the Underground Economy. In *Economics of information security and privacy*, pages 33–53. Springer, April 2010.

[90] K. Hong, Y. Kim, H. Choi, and J. Park. Sdn-assisted slow http ddos attack defense method. *IEEE Communications Letters*, 22(4):688–691, 2018.

[91] Yun Huang, Xianjun Geng, and Andrew B. Whinston. Defeating DDoS Attacks by Fixing the Incentive Chain. *ACM Trans. Internet Technol.*, 7(1):5–es, February 2007. ISSN 1533-5399. doi: 10.1145/1189740.1189745. URL `https://doi.org/10.1145/1189740.1189745`.

[92] Alice Hutchings and Richard Clayton. Exploring the Provision of Online Booter Services. *Deviant Behavior*, 37(10):1163–1178, 2016.

[93] Stewart Iain. Proof-of-Burn (PoB), 2012. URL `https://eprint.iacr.org/2019/1096.pdf`.

[94] Imperva. Distributed Denial-of-Service Attack (DDoS) Definition, jan 2019. URL https://bit.ly/3cZlzxN.

[95] John Ioannidis and Steven Michael Bellovin. Implementing PushBack: Router-based Defense Against DDoS Attacks, 2002. URL https://www.cs.columbia.edu/~smb/papers/pushback-impl.pdf.

[96] iotanalytics. State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating, Fev 2018. URL https://bit.ly/2MOtZoj.

[97] Sarah Jelinek. Design and Implementation of A Cooperative, Autonomous Anti-DDoS Network Using Intruder Detection and Isolation Protocol. *University of Colorado, Colorado Springs*, pages pp. 1–98, 01 2008.

[98] Y. Jia, F. Zhong, A. Alrawais, B. Gong, and X. Cheng. FlowGuard: An Intelligent Edge Defense Mechanism Against IoT DDoS Attacks. *IEEE Internet of Things Journal*, pages 1–11, 2020.

[99] Cheng Jin, Haining Wang, and Kang G Shin. Hop-Count Filtering: an Effective Defense Against Spoofed DDoS Traffic. In *The 10th ACM Conference on Computer and Communications Security (ACM CCS'03)*, pages 30–41, Washington, DC, USA, October 2003.

[100] A John and T Sivakumar. DDoS: Survey of Traceback Methods. *International Journal of Recent Trends in Engineering*, 1(2):241, 2009.

[101] Audun Jøsang, Roslan Ismail, and Colin Boyd. A Survey of Trust and Reputation Systems for Online Service Provision. *Decision Support Systems*, 43(2):618–644, March 2007. ISSN 0167-9236. doi: 10.1016/j.dss.2005.05.019. URL http://www.sciencedirect.com/science/article/pii/S0167923605000849.

[102] Sascha Just, Rahul Premraj, and Thomas Zimmermann. Towards the Next Generation of Bug Tracking Systems. In *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 82–85, Herrsching am Ammersee, Germany, September 2008. IEEE.

[103] Angelos D Keromytis, Vishal Misra, and Dan Rubenstein. SOS: Secure Overlay Services. *ACM SIGCOMM Computer Communication Review (ACM/CCR)*, 32(4):pp. 61–72, 2002.

[104] Alexander Khalimonenko, Oleg Kupreev, and Ekaterina Badovskaya. DDoS Attacks in Q1 2018. https://securelist.com/ddos-report-in-q1-2018/85373/, April 2018.

[105] Rizwan Khan and Avimanyou Vatsa. *Detection and Control of DDOS Attacks over Reputation and Score Based MANET*. Manet, November 2017. 00024.

[106] Sheharbano Khattak, Naurin Rasheed Ramay, Kamran Riaz Khan, Affan A Syed, and Syed Ali Khayam. A Taxonomy of Botnet Behavior, Detection, and Defense. *IEEE communications surveys & tutorials*, 16(2):pp. 898–924, 2013.

[107] Christian Killer. Security Management and Visualization in a Blockchain-based Collaborative Defense. Master's thesis, UZH, Zürich, Switzerland, December 2018. URL https://files.ifi.uzh.ch/CSG/staff/rodrigues/extern/theses/ma-christiankiller.pdf.

[108] Christian Killer, Bruno Rodrigues, and Burkhard Stiller. Threat Management Dashboard for a Blockchain Collaborative Defense. In *The IEEE GLOBECOM Workshop 27th on Blockchain in Telecommunications: Emerging Technologies for the Next Decade and Beyond*, pages 1–6, Waikoloa, USA, December 2019. IEEE. URL https://files.ifi.uzh.ch/CSG/staff/rodrigues/extern/publications/ICBC19_CK.pdf.

[109] Christian Killer, Bruno Rodrigues, and Burkhard Stiller. Security Management and Visualization in a Blockchain-based Collaborative Defense. In *ICBC 2019*, pages 108–111, Seoul, South Korea, May 2019. IEEE. URL https://bit.ly/3p5Knql.

[110] Jeonghye Kim, Youngseog Yoon, and Hangjung Zo. Why People Participate in the Sharing Economy: A Social Exchange Perspective. In *The 19th Pacific Asia Conference on Information Systems (PACIS 2015)*, page 76, Singapore, July 2015.

[111] Yoohwan Kim, Wing Cheong Lau, Mooi Choo Chuah, and H Jonathan Chao. PacketScore: a Statistics-based Packet Filtering Scheme Against Distributed Denial-of-Service Attacks. *IEEE transactions on dependable and secure computing*, 3(2):141–155, 2006.

[112] Matt Kindy. Divine: A Blockchain Reputation System For Determining Good Market Actors, June 2017. URL https://bit.ly/3hBE4IA. 00000.

[113] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. The Click Modular Router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.

[114] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. DDoS in the IoT: Mirai and Other Botnets. *Computer*, 50(7):pp. 80–84, 2017.

[115] Daniel Kopp, Matthias Wichtlhuber, Ingmar Poese, Jair Santanna, Oliver Hohlfeld, and Christoph Dietzel. DDoS Hide & Seek: On the Effectiveness of a Booter Services Takedown. In *The Internet Measurement Conference (ACM IMC)*, pages 65–72, New York, NY, USA, October 2019.

[116] H. Kopp, D. Mödinger, F. Hauck, F. Kargl, and C. Bösch. Design of a Privacy-Preserving Decentralized File Storage with Financial Incentives. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, pages 14–22, April 2017. doi: 10.1109/EuroSPW.2017.45.

[117] Seth Djane Kotey, Eric Tutu Tchao, and James Dzisi Gadze. On Distributed Denial of Service Current Defense Schemes. *Technologies*, 7(1):pp. 19–25, 2019.

[118] Georgios Koutepas, Fotis Stamatelopoulos, and Basil Maglaris. Distributed Management Architecture for Cooperative Detection and Reaction to DDoS Attacks. *Journal of Network and Systems Management (JNSM)*, 12(1):73–94, 2004.

[119] Grafana Labs. Grafana - The Open Platform for Analytics and Monitoring. `https://github.com/grafana/grafana`, July 2018.

[120] Protocol Labs. IPFS Stream Security Transport (libp2p-secio). `https://github.com/libp2p/go-libp2p-secio`, 2018.

[121] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem, 2019.

[122] Leslie Lamport et al. Paxos Made Simple. *Tech. Report*, 2001. URL `https://lamport.azurewebsites.net/pubs/paxos-simple.pdf`.

[123] Bob Lantz, Brandon Heller, and Nick McKeown. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *The 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1–6, Barcelona, Spain, August 2010.

[124] Felix Lau, Stuart H Rubin, Michael H Smith, and Ljiljana Trajkovic. Distributed Denial-of-Service Attacks. In *IEEE International Conference on Systems, Man and Cybernetics (IEEE SMC 2000).'Cybernetics Evolving to Systems, Humans, Organizations, and their complex interactions'(cat. no. 0*, volume 3, pages 2275–2280, Nashville, Tennessee, USA, October 2000. IEEE.

[125] James Andrew Lewis. *Assessing the Risks of Cyber Terrorism, Cyber War and other Cyber Threats*. Center for Strategic & International Studies Washington, DC, 2002.

[126] X. Li, Q. Wang, L. Yang, and X. Luo. Network Security Situation Awareness Method Based on Visualization. In *Third International Conference on Multimedia Information Networking and Security (MNES 2011)*, pages 411–415, Shanghai, China, November 2011.

[127] Orfeas Stefanos Thyfronitis Litos and Dionysis Zindros. Trust Is Risk: A Decentralized Financial Trust Platform. Technical Report 156, National Technical University of Athens, 2017. URL `http://eprint.iacr.org/2017/156`. 00000.

[128] Shengyun Liu, Paolo Viotti, Christian Cachin, Vivien Quéma, and Marko Vukolić. {XFT}: Practical Fault Tolerance Beyond Crashes. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages pp. 485–500, 2016.

[129] Xin Liu, Xiaowei Yang, and Yanbin Lu. To Filter or to Authorize: Network-layer DoS Defense Against Multimillion-node Botnets. In *The ACM SIGCOMM 2008 conference on Data communication*, pages 195–206, Seattle, WA, USA, August 2008.

[130] Xin Liu, Xiaowei Yang, and Yong Xia. NetFence: Preventing Internet Denial of Service from Inside Out. In *The ACM SIGCOMM 2010 Conference*, SIGCOMM '10, page 255–266, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450302012. doi: 10.1145/1851182.1851214. URL `https://doi.org/10.1145/1851182.1851214`.

[131] Yuan Liu, Zheng Zhao, Guibing Guo, Xingwei Wang, Zhenhua Tan, and Shuang Wang. An Identity Management System Based on Blockchain, 2017. URL https://www.ucalgary.ca/pst2017/files/pst2017/paper-8.pdf.

[132] Zhuotao Liu, Hao Jin, Yih-Chun Hu, and Michael Bailey. MiddlePolice: Toward enforcing Destination-defined Policies in the Middle of the Internet. In *The 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1268–1279, Vienna, Austria, October 2016.

[133] Zhuotao Liu, Hao Jin, Yih-Chun Hu, and Michael Bailey. Practical Proactive DDoS-attack Mitigation via Endpoint-driven In-network Traffic Control. *IEEE/ACM Transactions on Networking*, 26(4):pp. 1948–1961, 2018.

[134] Pradeep Loganathan. Distributed Denial-of-Service Attack (DDoS) Definition, Jul 2017. URL https://pradeeploganathan.com/blockchain/consensus/.

[135] Yaobin Lu, Ling Zhao, and Bin Wang. From Virtual Community Members to C2C E-commerce Buyers: Trust in Virtual Communities and its Effect on Consumers' Purchase Intention. *Electronic Commerce Research and Applications*, 9(4):pp. 346–360, 2010.

[136] Yiqin Lu and Meng Wang. An Easy Defense Mechanism Against Botnet-Based DDoS Flooding Attack Originated in SDN Environment Using SFlow. In *The 11th International Conference on Future Internet Technologies*, CFI '16, page 14–20, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450341813. doi: 10.1145/2935663.2935674. URL https://doi.org/10.1145/2935663.2935674.

[137] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making Smart Contracts Smarter. In *The 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 254–269, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450341394. doi: 10.1145/2976749.2978309. URL https://doi.org/10.1145/2976749.2978309.

[138] Leandros A. Maglaras, Ki-Hyung Kim, Helge Janicke, Mohamed Amine Ferrag, Stylianos Rallis, Pavlina Fragkou, Athanasios Maglaras, and Tiago J. Cruz. Cyber Security of Critical Infrastructures. *ICT Express,*, vol. 4:pp. 42 – 45, 2018. ISSN 2405-9595. doi: https://doi.org/10.1016/j.icte.2018.02.001. URL http://www.sciencedirect.com/science/article/pii/S2405959517303880. SI: CI and Smart Grid Cyber Security.

[139] Ratul Mahajan, Steven M Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. Controlling High Bandwidth Aggregates in the Network. *ACM SIGCOMM Computer Communication Review*, 32(3):62–73, 2002.

[140] ManaNET. MANANet DDoS Whitepaper, Jun 2020. URL http://www.cs3-inc.com/pubs/ps_MANAnet-Reverse-Firewall.pdf.

[141] S. Mannhart, B. Rodrigues, E. Scheid, S. S. Kanhere, and B. Stiller. Toward Mitigation-as-a-Service in Cooperative Network Defenses. In *2018 IEEE 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (CyberSciTech 2018)*, pages 362–367, Aug 2018. Athens, Greece.

[142] Stephan Mannhart. Mitigation as a Service in a Cooperative Network Defense. Master's thesis, UZH, Zürich, Switzerland, July 2018. URL `https://files.ifi.uzh.ch/CSG/staff/rodrigues/extern/theses/ma-stephan.pdf`.

[143] Stephan Mannhart. Mitigation as a Service in a Cooperative Network Defense. Master's thesis, University of Zürich, Binzmuehlestrasse 14, 8050 Zürich, Switzerland, July 2018.

[144] Stephan Mannhart. A Custom Debian Stretch Package for amd64 and armhf to Install the Solidity Compiler solc v0.4.24. `https://github.com/savf/solc.deb`, June 2018.

[145] Ognjen Maric, Christoph Sprenger, and David Basin. Consensus Refined. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 391–402. IEEE, 2015.

[146] Chris McGuffin and Paul Mitchell. On Domains: Cyber and the Practice of Warfare. *International Journal: Canada's Journal of Global Policy Analysis,*, vol. 69:pp. 394–412, 2014.

[147] Piper Merriam and Jason Carver. Web3.py. `https://web3py.readthedocs.io/en/stable/`, 2018.

[148] Charles Miers, Guilherme Koslovski, Maurício Pillon, Marcos Simplício, Tereza Carvalho, Bruno Rodrigues, and João Battisti. Análise de Mecanismos para Consenso Distribuído Aplicados a Blockchain, September 2019.

[149] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The Honey Badger of BFT Protocols. In *The 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42, Vienna, Austria, October 2016.

[150] J. Mirkovic, G. Prier, and P. Reiher. Attacking DDoS at the Source. In *10th IEEE International Conference on Network Protocols (IEEE ICNP'02)*, pages 312–321, Paris, France, November 2002.

[151] Jelena Mirkovic and Peter Reiher. A Taxonomy of DDoS attack and DDoS Defense Mechanisms. *ACM SIGCOMM Computer Communication Review (ACM/CCR)*, 34(2):39–53, 2004.

[152] Jelena Mirkovic, Gregory Prier, and Peter Reiher. Attacking DDoS at the Source. In *10th IEEE International Conference on Network Protocols, 2002.*, pages 312–321, Paris, France, November 2002. IEEE.

[153] Axel Moinet, Benoît Darties, and Jean-Luc Baril. Blockchain Based Trust & Authentication for Decentralized Sensor Networks. *arXiv:1706.01730 [cs]*, June 2017. URL `http://arxiv.org/abs/1706.01730`.

[154] Susan Moore. Gartner Forecasts Worldwide Information Security Spending to Exceed 124 Billion in 2019. *Gartner*, 2018. URL https://gtnr.it/2T4DTKc.

[155] Steve Morgan. 2019 Official Annual Cybercrime Report. *Herjavec Group*, 2019. URL https://bit.ly/2TouUT2.

[156] Andrew Mortensen, Flemming Andreasen, Tirumaleswar Reddy, Christopher Gray, Rich Compton, and Nik Teague. Distributed-Denial-of-Service Open Threat Signaling (DOTS) Architecture. Internet-Draft draft-ietf-dots-architecture-06, Internet Engineering Task Force, March 2018. URL https://datatracker.ietf.org/doc/html/draft-ietf-dots-architecture-06. Work in Progress.

[157] Mark Mossberg, Felipe Manzano, Eric Hennenfent, Alex Groce, Gustavo Grieco, Josselin Feist, Trent Brunson, and Artem Dinaburg. Manticore: A User-Friendly Symbolic Execution Framework for Binaries and Smart Contracts, 2019. URL https://arxiv.org/pdf/1907.03890.pdf. arXiv.

[158] Bernhard Mueller. Smashing Ethereum Smart Contracts for Fun and Real Profit. URL: https://bit.ly/2T4VL8g, 2018.

[159] Francisco-Javier Moreno Muro, Nina Skorin-Kapov, and Pablo Pavon-Marino. Revisiting core traffic growth in the presence of expanding CDNs. *Computer Networks*, 154:1 – 11, 2019. ISSN 1389-1286. doi: https://doi.org/10.1016/j.comnet.2019.03.005. URL http://www.sciencedirect.com/science/article/pii/S1389128618309423.

[160] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008. URL https://bitcoin.org/bitcoin.pdf.

[161] Christopher Natoli, Jiangshan Yu, Vincent Gramoli, and Paulo Esteves-Verissimo. Deconstructing Blockchains: A Comprehensive Survey on Consensus, Membership and Structure. *arXiv preprint arXiv:1908.08316*, 2019.

[162] Jose Nazario. Politically Motivated Denial of Service Attacks. *The Virtual Battlefield: Perspectives on Cyber Warfare*, pages pp. 163–181, 2009.

[163] A10 Networks. 2017 DDoS of Things Survival Guide, 2017. URL https://www.a10networks.com/resources/white-papers/2017-ddos-things-survival-guide.

[164] NFV White Paper. Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. Issue 1. White Paper, October 2012.

[165] Ivica Nikolic. MAIAN. URL: https://github.com/ivicanikolicsg/MAIAN/, 2020.

[166] Ivica Nikolić, Aashish Kolluri, Ilya Sergey, Prateek Saxena, and Aquinas Hobor. Finding The Greedy, Prodigal, and Suicidal Contracts at Scale. In *The 34th Annual Computer Security Applications Conference (ACSAC '18)*, pages 653–663, San Juan, PR, USA, December 2018.

[167] Kaname Nishizuka, Liang Xia, Jinwei Xia, Dacheng Zhang, Luyuan Fang, Christopher Gray, and Rich Compton. Inter-organization Cooperative DDoS Protection Mechanism. Internet-Draft draft-nishizuka-dots-inter-domain-mechanism-02, Internet Engineering Task Force, December 2016. URL `https://datatracker.ietf.org/doc/html/draft-nishizuka-dots-inter-domain-mechanism-02`. Work in Progress.

[168] NIST. The Bugs Framework. URL: `https://samate.nist.gov/BF/index.html/`, 2016.

[169] George Oikonomou, Jelena Mirkovic, Peter Reiher, and Max Robinson. A Framework for a Collaborative DDoS Defense. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 33–42, Miami Beach, Florida, USA, December 2006. IEEE.

[170] Kelly Olson, Mic Bowman, James Mitchell, Shawn Amundson, Dan Middleton, and Cian Montgomery. Sawtooth: An introduction, 2018.

[171] Diego Ongaro and John Ousterhout. In Search of an Understandable Consensus Algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, Philadelphia, PA, USA, June 2014.

[172] OpenDaylight. OpenDaylight: A Linux Foundation Collaborative Project, 2013. URL `http://www.opendaylight.org`.

[173] N. Oza, F. Fagerholm, and J. Münch. How Does Kanban Impact Communication and Collaboration in Software Engineering Teams? In *6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE 2013)*, pages 125–128, San Francisco, CA, USA, May 2013. doi: 10.1109/CHASE.2013.6614747.

[174] Pierlugini Paganini. 150,000 IoT Devices Behind the 1Tbps DDoS attack on OVH, September 2016. URL `https://goo.gl/NYDgi9`.

[175] Christos Papadopoulos, Robert Lindell, John Mehringer, Alefiya Hussain, and Ramesh Govindan. COSSACK: Coordinated Suppression of Simultaneous Attacks. In *DARPA Information Survivability Conference and Exposition*, volume 1, pages 2–13, Washington, DC, USA, April 2003. IEEE.

[176] Kihong Park and Heejo Lee. On the Effectiveness of Route-based Packet Filtering for Distributed DoS Attack Prevention in Power-law Internets. *ACM SIGCOMM Computer Communication Review (ACM/CCR)*, 31(4):15–26, 2001.

[177] Kihong Park and Heejo Lee. On the Effectiveness of Probabilistic Packet Marking for IP Traceback under Denial of Service Attack. In *IEEE Conference on Computer Communications (INFOCOM 2001). Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, volume 1, pages 338–347, Anchorage, Alaska, USA, April 2001. IEEE.

[178] M-Elisabeth Paté-Cornell, Marshall Kuypers, Matthew Smith, and Philip Keller. Cyber Risk Management for Critical Infrastructure: a Risk Analysis Model and Three Case Studies. *Risk Analysis*, 38(2):pp. 226–241, 2018.

[179] Paul Sztorc. Truthcoin Whitepaper, 2015. URL `http://www.truthcoin.info/papers/truthcoin-whitepaper.pdf`.

[180] Adam Pavlidis, Marinos Dimolianis, Kostas Giotis, Loukas Anagnostou, Nikolaos Kostopoulos, Theocharis Tsigkritis, Ilias Kotinas, Dimitrios Kalogeras, and Vasilis Maglaris. Orchestrating DDoS Mitigation via Blockchain-based Network Provider Collaborations. *Journal of The Knowledge Engineering Review*, 35:1–16, 2020. doi: 10.1017/S0269888920000259.

[181] Vern Paxson. Bro: a System for Detecting Network Intruders in Real-time. *Computer networks*, 31(23-24):2435–2463, 1999.

[182] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Protection from Distributed Denial of Service Attacks using History-based IP Filtering. In *IEEE International Conference on Communications, 2003 (IEEE ICC'03)*, volume 1, pages 482–486, Anchorage, Alaska, USA, May 2003. IEEE.

[183] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Survey of Network-based Defense Mechanisms Countering the DoS and DDoS Problems. *ACM Computing Surveys (CSUR)*, 39(1):pp. 03–15, 2007.

[184] Colin Percival. Cryptographic Right Answers. `http://www.daemonology.net/blog/2009-06-11-cryptographic-right-answers.html`, Jun 2009.

[185] Jack Peterson and Joseph Krug. Augur: A Decentralized, Open-Source Platform for Prediction Markets. *arXiv preprint arXiv:1501.01042*, 2015. URL `https://arxiv.org/pdf/1501.01042.pdf`.

[186] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. The Design and Implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130, Oakland, CA, January 2015. USENIX Association. ISBN 978-1-931971-218. URL `https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff`.

[187] Douglas Pike, Patrick Nosker, David Boehm, Daniel Grisham, Steve Woods, and Joshua Marston. Proof-of-Stake-Time, 2018. URL `https://wiki.vericoin.info/index.php?title=Proof-of-Stake-Time`.

[188] Thomas Pohle. DDoS Attacks in the Second Quarter of 2019: Increasing Attack Bandwidths, Fev 2019. URL `https://bit.ly/2SVqDJm`.

[189] PolySwarm. A Decentralized Cyber Threat Intelligence Market, July 2019. URL `https://polyswarm.io/polyswarm-whitepaper.pdf`.

[190] Purathani Praitheeshan, Lei Pan, Jiangshan Yu, Joseph Liu, and Robin Doss. Security Analysis Methods on Ethereum Smart Contract Vulnerabilities: A Survey, 2019. URL `https://arxiv.org/abs/1908.08605`.

[191] Ilung Pranata, Geoff Skinner, and Rukshan Athauda. A Holistic Review on Rrust and Reputation Management Systems for Digital Environments. *International Journal of Computer and Information Technology*, 1:pp. 44–53, 2012.

[192] The Associated Press. Hackers Used 'Internet of Things' Devices to Cause Friday's Massive DDoS Cyberattack, Oct 2016. URL `http://www.cbc.ca/news/technology/hackers-ddos-attacks-1.3817392`.

[193] Rojalina Priyadarshini and Rabindra Kumar Barik. A Deep Learning based Intelligent Framework to Mitigate DDoS Attack in Fog Environment. *Journal of King Saud University - Computer and Information Sciences*, 2019. ISSN 1319-1578. doi: https://doi.org/10.1016/j.jksuci.2019.04.010. URL `http://www.sciencedirect.com/science/article/pii/S1319157818310140`.

[194] Gregor N. Purdy. *Linux Iptables - Kurz & Gut.* O'Reilly, 08 2004. ISBN 3897215063.

[195] Bahman Rashidi and Carol Fung. CoFence: A Collaborative DDoS Defence Using Network Function Virtualization. In *12th International Conference on Network and Service Management (CNSM 16)*, Montreal, Canada, November 2016.

[196] Remix. "Remix IDE Documentation. Solidity Static Analysis". URL: `https://remix-ide.readthedocs.io/en/latest/static_analysis.html/`, 2020.

[197] B. Rodrigues, L. Eisenring, E. Scheid, T. Bocek, and B. Stiller. Evaluating a Blockchain-based Cooperative Defense. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2019)*, pages 533–538, April 2019. Washington, United States of America (USA).

[198] Bruno Rodrigues and Burkhard Stiller. Cooperative Signaling of DDoS Attacks in a Blockchain-based Network. In *The ACM SIGCOMM 2019 Conference Posters and Demos*, SIGCOMM Posters and Demos '19, pages 39–41, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6886-5. doi: 10.1145/3342280.3342300. URL `http://doi.acm.org/10.1145/3342280.3342300`.

[199] Bruno Rodrigues and Burkhard Stiller. Cooperative Signaling of DDoS Attacks in a Blockchain-based Network. In *The ACM SIGCOMM 2019 Conference Posters and Demos*, pages 39–41, Beijing, China, August 2019. ACM. ISBN 978-1-4503-6886-5. doi: 10.1145/3342280.3342300. URL `https://dl.acm.org/citation.cfm?id=3342300`.

[200] Bruno Rodrigues, Thomas Bocek, Andri Lareida, David Hausheer, Sina Rafati, and Burkhard Stiller. 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*, pages 16–29. Springer, July 2017. Zürich, Switzerland.

[201] Bruno Rodrigues, Thomas Bocek, and Burkhard Stiller. Enabling a Cooperative, Multi-domain DDoS Defense by a Blockchain Signaling System (BloSS). In *Demonstration Track*, pages 1–3, Singapore, Singapore, October 2017. IEEE. URL `https://goo.gl/5TMFUt`.

[202] Bruno Rodrigues, Thomas Bocek, and Burkhard Stiller. Multi-Domain DDoS Mitigation Based on Blockchains. In *IFIP International Conference on Autonomous Infrastructure, Management and Security*, pages 185–190, Zürich, Switzerland, July 2017. Springer.

[203] Bruno Rodrigues, Thomas Bocek, and Burkhard Stiller. The Use of Blockchains: Application-driven Analysis of Applicability. In *Advances in Computers*, volume 111, pages 163–198. Elsevier, 2018.

[204] Bruno Rodrigues, Muriel Franco, Eder Scheid, Burkhard Stiller, and Salil Kanhere. A Technology-driven Overview on Blockchain-based Academic Certificate Handling. *IGI Global*, pages 1–290, jan 2020. doi: 10.4018/978-1-5225-9478-9. URL `https://www.igi-global.com/book/blockchain-technology-applications-education/221313`.

[205] Bruno Rodrigues, Muriel Figueredo Franco, Eder Scheid, Salil S Kanhere, and Burkhard Stiller. A Technology-driven Overview on Blockchain-based Academic Certificate Handling. In *Blockchain Technology Applications in Education*, pages 197–223. IGI Global, 2020.

[206] Bruno Rodrigues, Eder John Scheid, Christian Killer, Muriel Franco, and Burkhard Stiller. Blockchain Signaling System (BloSS): Cooperative Signaling of Distributed Denial-of-Service Attacks. *Journal of Network and Systems Management*, 28(3):1–27, August 2020. URL `https://doi.org/10.1007/s10922-020-09559-4`.

[207] Bruno Rodrigues, Spasen Trendafilov, Eder John Scheid, and Burkhard Stiller. SC-FLARE: Cooperative DDoS Signalingbased on Smart Contracts. In *IEEE International Conference on Blockchain and Cryptocurrency (ICBC 2020)*, pages 1–3, Toronto, Canada, May 2020. IEEE.

[208] Rishikesh Sahay, Gregory Blanc, Zonghua Zhang, and Hervé Debar. Towards Autonomic DDoS Mitigation Using Software Defined Networking, 2015.

[209] Rishikesh Sahay, Gregory Blanc, Zonghua Zhang, and Hervé Debar. Towards Autonomic DDoS Mitigation Using Software Defined Networking. In *SENT 2015: NDSS Workshop on Security of Emerging Networking Technologies*, San Diego, California. USA, February 2015. Internet Society.

[210] Mehrdad Salimitari and Mainak Chatterjee. A Survey on Consensus Protocols in Blockchain for IoT Networks. *arXiv preprint arXiv:1809.05613*, 2018. URL https://arxiv.org/pdf/1809.05613.pdf.

[211] N. F. Samreen and M. H. Alalfi. Reentrancy Vulnerability Identification in Ethereum Smart Contracts. In *IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE 2020)*, pages 22–29, London, Canada, February 2020.

[212] José Jair Santanna, Romain Durban, Anna Sperotto, and Aiko Pras. Inside Booters: An analysis on operational databases. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 432–440, Ottawa, Canada, May 2015. IEEE.

[213] José Jair Santanna, Roland van Rijswijk-Deij, Rick Hofstede, Anna Sperotto, Mark Wierbosch, Lisandro Zambenedetti Granville, and Aiko Pras. Booters—An Analysis of DDoS-as-a-service Attacks. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 243–251, Ottawa, Canada, May 2015. IEEE.

[214] Alexander Schaub, Rémi Bazin, Omar Hasan, and Lionel Brunie. A Trustless Privacy-Preserving Reputation System. In *ICT Systems Security and Privacy Protection*, IFIP Advances in Information and Communication Technology, pages 398–411, Ghent, Belgium, May 2016. Springer, Cham. ISBN 978-3-319-33629-9 978-3-319-33630-5. doi: 10.1007/978-3-319-33630-5_27. URL https://link.springer.com/chapter/10.1007/978-3-319-33630-5_27.

[215] E. Scheid, T. Hegnauer, B. Rodrigues, and B. Stiller. Bifröst: a Modular Blockchain Interoperability API. In *IEEE Conference on Local Computer Networks (LCN 2019)*, pages 332–339, Osnabrück, Germany, October 2019.

[216] E. Scheid, B. Rodrigues, and B. Stiller. Toward a Policy-based Blockchain Agnostic Framework. In *IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2019)*, pages 609–613, Washington - DC, USA, April 2019.

[217] Eder J. Scheid, Bruno B. Rodrigues, Christian Killer, Muriel F. Franco, Sina Rafati, and Burkhard Stiller. Blockchains and Distributed Ledgers Uncovered:Clarification, Achievements, and Open Issues. *IFIP's AICT 600*, 1(1):1–29, 2020.

[218] Andreas Schlosser, Marco Voss, and Lars Brückner. Comparing and Evaluating Metrics for Reputation Systems by Simulation. In *The IEEE Workshop on Reputation in Agent Societies*, September 2004. 00031.

[219] Dan Schnackengerg, Harley Holliday, Randall Smith, Kelly Djahandari, and Dan Sterne. Cooperative Intrusion Traceback and Response Architecture (CITRA). In *Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01*, volume 1, pages 56–68. IEEE, 2001.

[220] Fred B. Schneider. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Comput. Surv.*, 22(4):299–319, December 1990. ISSN 0360-0300. doi: 10.1145/98163.98167. URL https://doi.org/10.1145/98163.98167.

[221] Secure, Reliable, Intelligent Systems Lab ETH Zürich. Securify 2.0. URL: https://github.com/eth-sri/securify2/, 2020.

[222] Dongwon Seo, Heejo Lee, and Adrian Perrig. PFS: Probabilistic Filter Scheduling against Distributed Denial-of-Service Attacks. In *IEEE 36th Conference on Local Computer Networks (LCN 2011)*, pages 9–17, Bonn, Germany, October 2011. IEEE.

[223] Dongwon Seo, Heejo Lee, and Adrian Perrig. APFS: Adaptive Probabilistic Filter Scheduling against Distributed Denial-of-Service Attacks. *Computers & security*, 39:366–385, 2013.

[224] Mike Sharples and John Domingue. The Blockchain and Kudos: A Distributed System for Educational Record, Reputation and Reward. In *Adaptive and Adaptable Learning*, Lecture Notes in Computer Science (LNCS), pages 490–496. Springer, Cham, September 2016. ISBN 978-3-319-45152-7 978-3-319-45153-4. doi: 10.1007/978-3-319-45153-4_48. URL https://link.springer.com/chapter/10.1007/978-3-319-45153-4_48.

[225] Sérgio SC Silva, Rodrigo MP Silva, Raquel CG Pinto, and Ronaldo M Salles. Botnets: A Survey. *Computer Networks*, 57(2):pp. 378–403, 2013.

[226] Isabel Skierka, Robert Morgus, Mirko Hohmann, and Tim Maurer. CSIRT Basics for Policy-Makers. *The History, Types & Culture of Computer Security Incident Response Teams*, 2015.

[227] Fabio Soldo, Katerina Argyraki, and Athina Markopoulou. Optimal Source-based Filtering of Malicious Traffic. *IEEE/ACM Transactions on Networking*, 20(2):381–395, 2011.

[228] Kyle Soska, Albert Kwon, Nicolas Christin, and Srinivas Devadas. Beaver: A Decentralized Anonymous Marketplace with Secure Reputation. *IACR Cryptology ePrint Archive*, 2016:464, 2016.

[229] Georgios Spathoulas, Nikolaos Giachoudis, Georgios-Paraskevas Damiris, and Georgios Theodoridis. Collaborative Blockchain-Based Detection of Distributed Denial of Service Attacks Based on Internet of Things Botnets. *Future Internet*, 11(11):226, 2019.

[230] Diane Staheli, Tamara Yu, R. Jordan Crouser, Suresh Damodaran, Kevin Nam, David O'Gwynn, Sean McKenna, and Lane Harrison. Visualization Evaluation for Cyber Security: Trends and Future Directions. In *11th Workshop on Visualization for Cyber Security (VizSec 2014)*, pages 49–56, New York, NY, USA, November 2014. ISBN 978-1-4503-2826-5. doi: 10.1145/2671491.2671492. URL http://doi.acm.org/10.1145/2671491.2671492.

[231] SteemIT. A Protocol for Enabling Smart, Social Currency for Publishers and Content Businesses Across the Internet, 2017. URL https://steem.com/steem-bluepaper.pdf.

[232] J. Steinberger, B. Kuhnert, A. Sperotto, H. Baier, and A. Pras. Collaborative DDoS Defense Using Flow-based Security Event Information. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 516–522, Istanbul, Turkey, April 2016. doi: 10.1109/NOMS.2016.7502852.

[233] Dan Sterne, Kelly Djahandari, Ravindra Balupari, William La Cholter, Bill Babson, Brett Wilson, Priya Narasimhan, Andrew Purtell, Dan Schnackenberg, and Scott Linden. Active Network based DDoS Defense. In *DARPA Active Networks Conference and Exposition*, pages 193–203, San Francisco, California, USA, May 2002. IEEE.

[234] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet Indirection Infrastructure. *IEEE/ACM Transactions on networking*, 12(2):205–218, 2004.

[235] Truffle Suite. Ganache - One Click Blockchain, Aug 2020. URL `https://www.trufflesuite.com/ganache`.

[236] Nick Szabo. Formalizing and Securing Relationships on Public Networks. *First Monday*, 2 (9), 1997.

[237] Michael Szydlo. Merkle Tree Traversal in Log Space and Time. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 541–554, Interlaken, Switzerland, May 2004. Springer.

[238] Keisuke Takemori, Masakatsu Nishigaki, Tomohiro Takami, and Yutaka Miyake. Detection of Bot Infected PCs using Destination-based IP and Domain Whitelists During a Non-operating Term. In *IEEE Global Telecommunications Conference (IEEE GLOBECOM 2008*, pages 1–6, New Orleans, LA, USA, December 2008. IEEE.

[239] David L Tennenhouse, Jonathan M Smith, W David Sincoskie, David J Wetherall, and Gary J Minden. A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1): 80–86, 1997.

[240] Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, and Kevin Gibbs. iPerf: TCP/UDP Bandwidth Measurement Tool, 01 2005. URL `https://iperf.fr/`.

[241] FUJITA Tomonori. Introduction to Ryu SDN Framework. *Open Networking Summit*, 2013. URL `https://ryu-sdn.org/slides/ONS2013-april-ryu-intro.pdf`.

[242] Trail of Bits. Manticore. URL: `https://github.com/trailofbits/manticore/`, 2020.

[243] Spasen Trendafilov. Cooperative Signaling Protocol, Aug 2019. URL `https://files.ifi.uzh.ch/CSG/staff/rodrigues/extern/theses/ba-spasen.pdf`.

[244] Petar Tsankov. Release of Securify v2.0. URL: `https://medium.com/chainsecurity/release-of-securify-v2-0-6304a40034f/`, January 2020.

[245] Petar Tsankov, Andrei Dan, Dana Drachsler Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. Securify: Practical Security Analysis of Smart Contracts, 2018.

[246] Tai-Won Um, Gyu Myoung Lee, and Jun Kyun Choi. Strengthening Trust in the Future Social-Cyber-Physical Infrastructure: an ITU-T Perspective. *IEEE Communications Magazine*, 54: pp. 36–42, 2016.

[247] Thaneswaran Velauthapillai, Aaron Harwood, and Shanika Karunasekera. Global Detection of Flooding-based DDoS Attacks Using a Cooperative Overlay Network. In *Network and System Security (NSS), 2010 4th International Conference on*, pages pp. 357–364, Melbourne, Australia, September 2010. IEEE.

[248] Michael Walfish, Mythili Vutukuru, Hari Balakrishnan, David Karger, and Scott Shenker. DDoS defense by offense. In *The ACM SIGCOMM 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 303–314, Pisa, Italy, September 2006.

[249] Michael Walfish, Mythili Vutukuru, Hari Balakrishnan, David Karger, and Scott Shenker. DDoS Defense by Offense. *ACM Transactions on Computer Systems (TOCS)*, 28(1):1–54, 2010.

[250] Haining Wang, Cheng Jin, and Kang G Shin. Defense Against Spoofed IP Traffic using Hop-count Filtering. *IEEE/ACM Transactions on networking*, 15(1):40–53, 2007.

[251] Y. Wang and J. Vassileva. Trust and Reputation Model in Peer-to-Peer Networks. In *The Third International Conference on Peer-to-Peer Computing (P2P2003)*, pages 150–157, Linkoeping, Sweden, September 2003. doi: 10.1109/PTP.2003.1231515.

[252] WhiteHouse. The Cost of Malicious Cyber Activity to the U.S. Economy. *White House*, 2018. URL `https://www.whitehouse.gov/wp-content/uploads/2018/03/The-Cost-of-Malicious-Cyber-Activity-to-the-U.S.-Economy.pdf`.

[253] Gavin Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger. *Ethereum Project Yellow Paper*, 151:pp. 1–32, 2014. URL `https://gavwood.com/paper.pdf`.

[254] workNation. Work.nation: Decentralized skill attestations using uPort, Ethereum and IPFS, October 2017. URL `https://github.com/worknation/work.nation`.

[255] Yang Xiao, Ning Zhang, Wenjing Lou, and Y Thomas Hou. A Survey of Distributed Consensus Protocols for Blockchain Networks. *IEEE Communications Surveys & Tutorials*, 2020.

[256] Li Xiong and Ling Liu. Building Trust in Decentralized Peer-to-Peer Electronic Communities. *International Conference on Electronic Commerce Research (ICECR-5)*, February 2003. 00172.

[257] Tengfei Xue, Yuyu Yuan, Zahir Ahmed, Krishna Moniz, Ganyuan Cao, and Cong Wang. Proof of contribution: A Modification of Proof of Work to Increase Mining Efficiency. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 636–644, Tokyo, Japan, July 2018. IEEE.

[258] Abraham Yaar, Adrian Perrig, and Dawn Song. Pi: A Path Identification Mechanism to Defend Against DDoS Attacks. In *IEEE Symposium on Security and Privacy (IEEE S&P 2003)*, pages 93–107, Berkeley, California, USA, May 2003. IEEE.

[259] Zheng Yang and Hang Lei. FEther: An Extensible Definitional Interpreter for Smart-Contract Verifications in Coq. *IEEE Access*, 7:37770–37791, 2019. ISSN 2169-3536. doi: 10.1109/access.2019.2905428. URL `http://dx.doi.org/10.1109/ACCESS.2019.2905428`.

[260] Fasheng Yi, Shui Yu, Wanlei Zhou, Jing Hai, and Alessio Bonti. Source-based Filtering Scheme Against DDoS Attacks. *International Journal of Database Theory and Application*, 1(1):9–20, 2008.

[261] Jie Yu, Zhoujun Li, Huowang Chen, and Xiaoming Chen. A Detection and Offense Mechanism to Defend Against Application Layer DDoS Attacks. In *International Conference on Networking and Services (ICNS'07)*, pages 54–54, Athens, Greece, June 2007. IEEE.

[262] Jie Yu, Chengfang Fang, Liming Lu, and Zhoujun Li. A Lightweight Mechanism to Mitigate Application Layer DDoS Attacks. In *International Conference on Scalable Information Systems (INFOSCALE'09)*, pages 175–191, Hong Kong, June 2009. Springer.

[263] Shui Yu, Wanlei Zhou, Robin Doss, and Weijia Jia. Traceback of DDoS Attacks using Entropy Variations. *IEEE Transactions on Parallel and Distributed Systems*, 22(3):412–425, 2010.

[264] Yu Zhang and Mihaela van der Schaar. Reputation-based Incentive Protocols in Crowdsourcing Applications. *arXiv:1108.2096 [physics]*, August 2011. URL `http://arxiv.org/abs/1108.2096`.

[265] Saman Taghavi Zargar, James Joshi, and David Tipper. A Survey of Defense Mechanisms against Distributed Denial of Service (DDoS) flooding attacks. *IEEE Communications Surveys & Tutorials*, 15(4):pp. 2046–2069, 2013.

[266] Guangsen Zhang and Manish Parashar. Cooperative Defence Against DDoS Attacks. *Journal of Research and Practice in Information Technology*, 38(1):pp. 69–84, 2006.

[267] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. In *2017 IEEE International Congress on Big Data (BigData Congress)*, pages 557–564, Honolulu, HI, USA, June 2017. IEEE.

[268] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. Blockchain Challenges and Opportunities: A Survey. *International Journal of Web and Grid Services*, 14 (4):pp. 352–375, 2018.

A
Description of DDoS Cooperative Mechanisms

This appendix presents details of each work presented in the different categories in the Chapter 3. Sections are organized according to the DDoS mechanism placement in the network. Section A.1 presents source-based, Section A.2 presents destination-based mechanisms. Section A.3 describe network-based. Lastly, Section A.4 detail hybrid DDoS mechanisms.

A.1 Source-based DDoS Mechanisms

MANANet [140]

The MANANet system [140] works like a reverse firewall preventing DDoS attacks by limiting the rate of "unexpected" TCP packets at a network's exit router. Thus, MANANet derives from previously sent traffic the sequence numbers of expected packets. Only outgoing packets matching the expected sequence number range will be forwarded. MANAnet can be used in a cooperative scenario where all participating routers are compatible with exchanging additional path information. However, it is a commercial solution, and details about its operation are not fully disclosed.

DDoS netWork Attack Recognition and Defense (D-WARD)

D-WARD is a source-based defense strategy that monitors the outgoing traffic and compares it to pre-defined traffic models, which helps the system decide whether a specific flow includes malicious traffic [150, 151]. The D-WARD system is installed on multiple systems. However, there exists no coordination among these systems, which makes D-WARD a centralized approach. As soon as a network flow being monitored by D-WARD is detected to be of malicious origin, the traffic is rate-limited to stifle the attack's effectiveness [151]. Rate-limiting the flows allows ongoing monitoring to lift the rate limit as soon as the flow ceases to exhibit malicious characteristics. This is preferable to outright stopping the flows since wrong classifications cannot be detected if continuous monitoring of the respective flows is unavailable. As soon as a network flow being monitored by D-WARD is detected to be of malicious origin, the traffic is rate-limited to stifle the effectiveness of the attack [151]. Rate-limiting the flows allows ongoing monitoring to lift the rate limit as soon as the flow ceases to exhibit malicious characteristics. This is preferable to outright stopping the flows since misclassifications cannot be detected if continuous monitoring of the respective flows is unavailable.

MUlti-Level Tree for Online Packet Statistics (MULTOPS) [76]

The MUlti-Level Tree for Online Packet Statistics (MULTOPS) data structure pursues a similar strategy as D-WARD by analyzing specific characteristics of the ingressing and egressing traffic of an AS [76]. Instead of gathering statistics and finding patterns that match a pre-defined model, MULTOPS is an attack-resistant data structure storing statistics about aggregated traffic to and from the AS it is running on. These statistics are then used to find significant differences between the amount of incoming and outgoing traffic, which would indicate an attack going in one direction. This is based on the assumption that benign traffic exhibits symmetric characteristics for incoming and outgoing traffic since the other host [76] generally acknowledges packets from one host.

Click Router [113]

[76] implemented MULTOPS as a module of the Click router [113]. The Click router's advantage is the ability to chain multiple modules, so-called "elements" together to form a processing pipeline for incoming packets. Each element only performs simple functions such as "communicating with devices, queuing packets, and implementing a dropping policy "[76]. The MULTOPS implementation contains two elements called "IPRateMonitor" and "RatioBlocker", which are directly interconnected so that the rate of traffic for incoming and outgoing packets is monitored. Based on these monitored rates, packets are dropped if the ratio is out of balance according to the pre-defined thresholds.

Lu, Yiqin and Wang, Meng [136]

[136] proposes a new detection algorithm based on the statistical inference model and response scheme through the abilities of Software-Defined Networking (SDN) and sFlow, a tool to sample SDN flows. Taking advantage of SDN's global network view, the authors define a metric to detect coordinated actions between bots within its network, the DCD (Distribution-Collaboration Degree). The DCD metric aims to quantify the degree of distribution and collaboration of flow on one dimension and quantify this flow's intensity on another dimension. While the sFlow works as a monitoring tool collecting samples of flows, the metric is implemented at the SDN controller. In summary, the authors detect in a sliding time window when many flows are installed targeting the same destination, an activity that is classified as an attack.

Rojalina Priyadarshini and Rabindra Kumar Barik [193]

An approach also based on SDN but using machine learning to detect traffic anomalies are presented in [193]. The intended scenario is Fog computing, where the SDN detection system is implemented at the edge. Priyadarshini's system uses the Floodlight controller (Java-based SDN Controller widely used for prototyping) and a machine learning module to distinguish anomalies in traffic; the authors use a learning model based in a Long Short Term Memory (LSTM) network. Thus, the controller performs a periodic pooling on the switches to collect flow data, updating the statistics table used to verify, in the machine learning model, if the traffic pattern corresponds to attack patterns. However, similarly to the other proposals, this approach imposes a computational overhead on the controller to verify whether the traffic patterns correspond to attacking patterns at each monitoring period. Besides, the proposal fits as an enabler of cooperative solutions to detect patterns in multiple edges.

Yi, Fasheng and Yu, Shui and Zhou, Wanlei and Hai, Jing and Bonti, Alessio [260]

[260] approaches source-based filtering as with a different angle by considering the issue of IP address spoofing as a support tool for cooperative mechanisms. The authors accumulate the source information of its clients, such as source IP address, and the number of hops from the server (*i.e.*, the TTL) to construct a table based on the statistics of source IP addresses under hops from that source IP addresses to the server in reasonable condition. Then, during an attack, a possible verification between collaborating organization could be based on matching destination with source information to check whether sources were spoofed. While [260] can be seen as an enabler towards cooperative defenses by working independently at the potential victim side, depending on the number of legitimate customers, the system may require large volumes of storage and require efficient address verification schemes.

Soldo, Fabio and Argyraki, Katerina and Markopoulou, Athina [227]

A similar approach is proposed by [227], in which the authors consider an Access Control List (ACL) filtering approach. ACL-based approaches can create white or blacklists of IP addresses, storing these in the memory of routers, to allow or deny external access. The authors focus on the proposal of a method for optimizing the memory space used by these ACLs by optimally selecting which source prefixes to filter for various realistic attack scenarios and operators' policies. Even if the authors seek to make address storage more efficient, detecting attackers' central problem (or differentiating those from legitimate users) remains. This highlights the central problem of efficiently detecting DDoS attacks involving thousands of devices that often have traffic patterns similar to legitimate users.

Badis, Hammi and Doyen, Guillaume and Khatoun, Rida [15]

A source-based and collaborative approach is proposed by [15]. Although the authors state that their proposal is collaborative, such collaboration takes place in a cloud system operated by a single organization, *i.e.*, there is coordination for the exchange of information between traffic flow within the cloud. The approach relies on an overlay network of hypervisor-level Intrusion Detection Systems (IDS) to detect coordinated traffic anomalies (*i.e.*, sudden burst traffic to a single destination). Therefore, the approach is similar to Lu *et al.* [136] and Yi *et al.* [260] detecting bots operating the network perimeter but expanding this detection point to multiple servers on which the hypervisor is running.

A.2 Destination-based DDoS Mechanisms

Management Information Base (MIB) [29]

MIB data contains statistical variables about the traffic in a Network Management System (NMS) [29]. These variables are organized in IP, ICMP, TCP, UDP, and SNMP, generally grouping the variables with the corresponding protocols [29]. The goal in utilizing the MIB for destination-based DDoS prevention is to deduce patterns before a DDoS attack based on the attacker's preparations to instruct the slave machines utilized in a DDoS attack [29]. This represents a much more efficient approach toward DDoS mitigation, especially considering the high volume of attack traffic destination-based approaches typically need to defend. By setting up such an online monitoring solution in the form of an Intrusion Detection System (IDS) for DDoS attacks, blocking attackers can already occur throughout the attack's preparation stages instead of after the attack is already taking place and wasting resources [29].

This scheme can already be applied for destination-based mitigation where the NMS defending against the attack only manages the DDoS target. However, if attackers are also part of the managed domain, additional instructive traffic between the attack master and the attack slaves can be analyzed to generate pointers for an early warning system. The proactive detection of attacks works in three steps [29]:

1. The first step is concerned with collecting information about possible MIB variables suitable to act as precursors for an imminent attack at a target host. These variables can be found by analyzing traffic data from past attacks and determining patterns based on the available MIB variables.

2. If attack hosts are within the NMS domain being protected, correlations between MIB variables for the attack hosts and the variables for the target hosts from step 1 can be calculated.

3. These correlations can then be used to increase the significance of the precursors found in step 1 to react to imminent attacks even faster.

Path identifier (Pi) [258]

Path identifier (Pi) [258] is a consistent approach based on packet marking toward fingerprinting the path of packets traversing the Internet. In this sense, the approach is not concerned with reconstructing the path of marked packets but providing the packet marking infrastructure in which a victim under attack could implement another approach to reconstruct the path and distinguish legitimate users from attackers. This allows a victim to employ the Pi mark to filter out packets matching the attackers' identifiers on a per-packet basis, which is considered a proactive role in defending against a DDoS attack [265]. However, Pi's relies on having its solution implemented on at least half of the Internet's routers to be effective, which can be considered a significant drawback of the solution.

Takemori, Keisuke and Nishigaki, Masakatsu and Takami, Tomohiro and Miyake, Yutaka [238]

[238] proposes a bot detection technique that checks outbound packets with destination based whitelists. Whitelists are generated by observing not infected computers (*i.e.*, the solution is intended for Personal Computers - PC) during the PC's non-operating time, in which a daemon is running. When a PC sends during the non-use time traffic to destinations outside the whitelist, it is considered infected. Whitelists are populated with IP addresses of public services (*e.g.*, DNS servers, patch servers, antivirus servers), and after the first installation of the operating system, the learning period takes place. After the learning period, packets that do not match any whitelist entry are dropped.

Hop-Count Filtering (HCF) [99, 250]

Hop-Count Filtering (HCF) 's main idea is to detect spoofed packets by looking at the number of hops contained in the packets [99, 250]. This information is inferred from the Time-To-Live (TTL) field in the IP header, which cannot be forged by attackers. HCF requires a packet-level inspection system based on an IP hop-count mapping table, which in cases of large-scale DDoS attacks may overhead the detection system by increasing the mapping table. Although HCF can be deployed in a collaborative context, HCF itself does not propose a collaborative approach to alleviate the destination's detection and mitigation efforts.

Yu, Shui and Zhou, Wanlei and Doss, Robin and Jia, Weijia [263]

[263] proposes a traceback approach using flow entropy variations between normal and DDoS attack traffic, which varies from standard used packet marking techniques since most solutions are either based on probabilistic or deterministic packet marking (PPM or DPM, respectively). The approach proposed by Yu *et al.* [263] requires that routers store information about flow variations check whether those variations correspond to an attack pattern. Thus, a victim should identify which upstream routers are in the attack tree, and based on the flow entropy variations *i.e.*, sequence of packets between source and destination, submit mitigation requests to those upstream routers. Despite requiring less overhead on routers than traditional PPM or DPM for storing information on variations in flow, large-scale DDoS attacks tend to considerably undermine the performance of solutions based on the on-packet-level analysis.

Packetscore [111]

Packetscore [111] is a destination-based approach focusing on automated attack profiling and discarding. The key idea is to prioritize packets that are legitimately recognized (*i.e.*, whitelisted sources) by increasing their points in a scoring system and defining low priorities for packets of unknown sources. In case of an attack and, once the score of a packet is computed, Packetscore can perform the selective packet discarding dynamically adjusting the dropping threshold based on the score of new incoming packets and the current overload of the system. However, packet score calculation is not an optimal solution for large-scale attacks. Such attacks would require from the detection system an enormous processing capacity in the network equipment, becoming infeasible in attacks of great magnitude such as the Botnet-based ones (*e.g.*, Mirai DynDNS attack [104]).

Peng, Tao and Leckie, Christopher and Ramamohanarao, Kotagiri [182]

[182] present History-based IP Filtering (HIF), an approach similar to [238], but applying a white-list of IP addresses in routers instead of PCs. An edge router keeps the history of legitimate IP addresses previously appeared in the network, and when the router is overloaded, the history is used to determine whether to forward or discard packets. HIF's strategy is relatively simple since edge routers (deployed at the destination or DDoS target) generally have more spare CPU time per packet than core routers, and their emphasis on the use of heuristics to keep the history up to date. Nonetheless, the drawback is similar to Packetscore [111] in the sense that HIF's approach falls short in large-scale attacks with traffic patterns similar to legitimate users is the case for Botnets based on thousands of IoT-based bots (*e.g.*, Mirai).

FlowGuard [98]

FlowGuard [98] presents a method to identify and classify IoT traffic at the network edge, *i.e.*, servers deployed at the edge of the IoT network can detect whether inbound traffic is an attack. FlowGuard builds upon two significant components: Flow Filter and Flow Handler. While the former maintains flow filtration rules generated by the Flow Handler, the latter analyzes suspicious flows for DDoS attack identification and classification and filtration rule generation using self-evolving machine learning algorithms.

Cziva, Richard and Pezaros, Dimitrios [54]

The use of resource virtualization allows the most efficient use of hardware. Network Function Virtualization (NFV) [164] can provide an efficient solution to the deployment conundrum by allowing the mitigation system to be encapsulated as Virtualized Network Functions (VNF), which can directly be deployed on commodity hardware running on-site. In this sense, [54] propose VNF as a similar approach to IP traceback to distribute network functions in mitigating DDoS attacks. For instance, such NFV could block malicious traffic by creating a new iptables firewall and setting up DROP rules on the selected traffic. Advantages are the high degree of isolation of this approach and the minimal code necessary to conduct the mitigation, therefore, directly contributing to a certain degree of trust by implicitly ensuring that the correct code is executed.

Cloud-based Protection Services [3, 43]

Another typical 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 [3] or CloudFlare [43]). These cloud-based protection services function as a typical destination-based mechanism (*e.g.*, deploying intrusion detection systems, firewalls, and other tools) but contain an infrastructure with a higher capacity for detecting and mitigating attacks. 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 [104].

A.3 Network-Based DDoS Mechanisms

Distributed Packet Filtering (DPF) [176]

Route-based packet filtering [176] or marking [177] are two popular approaches deployed in network-based mechanisms. DPF (Distributed Packet Filtering) is proposed in [176], as a proactive approach to filter out a significant fraction of spoofed packet flows and prevent attack packets from reaching their targets in the first place. The approach uses routing information to determine whether packets arriving at a router are valid concerning its inscribed source and destination addresses, given the network topology's reachability constraints.

Probabilistic Packet Marking (PPM) [177]

The route-based packet marking, *i.e.*, a Probabilistic Packet Marking (PPM) [177] occurs at the forwarding and discarding of packets based on either a table look-up and a filter table update. This corresponds to probabilistically "sampling" the route undertaken by an attack based on a constant space in the packet header independent of hop count, which provides the critical advantage over deterministic packet marking. The authors show that the probabilistic packet suffers under IP spoofing by attackers and exists a trade-off between the ability to traceback attackers and the severity of a DDoS attack. Thus, while PPM is useful against single-source attacks, it is potentially vulnerable when subject to large-scale DDoS attacks once these are typically originated from many sources.

Probabilistic Filter Scheduling (PFS) [222]

The Probabilistic Filter Scheduling (PFS) approach [222] is a variation of the PPM [177] to identify and create traffic filters. The PFS filter routers identify attack paths using probabilistic packet marking and maintain filters using a scheduling policy to maximize defense effectiveness. Thus, PFS assumes points of trust *i.e.*, routers in which PFS-enabled routers are installed over intermediate networks to mark packets. A filtering router writes its IP address to the IP header of packets, so a path can be determined by identifying the filter routers it has passed through. Then, PFS can propagate filters to the optimal routers located closer to the attack source and avoiding less attacking traffic to traverse the Internet until its target.

Adaptive Probabilistic Filter Scheduling (APFS) [223]

Adaptive Probabilistic Filter Scheduling (APFS) [223] extends its prior work PFS by combining the PPM [177] approach. The adaptability from APFS is calculated based on (1) hop count from a sender, (2) the filter router's resource availability, and (3) the filter router's link degree. Based on the effectiveness (*i.e.*, how accurate those three points are calculated), it would lead a victim to receive more markings from more effective filter routers, and thus, APFS would show a faster filter propagation than the hop-by-hop basis. Thus, APFS requires cooperation between APFS-enabled routers to install filters close to the source of the attack adaptively source.

Active Networking [233]

Based on the concept of active networks from the 90's [233] propose an automated defense against DDoS attacks. Active networks are also one of the predecessor concepts of SDN, describing a network in which infrastructure nodes (switches or routers) and end hosts serve as platforms for the execution of task-specific programs [239]. Thus, the approach's fundamental concept is to use active networks to distribute intrusion detection and response rules once a destination-based system detects an attack. Then, similarly to IP traceback mechanisms [100, 263] rules can be installed on compatible network devices to mitigate the closest possible attack to the source.

Cooperative Intrusion Traceback and Response Architecture (CITRA) [219]

Aligned with the proposal for cooperation between network-based routers is CITRA (Cooperative Intrusion Traceback and Response Architecture) [219]. CITRA provides an example of a cooperative agent-based system that enables interaction between different intrusion detection systems, routers, and firewalls. Similar to APFS [223], CITRA's architecture utilizes neighborhood structure where the information about detected intrusion is propagated back through the neighborhood to the source of the attack and submitted to the centralized authority.

Slow HTTP DDoS Defense Application (SHDA)

A network-based DDoS defense method is proposed in [90], namely Slow HTTP DDoS Defense Application (SHDA) targeting slow HTTP attacks. The SHDA is proposed as an application running on top of an SDN controller, analyzing requests web server requests, *i.e.*, flow counters toward a webserver. SDHA analyses suspicious incomplete HTTP requests from attackers based on a timeout-based DDoS detection, it requests the SDN controller to update the flow rules to block the attacker's flow at the network switches. Although the software-only approach facilitates its deployment on network equipment, it is possible to export flows to a generic server; there is no cooperation between multiple SDHA applications in the network for a global view of the attack.

MiddlePolice [132]

MiddlePolice [132] is a mechanism that works by receiving input from destination-based mechanisms to effectively enforce destination-chosen policies, while requiring no deployment from unrelated parties. In this sense, MiddlePolice operates as an environment where systems under attack can request to deploy mitigation policies on demand, combining cloud-based solutions (*e.g.*, Akamai and CloudFlare [3, 43]) with the destination-based control of capability-based systems. Further, the system uses a set of traffic policing units (referred to as *mboxes* by the authors), relies on a feedback loop of self-generated capabilities to guide scheduling and filtering.

A.4 Hybrid DDoS Defense Mechanisms

Secure Overlay Services (SOS) [103]

The overlay network proposed in [103], termed Secure Overlay Services (SOS), was developed considering the challenges of communicating P2P systems to provide security services. The system combines secure overlay tunneling, routing via consistent hashing, and filtering capabilities on the peers. In particular, SOS prevents members of the overlay network from performing DoS attacks on the channels (*i.e.*, malicious members would prevent honest communication by oversharing information), by proposing a proactive filtering mechanism on nearby networks.

Technical: Software-based approach whereas the communication between peers is tunneled.
Social: Participating members are considered trusted and are authenticated.
Economical: Not addressed.
Legal: Partially addressed once members are considered to be trusted.
Year: 2002.

Pushback [95, 139]

Pushback [95, 139] is a solution that is based on a similar (packet-marking) proposal but including mitigation functions for dropping packets. Pushback works rely on an 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 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.

Technical: Hardware-based approach requiring support on routers.
Social: Members are considered trusted.
Economical: Not addressed.
Legal: Partially addressed once members are trusted.
Year: 2002.

Coordinated Suppression of Simultaneous Attacks (COSSACK) [175]

An overlay network built on border routers of the edge networks based on assumptions that border routers need to have ingress/egress filtering mechanisms and prevent IP spoofing [175]. COSSACK is based on two main components, an IDS (Snort was used as a proof-of-concept) and a watchdog. While the IDS collects traffic statistics for different (including suspect) flows, the watchdog receives data from the IDS and applies filters on the routers when the victim detects an attack, the watchdog multi-casts attack notification to the watchdogs at the source networks to suppress the attack close to the source.

Technical: Hardware-based approach requiring support on border routers and IDS system.
Social: Members are considered trusted.
Economical: Not addressed.
Legal: Partially addressed once members are trusted.
Year: 2003.

Mayday - Distributed Filtering for Internet Services [8]

[8] combines overlay networks with lightweight packet filtering to defend against DDoS attacks. The overlay network generalizes the SOS [103] approach (with a similar design choice of authenticator and overlay routing), creating a secure network of authenticated and verified peers. Based on such generalization, Mayday supports different schemes using trusted or semi-trusted nodes to differentiate between legitimate and malicious clients, thus providing different balances of security and performance. One of the main benefits of the architecture proposed in Mayday is its flexibility, allowing different instances of Mayday to trade security for performance to create a better system that matches their needs.

Technical: Hardware-based approach similar (and generalizing) the SOS approach.
Social: Support different schemes of trusted and untrusted peers.
Economical: Not addressed.
Legal: Possible to define circles of trusted peers.
Year: 2003.

Internet Indirection Infrastructure (i3) [234]

The Internet Indirection Infrastructure (i3) was proposed in 2004 as a general overlay network (*i.e.*, not only to counter DDoS attacks), offering a rendezvous-based communication abstraction [234]. Instead of explicitly sending a packet to an end-host, each packet is associated with an identifier, which is then used by the receiver to obtain delivery of the packet. The overlay is composed of nodes that store triggers and forward packets (using IP) between i3 nodes and end-hosts.

Technical: Software-based approach proposed as a general overlay network.
Social: Not addressed, no reputation or trust schemes defined.
Economical: Not addressed.
Legal: Interaction with all subscribed peers.
Year: 2004.

Koutepas, Georgios and Stamatelopoulos, Fotis and Maglaris, Basil [118]

[118] propose a cooperative intrusion detection and reaction (*i.e.*, mitigation) framework anti DDoS attacks. The proposed approach relies on a trusted community that may cooperate on free will by exchanging security information on possible DDoS attacks. At this point, the assumption that all members must be trusted can limit the applicability of the solution in different organizations with similar

objectives within a consortium. In addition, free-will-based cooperation allows problems with the misuse of the collaborative platform by free-riders. On a technical basis, the main building component of [118] solution is the Cooperative anti-DDoS Entity, a software system deployed in each participating network domain that supports secure message exchanges and local responses tailored to individual sites' policies.

Technical: Software-based approach proposed as a general overlay network of IDS's.
Social: Based on trusted members.
Economical: Not addressed.
Legal: Partially addressed once members are trusted.
Year: 2004.

Active Internet Traffic Filtering - AITF [10]

AITF [10] is proposed as a distributed filtering protocol to counter large-scale DDoS attacks that leverage recorded route information to block attack traffic. A record route is an option that routers can insert their IP address into each packet, enabling them to trace back packets' origin. Like source-based mechanisms, a significant challenge lies in distributing filters in a scalable way among AITF peers. In this regard, AITF attempts to install filters as close as possible to the traffic sources by implementing an IP traceback system [263] that stamps their IP addresses to each packet. Filters are proposed based on hardware ACL (Access Control Lists) implemented by each peer to match white or blacklisted network flows.

Technical: Hardware-based approach proposed as a general overlay network.
Social: Members are considered to be trusted.
Economical: Mention incentives for deployment at edges but based on free will.
Legal: Partially addressed once members are trusted.
Year: 2005.

DefCOM [151]

It is one of the significant proposals toward cooperative network defenses. It proposes an overlay network based on a Peer-to-Peer (P2P) gossip-based protocol to facilitate coordination among peers with inherent scalability. DefCOM is specifically geared toward protection against flooding DDoS attacks and focuses on three critical defense functionalities [169]:

- Differentiating between benign and attack traffic through traffic classification
- Rate-limiting attack traffic to free resources
- Alert generation to signal all members of the cooperative defense about the IP address of the attack target and rate limits required to resolve traffic bottlenecks for the attack target

The strength of the DefCOM system lies in the highly distributed and scalable overlay network used for communication. However, the overlay network is only used for control messages; data packets still travel on the data links defined by the underlying routing protocols [169]. For traffic classification purposes, they used D-WARD [151] but deactivated all attack mitigation functionality of D-WARD to reduce it to a simple traffic classification engine. By utilizing the existing source-based D-WARD classifier and implementing it in an overlay network, communication among multiple peers can be leveraged to make the overall mitigation system much more efficient than the centralized approach.

Technical: Software-based overlay network geared toward scalability.
Social: Members are considered to be trusted.
Economical: Not addressed.
Legal: Partially addressed once members are trusted.
Year: 2006.

Zhang and Parashar [266]

An overlay network to detect and mitigate DDoS attacks is presented in [266]. It consists of two key stages to exchange attack information between independent detection points to aggregate information about the overall observed attacks. While the first consists of detecting attacks relying on a variety of existing IDS (*e.g.*, Snort, Bro), the second is based on a gossip-based communication mechanism to share information among the defense nodes, and increase the detection accuracy. The authors emphasize the use of collaboration (*i.e.*, communication) increases the precision of the detection and adjusts the rate limit mechanism of each defense node. However, [84] *et al.* disregards the existence of malicious nodes on the network, as described in the SOS [103] for instance, which can pollute the communication channel with malicious messages.

Technical: Hardware-based overlay focused on cooperative DDoS detection.
Social: Members are considered to be trusted.
Economical: Not addressed.
Legal: Partially addressed once members are trusted.
Year: 2006.

Speak-up [248, 249]

Speak-up [248, 249] is a defense mechanism that, unlike the previous, is designed to defend application-layer attacks. Speak-up is based on the principle of offense-defense by asking its clients to increase the volume of traffic sent, supposing that attackers are already using their maximum available bandwidth, and thus, they will not be able to react to the request. Hence, legitimate users, not using all bandwidth, can drastically increase the data volume. The goal is that the legitimate users crowd out the malicious ones. This technique is only usable against session flooding attacks and not in case of request flooding or an asymmetric attack. However, as a significant drawback of Speak-up, the technique is not applied to volumetric attacks at the network and transport layers, which do not know the application layer.

Technical: Software-based approach focused on application-layer DDoS attacks.
Social: Members are considered to be trusted.
Economical: Not addressed.
Legal: Partially addressed once members are trusted.
Year: 2006.

Chen, Yu and Hwang, Kai and Ku, Wei-Shinn [39]

[39] proposed a distributed DDoS detection mechanism targeting core networks named Distributed Change-point Detection (DCD) based on a Change Aggregation Tree (CAT) mechanism. A CAT operates at the router level for detecting abrupt changes in traffic flows, and when an attack is launched, the routers can observe changes in the spatial-temporal distribution of traffic volumes [39]. CAT's within the DCD are organized according to a hierarchy, in which edge routers are at the lower hierarchy level, and core routers are at a higher level.

Technical: Hardware-based approach requiring modified routers.
Social: Members are considered to be trusted.
Economical: Not addressed.
Legal: Partially addressed once members are trusted.
Year: 2007.

Defense and Offense Wall (DOW) [261]

The DOW (Defense and Offense Wall) is a mechanism using the Speak-Up technique combined with an anomaly detection method based on K-means clustering [261]. The anomaly detection algorithm is intended to detect session flooding, request flooding, and asymmetric attacks. Further, the authors propose an encouragement model to incentivize clients whose sessions are dropped and refused by the K-means anomaly detection model, to send more session connection requests. Similar to Speak-Up, legitimate users increase their rate so that they become more likely to be served, resulting in dropping suspicious and rising legitimate sessions.

Technical: Software-based approach build upon the Speak-Up approach.
Social: Members are considered to be trusted.
Economical: Propose an encouragement model to incentivize participation based on technical elements.
Legal: Partially addressed once members are trusted.
Year: 2007.

StopIt [129]

To Filter or To Authorize (StopIt) [129] allows third-parties to install filters to block undesirable traffic. StopIt has a closed-control and open-service architecture, which means that only authorized members can request filters' deployment and, in turn, each member can be requested to deploy filters.

It uses an authentication system for sources that prevent IP address spoofing. StopIt uses hierarchical queuing (first based on the source AS and then based on the source IP address) at congested links to separate legitimate traffic from attack traffic. Further, StopIt assumes that it exists an efficient detection system relying on authenticated nodes to install filters, further being able to recognize when authenticated nodes attempt a filter exhaustion attack.

Technical: Hardware-based approach including filtering capabilities.
Social: Members are authenticated but provide detection of misbehaving peers.
Economical: Not addressed.
Legal: Partially addressed once it is possible to select members to interact.
Year: 2008.

Trust Management Helmet (TMH) [262]

Trust Management Helmet (THM) uses trust to differentiate between legitimate users and attackers [262]. The idea is that legitimate users' connection should be prioritized and thus kept online before trying to identify all attack requests. This works with cryptographically secured licenses for legitimate users. Then, TMH differentiates trust aspects to distinguish between legitimate users and attackers' trust: short-term, long-term, negative, and misusing.misusen the license and trust values, the client's overall trust is calculated and TMH decides whether to accept his request. The collaborative aspect of TMH lies in the collaborative trust management among multiple servers where TMH is deployed, in order information about clients' trust scores.

Technical: Software-based approach to manage trust within an overlay network.
Social: Distinguish four levels of trust for its members (*i.e.*, reputation ranking).
Economical: Not addressed.
Legal: Possible to select circles of trust within the different levels.
Year: 2009.

Velauthapillai, Thaneswaran and Harwood, Aaron and Karunasekera, Shanika [247]

[247] presented a gossip-based protocol to exchange information concerning DDoS attacks using a cooperative overlay network. The authors propose a distributed detection algorithm in which each participating node locally measures the traffic bit rate routed to a specific victim. These nodes average their local estimates using the cooperative overlay to check whether the estimate exceeds a fixed inbound (traffic) threshold at the victim. However, the authors' approach observes only if the traffic capacity is exceeded or not, but it does not differentiate between legitimate and malicious traffic.

Technical: Software-based overlay to exchange DDoS detection information.
Social: Members are considered to be trusted.
Economical: Not addressed.
Legal: Partially addressed.
Year: 2010.

NetFence [130]

NetFence deploys a congestion policing feedback mechanism to guarantee legitimate users a fair share of network resources [130]. As the name fence suggests, the idea is to establish a trusted network perimeter enabling robust congestion policing within the network. By using a similar approach than the IP record route (*i.e.*, to traceback packets), trusted routers at the boundaries between the network and end systems, record stamp congestion policing feedback within each packet. Based on simulations, NetFence provably guarantees a legitimate sender its fair share of network resources without keeping per-host state at the congested link [130].

Technical: Hardware-based approach deployed on border routers.
Social: Members are considered to be trusted.
Economical: Not addressed.
Legal: Partially addressed.
Year: 2010.

FireCol [72]

FireCol [72] focuses on the early discovery of DDoS attacks proposing an overlay network composed by Intrusion Detection or Prevention Systems (IDS or IPS). The overlay of IDS/IPS follows a publish/subscribe method in each customer collaborate by computing and exchanging belief scores on potential attacks. Since attacks are highly distributed, the approach's idea is to explore the distribution of detection points to provide greater accuracy in detecting attacks. Although belief scores are computed for advertised attacks, there is no reputation assessment among customers who may act maliciously and result in false mitigation actions with possible economic impacts on third parties.

Technical: Hardware-based approach based on IDS/IPS systems.
Social: Members are considered to be trusted.
Economical: Not addressed.
Legal: Partially addressed.
Year: 2012.

Bohatei [69]

The emerging paradigm of NFV is often used in conjunction with Software-Defined Networking (SDN). Through SDN, the data plane is decoupled from the networking infrastructure's control plane, allowing tailored solutions for specific networking needs [69]. The routing required to relay the attack traffic in the case of CoFence could be simplified through SDN-based networking as similar mitigation solutions such as Bohatei, presented by Fayaz *et al.* [69] demonstrate.

Bohatei serves as a clear indicator of the scalability advantages in using SDN- and NFV-based networking to tackle the DDoS defense problem. The Bohatei proof of concept implementation discussed in [69] is realized with the OpenDaylight SDN controller [172] together with an assortment of Open Source tools to facilitate routing and mitigation such as OpenvSwitch [186], Snort [33], Bro

[181] and iptables[194]. Experiments conducted by Fayaz *et al.* show that the Bohatei solution could mitigate attacks with a total throughput of up to 100 Gbps while only requiring hardware, which was 2.1 to 5.4 times more cost-effective compared to ”fixed defense facilities” [69].

The proof-of-concept implementation of Bohatei as presented by Fayaz *et al.* does not directly incorporate inter-domain DDoS defense, however employing a scheme such as Pushback [95], which allows edge and access routers to relay traffic filtering to routers further upstream, could expand the approach and allow multi-domain cooperative defense with the flexibility of SDN and NFV.

Technical: Software-based approach based on SDN and VNF.
Social: Members are considered to be trusted.
Economical: Not addressed.
Legal: Partially addressed.
Year: 2015.

Sahay, Rishikesh and Blanc, Gregory and Zhang, Zonghua and Debar, Hervé [209]

The SDN paradigm offers benefits for network management through the global view of the network concentrated in a single management point. Based on these benefits, [209] proposes a collaborative framework that allows the customers to request DDoS mitigation from ASes. The proposal is based on an SDN controller implemented at the customer side interfaced with the AS, which can change the anomalous traffic label and redirect them to security middle-boxes. Upon request, ISPs can change the label of the anomalous traffic and redirect them to security middleboxes to provide on-demand DDoS mitigation services in an overlay network of trusted peers.

Technical: Software-based approach based on SDN.
Social: Members are considered to be trusted.
Economical: Not addressed.
Legal: Partially addressed.
Year: 2015.

Chin, Tommy and Mountrouidou, Xenia and Li, Xiangyang and Xiong, Kaiqi [40]

[40] proposes an attack detection and mitigation specific to SDN-enabled domains. A factor that distinguishes the proposed approach is that it operates with both real and virtual or software-based switches (Open Virtual Switches - OVS [186]) that can be deployed for specific virtual functions. The main component within the SDN controller is the monitor and correlator. While the monitor is responsible for collecting traffic statistics from installed flows, the correlator responds to monitors’ alerts on demand. The authors further describe a feedback loop where multiple correlator’s (deployed in different domains) communicate with each other to access related OVSs to reveal these attackers and generate insights into the attack traffic path. Once an attack is confirmed, mitigation actions can be taken to block attack traffic or reroute attack packets.

Technical: Software-based approach based on SDN and virtual OVS switches (suitable for cloud environment).
Social: Members are considered to be trusted.
Economical: Not addressed.
Legal: Partially addressed.
Year: 2015.

Giotis, Kostas and Apostolaki, Maria and Maglaris, Vasilis [78]

[78] presents a similar approach than [209] but including a reputation mechanism, which covers the possibility of participation of malicious peers in the network. SDN controllers in their proposal leverage on the SDNi (SDN interface) protocol (to provide lateral communication between SDN controllers), as the enabler for information exchange between adjacent SDN domains. Then, the victim domain can propagate links to incident reports, upstream, over BGP signaling, and mitigators (*i.e.*, SDN domains helping the victim) can decide whether to devote resources based on the reputation of the victim.

Technical: Software-based approach based on SDN.
Social: There is a reputation system to evaluate the contribution of peers.
Economical: Not addressed.
Legal: Partially addressed.
Year: 2016.

J. Steinberger and B. Kuhnert and A. Sperotto and H. Baier and A. Pras [232]

[232] uses a similar architecture than IETF DOTS [156] to simplify the exchange of threat information among trusted partners is used to reduce the time needed to detect and respond to large-scale network-based attacks. The authors propose an advertising protocol based on the FLEX (FLow-based Event eXchange) format, which simplifies the integration and deployment of the solution and facilitates the communication process between the involved domains. The FLEX standard uses both signature and encryption methods to prevent unauthorized access to the security event message at the application layer and ensure the confidentiality of the exchanged information. The main advantage of [232] communication process is that it easily integrates with the existing infrastructure and is easy to deploy, *i.e.*, it decreases deployment and operation complexities.

Technical: Software-based communication protocol based on FLEX.
Social: Not addressed.
Economical: Not addressed.
Legal: Not addressed.
Year: 2016.

Collaborative Intrusion Prevention Architecture (CIPA) [38]

CIPA [38] proposes an overlay network for the collaborative intrusion prevention architecture of DDoS attacks. CIPA builds an Artificial Neural Network (ANN) composed by collaborative SDN controllers, where each controller implements a neuron (monitoring the network traffic passing through them) and extract features that can characterize attacks. Therefore, the whole ANN works like an integrated IDS/IPS, allowing CIPA to detect attacks on a global view. However, it is noted that there is no mention of the possibility of malicious controllers, which can manipulate information to disrupt the whole network, as well as provide incentives for the exchange of information and mitigation actions, in case attacks are confirmed.

Technical: Software-based overlay network based on SDN.
Social: Members are considered to be trusted.
Economical: Not addressed.
Legal: Partially addressed.
Year: 2016.

CoFence [195]

The inter-domain mitigation requires an efficient scheme to provision the filters required to defend against a large-scale DDoS attack cooperatively. CoFence presented by [195] addresses this problem by relaying attack traffic to other participants in the cooperative alliance. This circumvents the problem of filter provisioning and provides a simple check for the mitigation's efficacy since the participating systems will only relay back attack-free traffic, which directly proves that they have conducted the mitigation [195].

Deployment of a CoFence instance relies on Network Function Virtualization (NFV) [164] to simplify the entire system's instantiation. Utilizing NFV for a collaborative defense system can help persuade potential new members of a DDoS alliance to join since device upgrading and creation are relatively fast and low cost due to the virtualized nature of all networking components [195]. Instead of relying on fixed hardware-based networking solutions, commodity hardware can launch virtualized networking appliances that can be spun up on-demand [69].

Technical: Software-based overlay network based on NFV.
Social: Members are considered to be trusted.
Economical: Not addressed.
Legal: Partially addressed.
Year: 2016.

DDoS Open Threat Signaling (DOTS) [156, 167]

The Internet Engineering Task Force (IETF) Distributed-Denial-of-Service Open Threat Signaling (DOTS) architecture [156] was devised as a standardization attempt for collaborative DDoS defense. Through the DOTS protocol, data models are provided to enable intra- and inter-organizational DDoS defense with multiple parties [167]. DOTS focuses explicitly on aiding with the coordination of attack responses with a client and server model. The DOTS client requests mitigation from the DOTS server after detecting an ongoing attack [156].

Communication between the DOTS server and client takes place over a data- as well as a signal-channel. The client uses the signal channel to request mitigation from the server, and the server uses the signal channel to inform the client about the status of the mitigation [156]. As part of the client's information to signal the server for help, attack targets, as well as telemetry data about the attack, can be provided through the signal channel to simplify the mitigation for the server [156]. The data channel, which is an optional component in the DOTS scheme, is used to exchange additional configuration information that can then be used and the information transferred through the signaling channel. These configurations may consist of host identifiers, blacklists, whitelists, traffic filters, or DOTS client provisioning [156].

Technical: Hardware-based system and protocol supported by IETF.
Social: Members are considered to be trusted.
Economical: Not addressed.
Legal: Partially addressed.
Year: 2017.

Hameed, Sufian and Ahmed Khan, Hassan [85]

[85] proposes a controller-to-controller protocol (*i.e.*, a east/west communication similar to [38, 78]) that allows SDN-controllers to securely communicate and transfer attack information. However, the authors seek to present a protocol aiming at three aspects: lightweight, efficient, and easy to deploy. The overlay network is composed of certified domains organized as a source, intermediate, and destination domains depending on the attack.

Depending on the origin of the reported attack, the domains involved behaving differently to provide an accurate detection (and mitigation) of the attack. Although participating members are trusted (*i.e.*, certified), there are no incentives for them to use their local resources to participate in the collaborative detection and mitigation process. This creates the possibility of free-riders who use other people's resources without contributing.

Technical: Software-based overlay network based on SDN.
Social: Members are considered to be trusted (certified domains).
Economical: Not addressed.
Legal: Partially addressed.
Year: 2018.

OverWatch [86]

OverWatch [86] presents a cross-plane DDoS attack detection and mitigation framework in SDN. The detection consists of a coarse-grained flow monitoring algorithm and a fine-grained machine learning-based attack classification algorithm. The collaborative tracking aims to locate the switches close to the botnets by looking up the source IP address of a given flow. However, the approach does not address the possibility of spoofed IP addresses, which is a common problem in IP traceback approaches (*e.g.*, IP Traceback [263] and AITF [10]) to mitigate the attack near the source. Also, there are problems related to malicious and selfish SDN controllers, which may disclose false information or use other people's resources without contributing.

Technical: Software-based overlay network based on SDN.
Social: Members are considered to be trusted.
Economical: Not addressed.
Legal: Partially addressed.
Year: 2018.

CoChain [1]

CoChain [1] is a proposal similar (and based on) to this thesis that uses BC as a means of signaling attacks between internet domains. Despite based on the same set of technologies for the development of the proposal, CoChain's focus is on the aspect of mitigating inter and inter-domain attacks, and not in the aspect of signaling that expands the applicability of the solution and does not impose restrictions as to the type of software/hardware to be used in the underlying layers.

Concerning inter-domain communication, the authors proposed a SC-based framework that uses Ethereum's technology to facilitate collaboration among SDN-based domain peers, which make use of a permissionless setting. In a permissionless setting, all information exchanged between peers and contracts is open to the public, resulting in a issue of confidentiality for peers, impacting social, security and legal aspects, about which and how data is shared.

Technical: Software-based approach based on BC and SDN.
Social: Not addressed (*i.e.*, allows permissionless BCs).
Economical: Not addressed.
Legal: Not addressed.
Year: 2019.

Spathoulas, Georgios and Giachoudis, Nikolaos and Damiris, Georgios-Paraskevas and Theodoridis, Georgios [229]

[229] propose the deployment of lightweight agents on Internet-of-Things (IoT) devices to detect DDoS attacks collaboratively. The authors propose a SC in which agents (decentralized apps) deployed on IoT devices can be communicated in a traceable and immutable way. In this regard, the authors declare that their scheme can be implemented in any BC platform that offers SC functionality. However, it is noted that there is no mention concerning the impacts on the minimum hardware capacity required in the IoT devices, nor regarding the confidentiality of the information exchanged between the devices (*i.e.*, the use of permissionless BCs makes all information public). Even if a software-based approach imposes less overhead for deployment and operation of collaborative defense, the consensus mechanism employed by public BCs (*i.e.*, PoW) requires high computational power from the participating nodes, which may render the proposal unfeasible.

Technical: Software-based approach based on BC for signaling attacks.
Social: Not addressed (allows permissionless BCs).
Economical: Not addressed.
Legal: Not addressed.
Year: 2019.

Essaid, Meryam and Kim, DaeYong and Maeng, Soo Hoon and Park, Sejin and Ju, Hong Taek [66]

[66] presents a BC-based collaborative DDoS mitigation solution inspired in this thesis (as proposed in [200]) further combining a deep learning DDoS detection system for the signaling of IP addresses in the BC. Each participant needs to deploy a SC to join the collaborative DDoS mitigation. In its turn, SCs define how participants exchange information and parameters defining the cooperative defense (*e.g.*, which networks are managed). The proposal has a significant focus on implementing the decentralized agent (*i.e.*, dApp) for the detection of attacks and not on the collaborative aspect. In this sense, the work does not address aspects related to tracking members' reputations or aspects related to the provision of financial incentives to cover expenses related to collaborative detection and mitigation tasks.

Technical: Software-based approach based on BC for signaling attacks.
Social: Not addressed.
Economical: Not addressed.
Legal: Not addressed.
Year: 2019.

Pavlidis, Adam and Dimolianis, Marinos and Giotis, Kostas and Anagnostou, Loukas and Kostopoulos, Nikolaos and Tsigkritis, Theocharis and Kotinas, Ilias and Kalogeras, Dimitrios and Maglaris, Vasilis [180]

[180] extended the proposal of a federated BC-based approach proposed by [77], presenting an approach for signaling, coordination, and orchestration of collaborative mitigation. The authors further consider the work proposed in this thesis [200] and their previous work on SDN involving the tracking of the reputation of collaborative instances [78] to implement their BC-based approach. As an additional aspect, the authors consider resource allocation within the collaborative instances to mitigate attacks, modeling a combinatorial optimization problem considering reputation scores and network flow weight of each member. In this sense, their approach presents similar characteristics of this thesis but does not include the aspect of incentives for the use of third-party infrastructure, *i.e.*, there is the possibility of free-riding in the collaborative environment.

Technical: Software-based approach based on BC and SDN.
Social: Provide tracking of reputation.
Economical: Not addressed.
Legal: Partially addressed once is possible to select peers to interact.
Year: 2020.

B
BloSS Smart Contracts

This appendix presents details of implementation aspects of the contracts used in BloSS. The system is based on two main contracts: the Register and the Protocol. Whereas the Register acts as an anchor in which peers can find addresses of other peers so that they can interact, the Protocol is the main contract in which the relationship between target and mitigator is performed.

B.1 Register Contract

The Listing B.1 shows the Register mapping (named mitigators) and the struct pair of addresses consisting of the Protocol, mitigator as well as a boolean variable. First the mitigator registers with the *setMitigator* function by passing a string name and the address of the mitigator itself to the Register. At this point the Protocol field is defined as the default address value of zero.

Mapping a string to the struct allows for usable name searching for mitigators by the target of the DDoS attack. Optimally the target already has the address of the mitigator and only needs to initialize the Protocol for the mitigation signaling to start. But in case the target requests a specific mitigator, which requires the name of that mitigator to actually exist in the Register, the *getMitigator* function is called. Then the address of the mitigator is returned to the Protocol, being set as the payable mitigator. This also allows the mitigator to get the Protocol address and therefore enables interaction with the Protocol Smart Contract for both parties.

```
pragma solidity ^0.5.8;
import "./Enums.sol";

contract Register{

    // set new mitigator by a possible mitigator address
    function setMitigator(string memory _name, address _Mitigator)
        public {
        if(mitigators[_name].isAdded == false){
            mitigators[_name].Protocol = address(0);
            mitigators[_name].Mitigator = _Mitigator;
            mitigators[_name].isAdded = true;
        }else{
            emit LogNotValid('Mitigator already registered.');
        }
    }

    // gets called by the target via the Protocol and sets Protocol
        address to the mapping of the mitigator
    function getMitigator(address _Protocol, string memory _name)public
        returns(address){
        require(mitigators[_name].Mitigator != msg.sender, "[
            getMitigator] Protocol is not allowed to be a Mitigator.");
        if(mitigators[_name].isAdded){
            mitigators[_name].Protocol = _Protocol;
            return mitigators[_name].Mitigator;
        }else{
            emit LogNotValid('Mitigator not registered.');
            return address(0);
        }
    }

    // gets called by mitigator, returns address of the Protocol
    function getProtocol(string memory _name) public view returns(
        address){
        require(mitigators[_name].Mitigator == msg.sender,'[getProtocol]
             You have no permission to see this information.');
        if(mitigators[_name].Protocol != address(0)){
            return mitigators[_name].Protocol;
        }else{
            return address(0);
        }
    }

    function getCreator() public view returns(address){
        return creatorRegister;
    }
}
```

Listing B.1: Register Smart Contract

B.2 Cooperative Signaling Protocol

Listing B.2 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 B.3).

```
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 B.2: First Interactions of T and M as Mapped in BloSS: Initiate SC, Its Acceptance, and an Upload of Proof by M

Rating actions in Listing B.3 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()*.

```
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 B.3: Rating by T and M in the BloSS SC

C
Cooperative Signaling Global Latency Evaluations

Listed below *performance* results in terms of latency to complete the signaling protocol, and their differences between the standard deviations and median times, are depicted in the Figures Scenario 1 to Scenario 11. Tests are based on the used blockchains ganache, local Rinkeby or global Rinkeby. Tests run on Ganache were set to 15 seconds block time and both Rinkeby tests relied on the average 15 second blocktime. Also, the range of the outliers in close to every scenario is mostly found in the global Rinkeby tests, for instance in scenarios 3, 5, 6 and 11.

Table C.1: Control Condition Rinkeby Processing Times [s] for Protocol (Tokyo-São Paulo)

Scenario	Average Processing 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

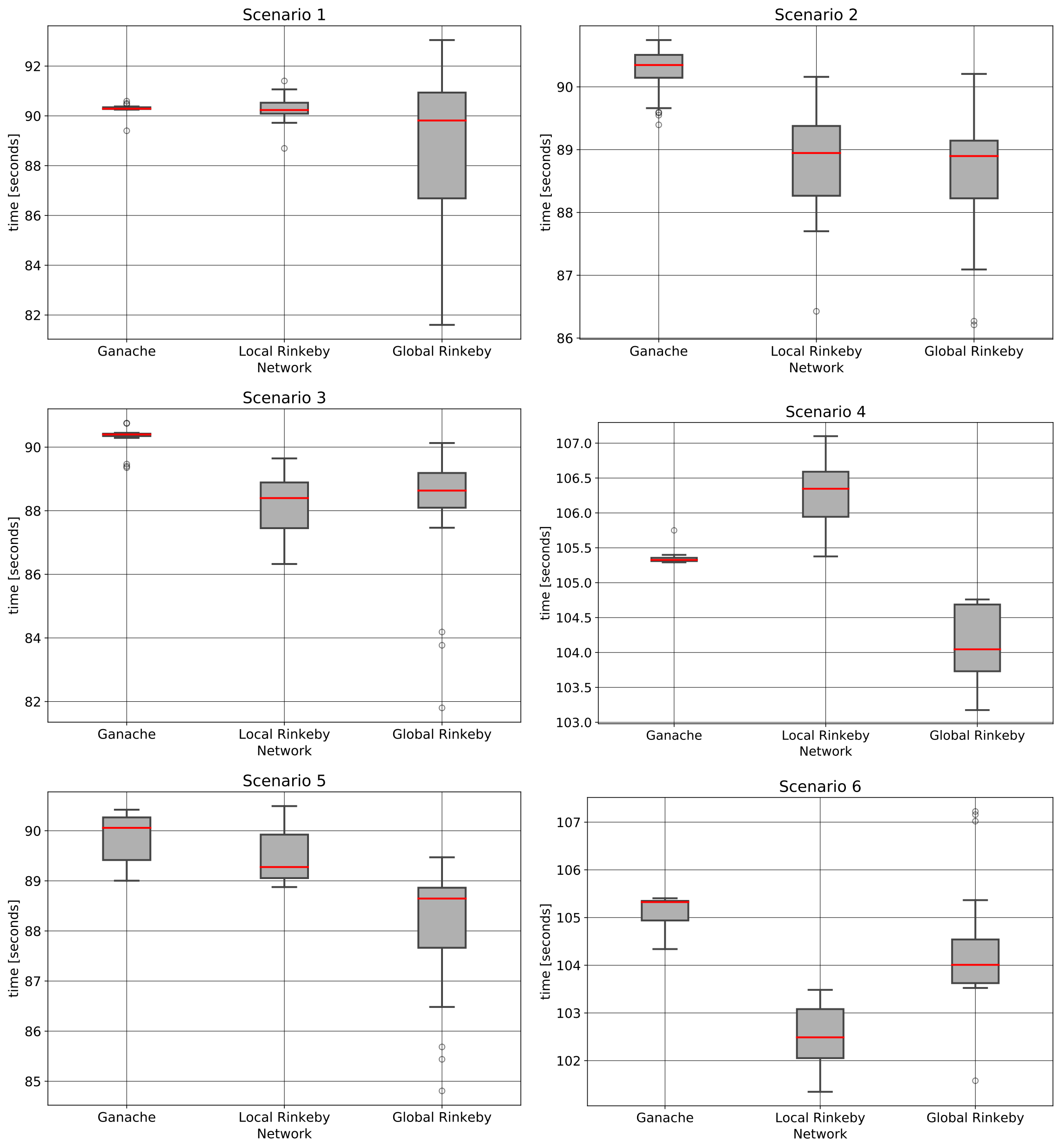


Figure C.1: Latency [s] to Execute the Protocol' Smart Contract. Outcome for Scenarios 1 to 6 [243]

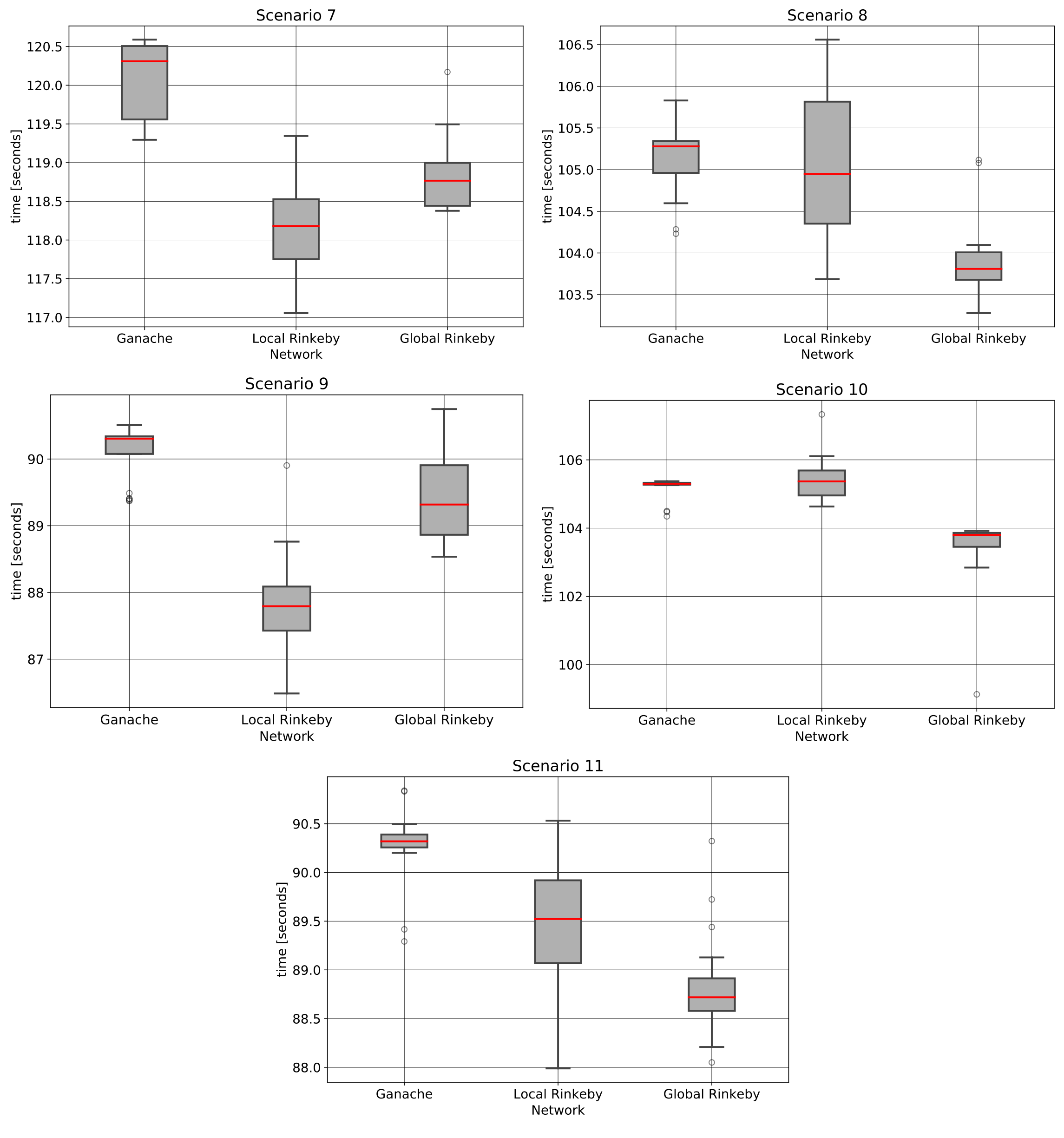


Figure C.2: Latency [s] to Execute the Protocol' Smart Contract. Outcome for Scenarios 7 to 11 [243]

D
Reputation in Acceptance Mode

The tests with the simulator in acceptance mode were performed using different compositions of customer strategies. The three experiments only differ in measurement duration (from one up to 10 days) and the amount of customers. Figures on the remaining pages present the measured average Beta reputation by time and customer strategy for all three test runs in acceptance mode.

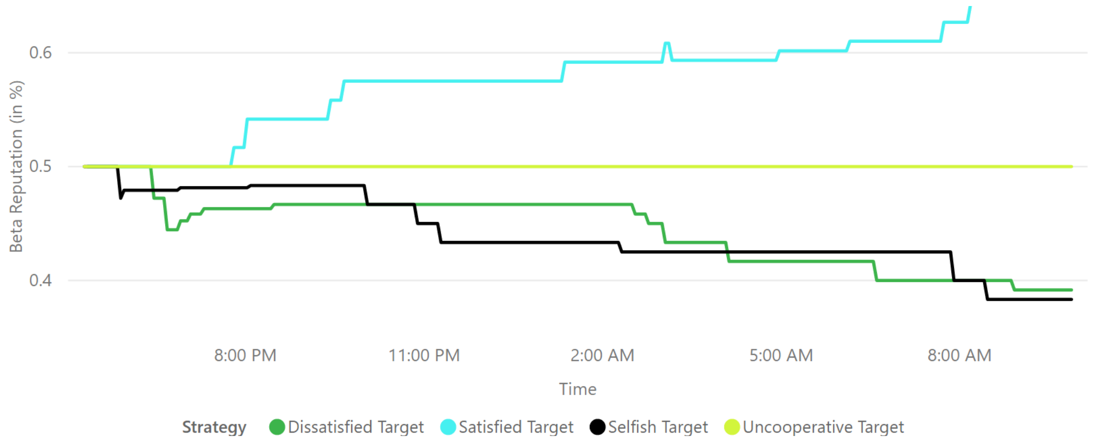


Figure D.1: Average Target Beta Reputation for First Test in Acceptance Mode, with 10 Customer Strategies Each [82]

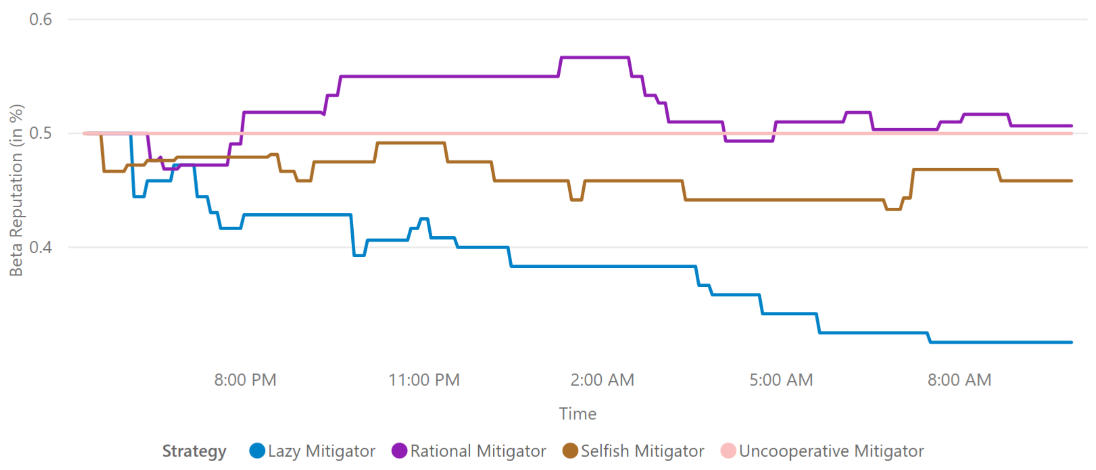


Figure D.2: Average Mitigator Beta Reputation for First Test in Acceptance Mode, with 10 Customer Strategies Each [82]

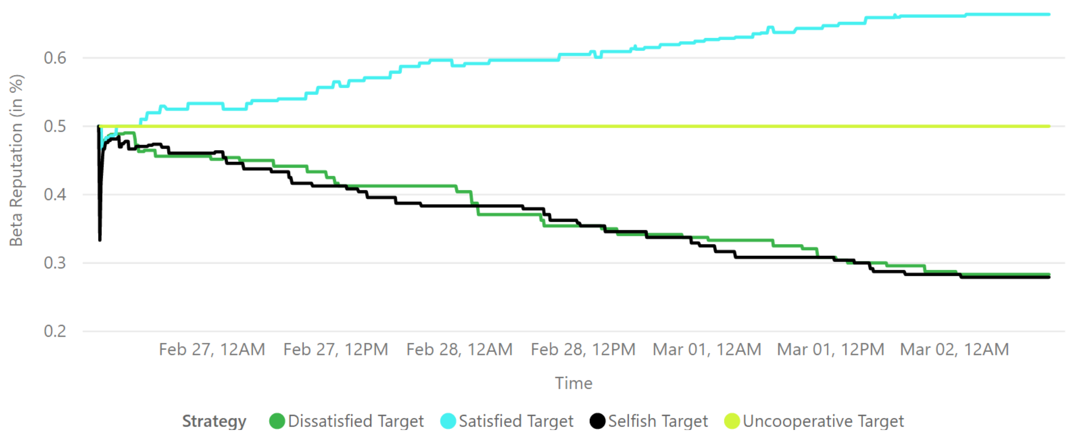


Figure D.3: Average Mitigator Beta Reputation for Second Test in Acceptance Mode, with 20 Customer Strategies Each [82]

Overall, Beta reputation scores for attack targets (T's) allow to identify a satisfied T, because in comparison with the undesired T strategies it develops the highest average reputation value (see Figure D.1). From Figure D.3 it becomes evident, that dissatisfied and selfish T's are constantly downgraded and will eventually no longer receive help from mitigators (M's), due to their bad rating. It is not desirable in all circumstances, that satisfied T's crowd out dissatisfied T's, since this will incentivize T's to accept also poorly (or even badly) delivered mitigation services. Because in the current design, no third party checks that a rating indeed matches the quality of the delivered service, this problem currently remains unsolved.

After the first mitigation contracts, the M reputation values in Figure D.2 also allow to clearly distinguish a constant rational M from the lazy M. However, unlike for the T's, it is difficult to distinguish the rational M from the selfish and uncooperative M (see Figure D.4). The uncooperative M usually shows very low positive and negative reputation, because it aborts mitigation tasks early. Hence,

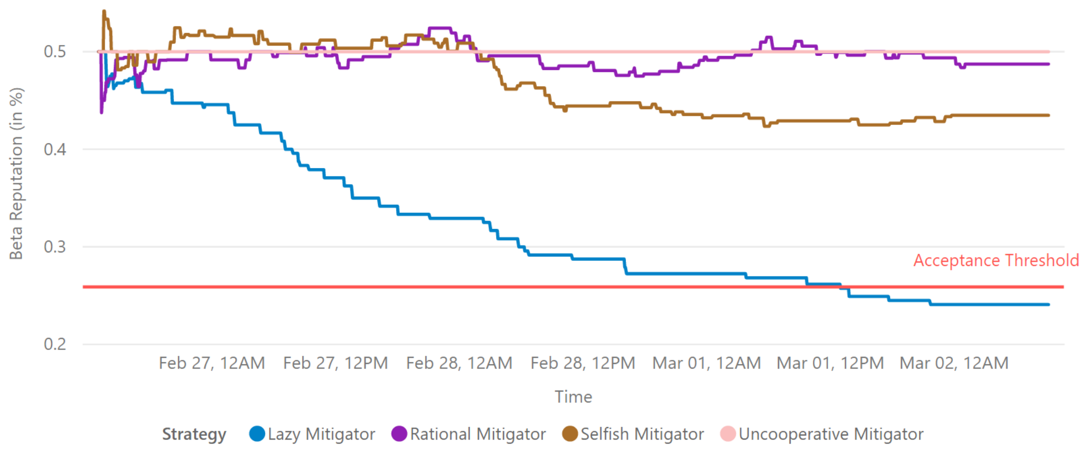


Figure D.4: Average Mitigator Beta Reputation for Second Test in Acceptance Mode, with 20 Customer Strategies Each [82]

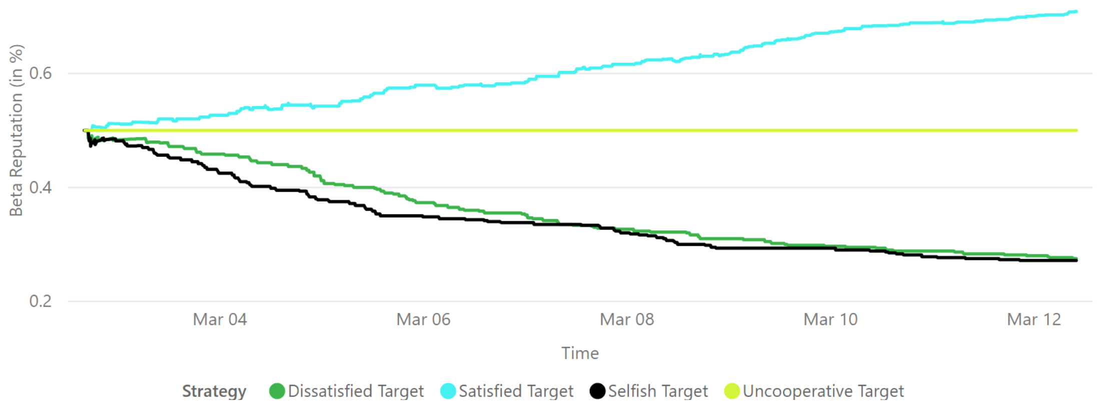


Figure D.5: Average Target Beta Reputation for Third Test in Acceptance Mode, with 50 Customer Strategies Each [82]

chances of picking an uncooperative M for a transaction is diminished the most by considering raw reputation values (especially the amount of positive and negative ratings). Because both, the selfish and rational M upload proofs, the only difference between the two strategies is, that the selfish M never rates. Selfish M behavior is an irrational strategy, since the selfish M (deliberately or not) deprives itself of reward payments. If we assume, that M rarely forgets to rate services in the end (which is the definition of selfish behavior), we can also assume that there exist more rational than selfish M's. This leads us to the conclusion, that chances picking a rational M compared to a selfish M with the same Beta reputation score are higher, due to the reward incentive.

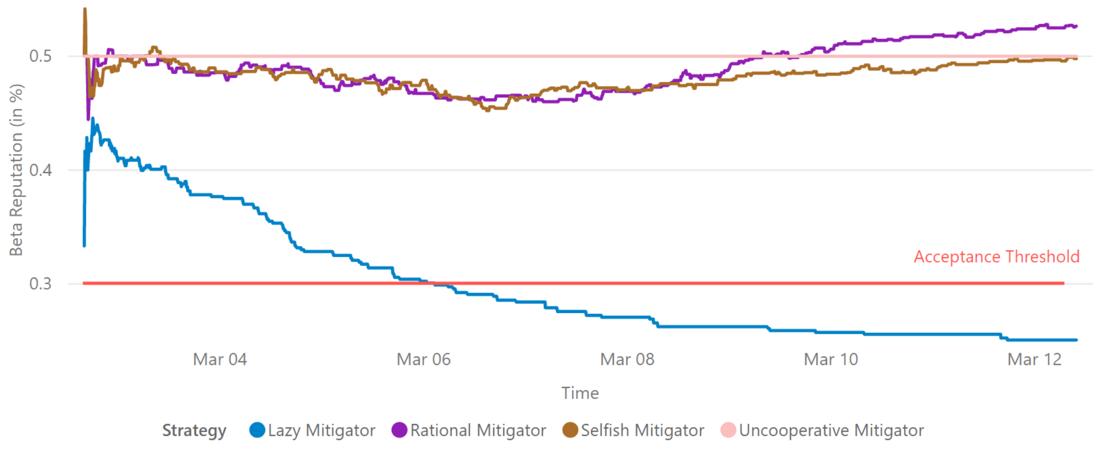


Figure D.6: Average Mitigator Beta Reputation for Third Test in Acceptance Mode, with 50 Customer Strategies Each. [82]

E
Description of Smart Contract's Vulnerabilities

This Appendix lists the specific vulnerabilities found in the Smart Contract (SC) vulnerability analysis. The list includes a brief description of those vulnerabilities.

1. **Reentrancy**. Reentrancy occurs if one contract hands over the control to another contract, the second contract can call back into the first one several times before the first initiated interaction is completed [62]. There are two types of reentrancy:

   - Single function reentrancy [211]:
     (a) A *call, send* or *transfer* function which can hand over a control to an external contract is executed.
     (b) The external contract has a fallback function.
     (c) After the *call, send* or *transfer* function was executed, the state is updated.
   - Cross-function reentrancy [211]: similar to single function reentrancy, happens when two different functions or contracts share the same state.

2. **Transaction ordering**. The state of a contract in which a transaction is executed depends on the transaction order determined by the block's miners and cannot be predicted [190] reliably.

3. **Block timestamp dependency**. Users can generate block timestamps which differ up to 900 seconds from other users' block timestamps [190]. Often functions rely on the starting time (*StartTime*), current time (*now*) and ending time (*EndTime*), which depend on *block.timestamp* [190].

4. **Blockhash usage**. Similar to block timestamp dependency. Malicious users can manipulate the outcome by changing the blockhash. Example: Using *blockhash* for generating randomness [62].

5. **Exception handling**. No proper exception handling. It is impossible to check the return value after a function call [190].

6. **Call stack depth limitation**. The call stack depth limit is hard-coded to 1024 frames. Every time a *call* or *send* function calls another contract, the call stack depth is increased by one [35].

7. **Integer overflow and underflow**. The result of an arithmetic operation is outside of the range of a Solidity data type [35].

8. **Unchecked and failed *send***. The *send* function may fail if the gas limit is exceeded or if there are not enough Ethers on balance. However, the function does not have a built-in error handling [190].

9. **Destroyable and suicidal contracts**. A smart contract that can be terminated by an anonymous *suicide* or *kill* function. This function is usually executed by the owner in case of an attack or an emergency [190].

10. **Unsecured balance**. The balance of a smart contract is unsecured. Possible reasons include improper access control for *balance* and *constructor* and updating the balance after sending the money [190].

11. **Use of *tx.origin***. *tx.origin* returns the account address initiating the transaction [190].

12. **Unrestricted write**. A write operation to the storage which does not have any restricting conditions [190].

13. **Unrestricted transfer**. By default, the *call* function allows to transfer of money between any users and smart contracts [190].

14. **Non-validated arguments**. The arguments which are not checked before passing to a method [190].

15. **Greedy contracts**. If an external library contract is terminated, the contracts call this library to become greedy as they cannot access the library and transfer the funds anymore [190]. Besides, some contracts can receive Ethers but lack the instructions for sending them out (*e.g.*, *, send, transfer, call*), or the instructions are unreachable [166].

16. **Prodigal contracts**. The sending function can be called by any user and can be used to send funds to any address chosen by the sender [190].

17. **Overspent gas**. Many patterns in Solidity smart contracts are costly in terms of gas required to spend for their execution [190]. Gas-costly programming patterns include dead code, opaque predicate, expensive operations in a loop, the constant outcome of a loop, loop fusion, repeated computations in a loop, and comparison with unilateral outcomes in a loop [36].

18. **Gasless send**. A transaction fails because not enough gas is provided, such as an expensive function that requires much gas to execute [62].

19. **External calls**. A call of an external contract. Pushing data to an external contract is, in general, more dangerous than pulling data [62].

20. **DoS with unexpected revert**. The vulnerability occurs when a conditional statement *if*, *for* or *while* depends on an external call [62].

21. **DoS with unbounded operations**. Due to the improper programming of unbounded operations (*e.g.*, a loop over a large array), the amount of gas required for contract execution may exceed the gas limit of the block [35].

22. **DoS with block stuffing**. Because of the greedy mining incentive mechanism, an attacker can offer a high *gasPrice* and, thus, motivate miners to prioritize their transactions over others [35].

23. ***Delegate call* injection**. Ethereum allows embedding a callee's bytecode into the caller contract using the *delegatecall* function. As a result, the caller contract's state variables can be modified by the bytecode of the callee contract [35].

24. **Ether lost to orphan address**. When money is transferred, Ethereum only checks if the recipient's address is no longer than 160 bit but does not check if the address exists. If money is sent to a non-existing (orphan) address, Ethereum registers the address automatically. However, the address does not have any associated user or contract account. Thus, it is impossible to withdraw the transferred money [35].

25. **Manipulated balance**. The vulnerability occurs if the contract's control-flow relies on *this.balance* or *address(this).balance*, as they can be manipulated by an attacker [35].

26. **Outdated compiler version**. An outdated compiler may contain unfixed bugs [35].

27. **Upgradeable contracts**. In order to solve the problem of immutability in blockchain, developers can split smart contracts into two parts (a *proxy contract* which remains immutable when added to the blockchain and a *logic contract* which can be updated by the developer) or use a *registry contract* for recording the updates [35].

28. **Erroneous visibility**. Functions that should not be called from external contracts are sometimes erroneously marked as *public* or *external*. As a result, they can be called directly by attackers [35].

29. **Secrecy failure**. Due to transparency of the blockchain, marking functions and variables as *private* does not guarantee data secrecy [35].

30. **Insufficient signature information**. Instead of using multiple transactions, a user sends money to multiple recipients with a proxy contract. The user can send digitally signed messages off-chain to recipients letting them withdraw the money. The proxy contract validates the digital signature to check if the transaction is approved. If the digital signature does not contain due information (*e.g.,* nonce and proxy contact address), a recipient can use the signed message several times and withdraw additional funds [35].

31. **Type casts**. When a function in an external contract is called using an address argument, the Solidity compiler checks if the function is declared in the contract but does not check if the address argument conforms to its address. If there is another contract with the same declaration and a function named as in the first contract, the wrong contract's function may be executed by mistake [35].

32. **Short address**. In contract-invocation transactions such as *transfer*, the selector and arguments are automatically encoded. The first four bytes are reserved for the callee function, and the rest stands for arguments in chunks of 32 bytes. That means that if one argument's last byte is missed, two hexadecimal zeros are added to the end of the last argument [35].

33. **Under-priced opcodes**. Some contacts contain many opcodes which have a low gas price but consume a lot of computing resources [35]. For example, *balance*, *extcodecopy*, *extcodesize*, *sload* and *suicide* are considered under-priced [37].

34. **Generating randomness**. A seed such as *block.number*, *block.timestamp*, *block.difficulty* and *blockhash* used for generating randomness can be manipulated by miners [35].

35. **Outsourceable puzzle**. Ethereum's Proof-of-Work puzzle makes a solution only partially sequential. Therefore, it is possible to divide a Proof-of-Work task into several parts and outsource them [35].

36. **51% hashrate**. Due to the Proof-of-Work consensus mechanism, attackers can take over the blockchain if they have at least 51% of the mining power [35].

37. **Fixed consensus termination**. Ethereums's consensus protocol uses deterministic termination to achieve a probabilistic agreement. In other words, if a block is followed by a fixed number of blocks *n*, it will most likely remain on the blockchain. When all the block transactions are committed and the next *n* blocks are added to the blockchain, the consensus for the block is terminated. However, in reality, the probability of agreement can be affected by external factors such as a communication delay as, in an asynchronous network, a deterministic protocol cannot guarantee agreement, termination, and validity at the same time [35].

38. **Rewards for uncle blocks**. A stale block referenced by a regular block is called an *uncle* block. In Ethereum, not only regular blocks on the main chain but also uncle blocks are rewarded [35].

39. **Unlimited nodes creation**. The node generation is weakly restricted. Malicious users can create an unlimited number of nodes (even with the same IP address) and use them to monopolize the connections to the victim's nodes [35].

40. **Public peer selection**. When a node tries to locate a target, it queries 16 nodes in the bucket close and asks each of them for 16 closest to the target neighbors. This process iterates until the target is identified. During this process, the IDs of different nodes are provided to the querying node. The mapping of a node ID to the buckets in the routing table is public and can be exploited by attackers [35].

41. **Sole block synchronization**. A node may miss the synchronization with another one. If the second node is malicious, it can delay the synchronization on purpose. As it is only possible to synchronize with one node at a time in Ethereum, the first node becomes stalled and has to reject any subsequent blocks [35].

F
Publications

The thesis' proposal and development resulted in the publication of several scientific articles, which are directly or indirectly related to cooperative network defenses. Further, master's and bachelor's theses were developed in the Blockchain Signaling Systems (BloSS) context, whose individual outcomes in terms of evaluations and system improvements contributed toward the overall BloSS architecture and results.

F.1 Contribution of Own Publications Within Chapters

The thesis objectives outlined at the macro level are typically decomposed into several specific goals, which are achieved by publications across the thesis period. Thus, it relevant to highlight where in the thesis those publications appear as a core or additional elements in each Chapter. In the case of own publications as additional elements, references to the state of the art are duly cited when mentioned in the Chapters, mainly in the cases of Chapters 2 (Theoretical Foundations) and 3 (State-of-the-art of Cooperative Network Defenses). Table F.1 lists own contributions in the thesis' Chapters.

It is important to note that [206] is a journal publication, which presents an overview of BloSS in a summarized way (in contrast to this thesis) based on the contributions of several individual publications. Thus, [206] appears as a contribution in all Chapters of this thesis.

F.1.1 Chapter 1. Introduction

Refers to the introduction and motivation about the need and relevance of cooperative defenses, as well as an overview of the thesis. In this sense, [202] and [200] were the first publications that outlined the proposal to use Blockchains (BC) as a signaling platform in this context. While [202] was

Table F.1: List of Publications per Chapter

Chapter	Related Publications					Student Thesis	
	Full Paper	**Short Paper**	**Journal**	**Book Chapter**	**Demo**	**MSc**	**BSc**
1. Introduction	[200]	[202]	[206]				
2. Theoretical Foundations			[206]	[148, 203, 217]			[73]
3. State-of-the-art of Cooperative Network Defenses			[206]				
4. Design of the Cooperative Signaling Protocol	[81, 200]	[207]	[206]			[82]	[65, 243]
5. Design of the Blockchain Signaling System	[81, 141, 197]	[108, 109]	[206]			[82, 107, 142]	[65]
6. Evaluation	[81, 141, 197]		[206]		[199, 201]	[82, 142]	[26, 65]

Publication Details
[206] Blockchain Signaling System (BloSS): Cooperative Signaling of Distributed Denial-of-Service Attacks
[202] Multi-Domain DDoS Mitigation Based on Blockchains
[201] Enabling a Cooperative, Multi-domain DDoS Defense by a Blockchain Signaling System (BloSS)
[200] A Blockchain-based Architecture for Collaborative DDoS Mitigation with Smart Contracts
[203] The Use of Blockchains: Application-driven Analysis of Applicability
[148] Análise de Mecanismos para Consenso Distribuído Aplicados a Blockchain
[217] Blockchains and Distributed Ledgers Uncovered:Clarification, Achievements, and Open Issues
[81] A Reputation Scheme for a Blockchain-based Network Cooperative Defense
[141] Toward Mitigation-as-a-Service in Cooperative Network Defenses
[197] Evaluating a Blockchain-based Cooperative Defense
[108] Threat Management Dashboard for a Blockchain Collaborative Defense
[109] Security Management and Visualization in a Blockchain-based Collaborative Defense
[199] Cooperative Signaling of DDoS Attacks in a Blockchain-based Network
[207] SC-FLARE: Cooperative DDoS Signalingbased on Smart Contracts
[82] A Reputation and Reward Scheme for a Cooperative, Multi-domain DDoS Defense
[142] Mitigation as a Service in a Cooperative Network Defense
[26] Cooperative Signaling of DDoS Attacks in a Blockchain-based Network
[65] Performance Analysis of Blockchain Off-chain Data Storage Tools
[243] Cooperative Signaling Protocol
[107] Security Management and Visualization in a Blockchain-based Collaborative Defense
[73] Botnet Command-and-Control Traffic Analysis

a publication made in a workshop for PhD students in order to provide an overview of the topic and validate research questions, [200] was published at the same event presenting the initial architecture of the system based on an emulated prototype of BloSS. The emulated prototype presented in [200] was based on the use of Mininet [123] to emulate different domains (*i.e.*, Autonomous Systems) and attacking traffic, where BloSS was implemented in an initial version strongly coupled with the controller, based on an Ethereum PoW network.

F.1.2 Chapter 2. Theoretical Foundations

It is the Chapter that provides the conceptual background of the thesis and, in this sense, the publications made during the thesis period were more comprehensive in terms of how and when to use Blockchain, *e.g.*, the case of [203] where one of the use cases presented in the book Chapter was of a cooperative network defense. In the same direction, participation in two other book Chapters [148] and [217] contributed to this thesis in the description of consensus mechanisms and different types of blockchains deployment. Concerning another fundamental concept involved in the thesis (DDoS attacks), Getoar Gallopeni's bachelor thesis [73] conducted a study of Mirai Botnet in the BloSS cluster to evaluate in practice the effects of the bot, mode of propagation and countermeasures, thus, concepts reported in the literature could be verified in practice.

F.1.3 Chapter 3. State-of-the-art of Cooperative Network Defenses

Describes the state of the art in relation to collaborative network defenses. Altough being one of the most extensive Chapters in the thesis by providing a wide review of the literature, it is not thoughtfully based on own publications. An overview of major related work is provided in the Journal [206] but not in detail as Chapter 3 and extended description in Appendix A.

F.1.4 Chapter 4. Design of the Cooperative Signaling Protocol

Describes the final design of BloSS on-chain protocol based on the various refinements and interactions since its initial proposal in [200]. In this regard, the Chapter describe the refined protocol based several individual contributions. The investigation on the truthfullness of mitigation proofs performed in [143] revealed the need to integrate a reputation system as there is no automated way to verify the accuracy of a mitigation service performed by a third party. Hence, the Chapter also counts on the description of the reputation system proposed in [81, 82], the refined cooperative signaling protocol described in [207, 243], and the off-chain signaling approach [65, 197].

F.1.5 Chapter 5. Design of the Blockchain Signaling System

Describe the decentralized architecture of BloSS interfacing with the cooperative protocol and the network management system. In its first version published in [200], BloSS was tightly integrated with the network management system. The architecture refinement included modularization of its components providing well-defined interfaces that allow the decoupling of the underlying network management system was disclosed in [142, 143], which allowed to simplify the operation and development of BloSS, consequently its ease of deployment and operation.

F.1.6 Chapter 6. Evaluation

The evaluation Chapter presented several evaluations performed during the thesis period based mainly on the decoupling of architecture [142]. Then, it was possible to perform individual evaluations on the different components, in different settings (*e.g.*, simulated, local, and global). For instance, the Chapter disclosed evaluations performed at the local prototype and deployed in the cluster aiming at the assessment of BloSS' functionalities and correctness [142]. Also, usability assessments of the dashboard described in [107–109] improved the previous system (published in [200]) with a focus on improving usability; The performance evaluation of the off-chain signaling channel based on IPFS [65, 197]. The cooperative signaling protocol latency was evaluated locally and globally whereas the goal was to assess the correctness the signaling performance (in terms of latency) in all possible outcomes of the protocol, based on results published in [206, 207]. Further, an analysis on BloSS contracts conducted in [26] based on local experiments using different automated vulnerability assessment tools was included in the Chapter. Lastly, the resulting prototype was published as a demonstration in [199, 201].

F.2 List of Publications

This section presents publications made by the author only during the thesis period. In addition to the previously cited publications that contributed directly to the thesis, participation in the Communication Systems Group (CSG) research group resulted in the contribution of several other projects and publications listed in the period 2017 to 2020.

2020

- *Full paper*. M. Franco, **B. Rodrigues**, E. Scheid, A. Jacobs, C. Killer, L. Granville, B. Stiller: SecBot: a Business-Driven Conversational Agent for Cybersecurity Planning and Management; 16th International Conference on Network and Service Management (CNSM 2020), Izmir, Turkey, November 2020, pp 1–7.

- *Full paper*. Christian Killer, **Bruno Rodrigues**, Eder John Scheid, Muriel Franco, Moritz Eck, Nik Zaugg, Alex Scheitlin, Burkhard Stiller: Provotum: A Blockchain-Based and End-to-End Verifiable Remote Electronic Voting System; IEEE 45th Conference on Local Computer Networks (LCN), Sidney, Australia, November 2020, pp 1–12.

- *Full paper*. Clemens Jeger, **Bruno Rodrigues**, Eder Scheid, Burkhard Stiller. Analysis of Stablecoins during the Global COVID-19 Pandemic. The Second International Conference on Blockchain Computing and Applications (BCCA 2020).

- *Technical Report*. Christian Killer, Lucas Thorbecke, **Bruno Rodrigues**, Eder John Scheid, Muriel Franco, Burkhard Stiller: Proverum: A Hybrid Public Verifiability for Decentralized Identity Management; IFI Technical Report 2020.03, Zürich, Switzerland, August 2020.

- *Journal*. **Bruno Rodrigues**, Eder Scheid, Christian Killer, Muriel Franco, Burkhard Stiller: Blockchain Signaling System (BloSS) Cooperative Signaling of Distributed Denial-of-Service Attacks. Journal of Network and Systems Management (JNSM), Special Issue on Future of Network and Service Operations and Management: Trends, Developments, and Directions. Springer's.

- *Full paper*. Muriel Franco, Erion Sula, **Bruno Rodrigues**, Eder Scheid, Burkhard Stiller. ProtectDDoS: Trustworthy Offering and Recommendation of Protections. 17th International Conference on the Economics of Grids, Clouds, Systems, and Services (GECON 2020). Online conference.

- *Demo*. Getoar Gallopeni, **Bruno Rodrigues**, Muriel Franco, Burkhard Stiller: A Practical Analysis on Mirai Botnet Traffic; IFIP Networking 2020, Paris, France, 22-25 June, 2020.

- *White Paper*. Alessandro De Carli, Muriel Franco, Andreas Gassmann, Christian Killer, **Bruno Rodrigues**, Eder John Scheid, David Schoenbaechler, Burkhard Stiller: WeTrace: Privacy-preserving Mobile COVID-19 Tracing Approach and Application; Technical Report, Zürich, Switzerland, April 2020.

- *Tutorial*. Christian Killer, **Bruno Rodrigues**, Eder Scheid, Muriel Franco, Burkhard Stiller. Practical Introduction to Blockchain-based Remote Electronic Voting. IEEE International Conference on Blockchain and Cryptocurrency (ICBC 20). 3-6 May. Toronto, Canada.

- *Research Project*. **Bruno Rodrigues**, Thomas Bocek, Simon Tuck, Burkhard Stiller - PasWITS - Passives Wireless Intelligence Tracking System. Innovation projects ICT. Inosuisse. 42193.1 IP-ICT.

- *Full paper*. Scheid, E. J, Widmer P., **Rodrigues B.**, Franco M., Stiller B. A Controlled Natural Language to Support Intent-based Blockchain Selection; IEEE International Conference on Blockchain and Cryptocurrency (ICBC 2020), 3-6 May 2020, Toronto, Canada.

- *Poster*. **Bruno Rodrigues**, Spasen Trendafilov, Eder Scheid, Burkhard Stiller. SC-FLARE: Cooperative DDoS Signaling based on Smart Contracts. IEEE International Conference on Blockchain and Cryptocurrency (ICBC 2020), 3-6 May 2020, Toronto, Canada.

- *Tutorial*. **Bruno Rodrigues**, Eder Scheid, Burkhard Stiller. Blockchains in the Age of Softwarization – Hands-on Experiences with Programming Smart Contracts and Their Security Pitfalls. IFIP/IEEE Network Operations and Management Symposium. 20-24 April. 2020. Budapest, Hungary.

- *Full paper*. Eder J. Scheid, Daniel Lakic, **Bruno Rodrigues**, Burkhard Stiller. PleBeuS: a Policy-based Blockchain Selection Framework. IFIP/IEEE Network Operations and Management Symposium. 20-24 April. 2020. Budapest, Hungary.

- *Full paper*. Christian Killer, **Bruno Rodrigues**, Raphael Matile, Eder Scheid, Burkhard Stiller. Design and Implementation of Cast-as-Intended Verifiability for a Blockchain-based Voting System. The 35th ACM Symposium on Applied Computing (SAC). March 30th to April 3rd, 2020. Brno, Czech Republic.

- *Book Chapter*. **Bruno Rodrigues**, Muriel Franco, Eder John Scheid, Burkhard Stiller, Salil Kanhere: A Technology-driven Overview on Blockchain-based Academic Certificate Handling; in: Ramesh Sharma, Hakan Yildirim, Gulsun Meric (Edt.), "Blockchain Technology Applications in Education", IGI Global, Pensilvania, U.S.A, January 2020, ISBN 978-1-522-59478-9, pp 1–290. doi:10.4018/978-1-5225-9478-9

2019

- *Workshop Paper*. Christian Killer, **Bruno Rodrigues**, Burkhard Stiller. Threat Management Dashboard for a Blockchain Collaborative Defense. 2019 IEEE Globecom Workshops (GC Wkshps): IEEE GLOBECOM 2019 Workshop on Telecommunications and Blockchain. 9-13 December 2019 // Waikoloa, HI, USA.

- *Book Chapter*. Charles Miers, Guilherme Koslovski, Mauricio Pillon, Marcos Simplício, Tereza Carvalho, **Bruno Rodrigues**, João Battisti. Análise de Mecanismos para Consenso Distribuído

Aplicados a Blockchain. Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg). São Paulo, 2 a 5 de Setembro, 2019.

- *Full Paper*. Muriel Franco, **Bruno Rodrigues**, Burkhard Stiller. MENTOR: The Design and Evaluation of a Protection Services Recommender System. 15th International Conference on Network and Service Management (CNSM19). Halifax, Canada, 21-25 October 2019.

- *Full paper*. Eder J. Scheid, Timo Hegnauer, **Bruno Rodrigues**, Burkhard Stiller. Bifröst: a Modular Blockchain Interoperability API. 2019 IEEE 44th Conference on Local Computer Networks (LCN).

- *Full paper*. **Bruno Rodrigues**, Muriel Franco, Geetha Parangi, Burkhard Stiller. SEConomy: A Framework for the Economic Assessment of Cybersecurity; 16th International Conference on Economics of Grids, Clouds, Systems, and Services (GECON 2019), Leeds, UK, September 17-19.

- *Demo*. **Bruno Rodrigues**, Burkhard Stiller. Cooperative Signaling of DDoS Attacks in a Blockchain based Network. ACM SIGCOMM 2019 Conference (SIGCOMM Posters and Demos '19), August 19–23, 2019, Beijing, China..

- *Tutorial*. **Bruno Rodrigues**, Eder Scheid, Roman Blum, Thomas Bocek, Burkhard Stiller. Blockchain and Smart Contracts – From Theory to Practice. IEEE International Conference on Blockchain and Cryptocurrency (ICBC 19). 14-17 May. Seul, South Korea.

- *Short-paper*. Raphael Matile, **Bruno Rodrigues**, Burkhard Stiller. Cast-as-Intended Verifiability in Blockchain-based Electronic Voting. IEEE International Conference on Blockchain and Cryptocurrency (ICBC 19). 14-17 May. Seul, South Korea.

- *Short-paper*. Christian Killer, **Bruno Rodrigues**, Burkhard Stiller. A Proposal for Security Management and Visuualization in a Blockchain-based Collaborative Defense. IEEE International Conference on Blockchain and Cryptocurrency (ICBC 19). 14-17 May. Seul, South Korea.

- *Tutorial*. **Bruno Rodrigues**, Eder Scheid, Burkhard Stiller. Information Management in the Blockchain Era Challenges and Opportunities. 16th IFIP/IEEE International Symposium on Integrated Network Management (IM 2019). 8-12 April. Washington DC, USA.

- *Full paper*. Andreas Gruhler, **Bruno Rodrigues**, Burkhard Stiller. A Reputation and Reward Scheme for a Cooperative Network Defense. 16th IFIP/IEEE International Symposium on Integrated Network Management (IM 2019). 8-12 April. Washington DC, USA.

- *Full paper*. Eder Scheid, **Bruno Rodrigues**, Lisandro Granville, Burkhard Stiller. Enabling Dynamic SLA Compensation Using Blockchain-based Smart Contracts. 16th IFIP/IEEE International Symposium on Integrated Network Management (IM 2019). 8-12 April. Washington DC, USA.

- *Experience paper*. **Bruno Rodrigues**, Lukas Eisenring, Eder Scheid, Thomas Bocek, Burkhard Stiller. Evaluating a Blockchain-based Cooperative Defense. 16th IFIP/IEEE International Symposium on Integrated Network Management (IM 2019). 8-12 April. Washington DC, USA.

- *Short paper*. Eder Scheid, **Bruno Rodrigues**, Burkhard Stiller. Toward a Policy-based Blockchain Agnostic Framework. 16th IFIP/IEEE International Symposium on Integrated Network Management (IM 2019). 8-12 April. Washington DC, USA.

2018

- *Tutorial*. **Bruno Rodrigues**, Roman Blum, Thomas Bocek, Burkhard Stiller. Blockchain and Smart Contracts: From Theory to Practice . 14th International Conference on Network and Service Management (CNSM). CNSM Tutorials. Rome, Italy, 2018.

- *Workshop Paper*. Jerinas Gresch, **Bruno Rodrigues**, Eder Scheid, Salil S. Kenhere, Burkhard Stiller The Proposal of a Blockchain-based Architecture for Transparent Certificate Handling. BSCT 2018: 1st Workshop on Blockchain and Smart Contract Technologies. Berlin, Germany, July 18, 2018.1,2

- *Full paper*. Stephan Mannhart, **Bruno Rodrigues**, Eder Scheid, Salil S. Kanhere, Burkhard Stiller. Towards Mitigation-as-a-Service in Cooperative Network Defenses . The 3rd IEEE Cyber-Science and Technology Congress.

- *Book chapter*. **Bruno Rodrigues**, Thomas Bocek, Burkhard Stiller: The Use of Blockchains: Application-Driven Analysis of Applicability; in: Pethuru Raj, Ganesh Deka (Edt.), "Blockchain Technology: Platforms, Tools and Use Cases, Volume 111 (Advances in Computers)", Springer, Waltham, MA, U.S.A, No. 111, September 2018, ISBN 978-0-128-13852-6, pp 1–22.

2017

- *Journal*. Riekstin, A. C., **Rodrigues, B.B.**, Nguyen, K. K., Carvalho, T., Meirosu, C., Stiller, B., Cheriet, M. A Survey on Metrics and Measurement Tools for Sustainable Distributed Cloud Networks, vol. PP, no. 99, pp. 1-1. doi: 10.1109/COMST.2017.2784803, in Communications Surveys and Tutorials, IEEE COMST, 2017.

- *Demo* - **Bruno Rodrigues**, Thomas Bocek, Burkhard Stiller. Blockchain Signaling System (BloSS): Enabling a Cooperative and Multi-domain DDoS Defense. Demonstrations of the 42nd Annual IEEE Conference on Local Computer Networks (LCN-Demos 2017). Singapore, 8-12 October.

- *Journal*. - Thomas Bocek, Sina Rafati, **Bruno Rodrigues**, Burkhard Stiller. CoinBlesk - A Real-time, Bitcoin-based Payment Approach and App. ERCIM News - Special Issue: Blockchain Engineering, Vol. 2017, No. 110, July 2017, pp 14-15.

- *PhD Workshop*. - **Bruno Rodrigues**, Thomas Bocek, Burkhard Stiller. Multi-domain DDoS Mitigation Based on Blockchains. 11th International Conference on Autonomous Infrastructure, Management and Security (AIMS 2017). "Security of Networks and Services in an All-Connected World", Zürich, Switzerland, July 2017, ISBN 978-3-319-60773-3, pp 159–164.

- *Full paper*. **Bruno Rodrigues**, Thomas Bocek, David Hausheer, Andri Lareida, Sina Rafati, Burkhard Stiller. A Blockchain-based Architecture for Collaborative DDoS Mitigation with Smart Contracts and SDN. 11th International Conference on Autonomous Infrastructure, Management and Security (AIMS 2017). "Security of Networks and Services in an All Connected World", Zürich, Switzerland, July 2017, ISBN 978-3-319-60773-3, pp 16–29.

- *Poster *Not peer-reviewed**. **Bruno Rodrigues**, Thomas Bocek, David Hausheer, Andri Lareida, Sina Rafati, Burkhard Stiller: Blockchain-based Architecture for Collaborative DDoS Mitigation using Smart Contracts; 1st ForDigital Blockchain Conference, "ForDigital Blockchain", Karlsruhe, Germany, February 2017.

- *Experience paper*. Thomas Bocek, **Bruno Rodrigues**, Tim Strasser, Burkhard Stiller: Blockchains Everywhere - A Use-case of Blockchains in the Pharma Supply-chain; 15th IFIP/IEEE International Symposium on Integrated Network Management (IM 2017), "15th IFIP/IEEE International Symposium on Integrated Network Management (IM 2017)", Lisbon, Portugal, May 2017, pp 1–6.

- *Full paper*. Andri Lareida, Romana Pernischova, **Bruno Rodrigues**, Burkhard Stiller: Abstracting .torrent Content Consumption into Two-mode Graphs and their Projection to Content Networks (ConNet); 2017 IFIP/IEEE International Symposium on Integrated Network Management (IM), "2017 IFIP/IEEE International Symposium on Integrated Network Management (IM)", Lisbon, Portugal, May 2017, pp 1–9.

List of Figures

List of Tables

List of Listings

Curriculum Vitae

Personal Details

Name	Bruno Bastos Rodrigues
Date of Birth	July 03, 1989
Place of Birth	Brasília, Distrito Federal (DF), Brazil

Education

August 06 – January 13	Bachelor in Science (B.Sc.) Center of Technological Sciences University of the State of Santa Catarina
January 14 – August 16	Master in Engineering (M. Sc.) Polytechnic School University of São Paulo
September 16 – February 21	Doctoral program at the University of Zurich Department of Informatics Communication Systems Group

Professional Experience

July 12 – December 13	Internship on Network Infrastructure Support Public Ministry of Santa Catarina State
January 13 – July 14	Internship on Software Development Staff Informatica Ltda.
January 14 – August 16	Research Assistant University of São Paulo
October 16 – December 20	Research Assistant University of Zurich