



University of
Zurich^{UZH}

Cooperative Signaling

Spasen Trendafilov
Zurich, Switzerland
Student ID: 15-712-805

Supervisor: Bruno Rodrigues, Eder Scheid, Prof. Dr. Burkhard
Stiller

Date of Submission: August 26, 2019

Abstract

Mit der ständig steigenden Anzahl von Geräten, die auf das Internet zugreifen, und der Verbreitung von Internet of Things (IoT) werden böswillige, groß angelegte Distributed Denial of Service (DDoS)-Angriffe durch die Anzahl der ungeschützten Geräte unterstützt. Strategisch fortgeschrittene DDoS-Angriffe mit zentralisierter Verteidigung erfordern eine enorme Menge an Hard- und Software Ressourcen um einen solchen Angriff abzuwehren. Ein verteilter kooperativer Ansatz zur Kombination der Erkennung und der Verteidigung verschiedener Domänen könnte jedoch den Aufwand vor Ort reduzieren und den Angriff in der Nähe der Quelle blockieren [45].

Diese Bachelorarbeit beschreibt die Implementation und Evaluation eines Prototypen des verteilten kooperativen Signalisierungsprotokolls und eines Register Smart Contracts für die Kernsignalisierung des Blockchain Signaling System (BloSS) [40]. Das Register dient als Eintrag für mehrere Verteidiger der DDoS Attacke, während das Opfer einer DDoS Attacke das Protokoll einsetzt und einen Verteidiger im Register findet, um den Verteidigungsprozess zu starten. Während der Durchführung des Protokolls können die teilnehmenden Akteure möglicherweise nicht online sein oder einen definierten Termin absichtlich verpassen, was dazu führt, dass eine Vielzahl von Szenarien möglich ist. Außerdem kann die gegenseitige Bewertung durch das Opfer und den Verteidiger negativ, nicht verfügbar oder positiv sein, und je nach Szenario ist die Bewertung entweder irrational, egoistisch oder rational. Schließlich wird je nach Szenario der monetäre Anreiz aus dem Protokoll entweder an das Ziel zurückerstattet oder als Belohnung an den Verteidiger transferiert.

Bei der Auswertung der Leistungstests mit der privaten Blockchain Ganache sowie dem öffentlichen Testnet Rinkeby durch Infura als Proxy und in einem zweiten Schritt mit dem Geth-Client, werden *Gas verbrauch* und *Prozess Zeiten* verglichen. Durch die Auswertung der Ergebnisse aus der Testsuite soll das Vertrauen in die *Richtigkeit*, *Sicherheit* und *Skalierbarkeit* der kooperativen Signalisierung erreicht werden.

With the ever-increasing number of devices accessing the internet and the prevalent adoption of Internet of Things (IoT) devices, malicious large-scale Distributed Denial of Service (DDoS) attacks are empowered by the number of unsecured devices. Withstanding a sophisticated DDoS attack with centralized defenses requires an enormous amount of hardware and software resources. However, a distributed cooperative approach to combine the detection, as well as mitigation capabilities of different domains, could reduce the overhead at a single point and block adversary traffic near the source [45].

This thesis proposes a prototype of a distributed Cooperative Signaling Protocol and Register Smart Contract for the core signaling of the Blockchain Signaling System (BloSS) [40]. The Register serves as an entry for multiple mitigators, whereas the target deploys the Protocol and finds a mitigator on the Register in order to start the mitigation process. While running the Protocol for the mitigation, participating actors may not be online or miss a defined deadline on purpose, creating a variety of possible scenarios. Also, rating by the target and by the mitigator may be negative, not available or positive, and depending on the scenario, the rating decision is either irrational, selfish, or rational. Finally, depending on the scenario, the locked up incentive from the Protocol is either refunded to the target or transacted as a reward for the mitigator.

Evaluating the performance tests using the private blockchain Ganache as well as the public testnet Rinkeby through Infura as a proxy and in a second step the Geth client, *gas usage* and *processing times* are compared. By evaluating the results from the test suite, confidence in regards to the *correctness*, *security*, and *scalability* of the Cooperative Signaling is to be reached.

Acknowledgments

I would like to take this opportunity to express my gratitude towards my supervisors Bruno Rodrigues, Eder Scheid and Prof. Dr. Burkhard Stiller at the Communication Systems Group of the University of Zurich. Especially, I want to thank Bruno Rodrigues for answering my questions, discussing complications and allowing me to work in a constructive environment.

I also want to thank my family for the support, patience, and for the motivational persuasion on my journey of writing this bachelors thesis.

Contents

Abstract	i
Acknowledgments	iii
1 Introduction	1
1.1 Motivation	2
1.2 Thesis Goals	2
1.3 Methodology	2
1.4 Thesis Outline	3
2 Background	5
2.1 Blockchain	5
2.1.1 Consensus Algorithms	5
2.1.2 Proof-of-Work (PoW)	6
2.1.3 Proof-of-Authority (PoA)	6
2.2 Ethereum	6
2.2.1 Ethereum Virtual Machine	7
2.2.2 Smart Contract	7
2.3 Related Work	7
2.3.1 IETF DOTS	8
2.3.2 CoFenCe	9
2.3.3 Bohatei	9
2.3.4 DefCOM	9
2.3.5 Discussion	10

3	Prototype Design	13
3.1	Blockchain Signaling System (BloSS)	13
3.1.1	Blockchain Performance	14
3.1.2	Blockchain as an Enabler of Trust	16
3.1.3	Existence of Financial Incentives	18
3.2	Architecture	19
3.3	Signaling Process	20
3.3.1	Register	20
3.3.2	Cooperative Signaling Protocol	21
3.3.3	Implementation	24
3.4	Requirements for the Prototype	28
3.4.1	First Stage	29
3.4.2	Second Stage	29
3.4.3	Third Stage	29
4	Evaluation	31
4.1	Evaluation Setup	31
4.2	Ganache	32
4.3	Rinkeby Local	35
4.4	Rinkeby Global	36
5	Summary and Conclusions	43
	Bibliography	45
	Abbreviations	51
	Glossary	53
	List of Figures	53
	List of Tables	55

<i>CONTENTS</i>	vii
List of Listings	57
A Contents of the CD	61

Chapter 1

Introduction

A Distributed Denial-of-Service (DDoS) attack is a large-scale, coordinated attempt to render a target system's resources unavailable to legitimate users. Although being a widely known type of attack that dates back to the beginning of the Internet, it remains as one of the major causes of concerns for service providers around the world, and more recently, nations are concerned with cyber threats. The main reason that drives the escalation of these attacks is the increasing availability of stationary and portable devices ranging from small sensors to baby cameras and home gateways connected to the Internet (*i.e.*, Internet of Things - IoT devices), that empower attackers to exploit a large number of unsecured devices to generate malicious attacks on operational machinery and systems.

Botnets taking advantage of the lack of security of these devices are the primary basis for these large-scale attacks. The most prominent example is the Mirai botnet [2], 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 TB/s. For example, the first appearance of Mirai was on September 2016, when it peaked 623 GB/s in volume of traffic in an attack against Krebs Security [36]. Akamai, the company hosting the Web site, had to shut down this site because defending it during three days became too costly [1]. It was reported that so many devices were used that attackers did not have to use any sophisticated strategy [41]. A month later, on October 2016, the attack on DynDNS [36] peaked 1.2 TB/s, resulting in the unavailability of major Internet platforms and services (*e.g.*, Twitter, GitHub, PayPal, and Spotify) by rendering their Domain Name System (DNS) servers unavailable.

As a consequence, these and other attacks triggered an alert in the cybersecurity community. Nations were concerned with the dimensions reached by these large-scale attacks, and the possibility of these attacks being used as a weapon to render attacks 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 cyberspace as the fifth dimension of their national security along with air, land, sea, and space operations. Therefore, as DDoS attacks become progressively more distributed and coordinated, the defense from such attacks likewise requires distribution and coordination.

To prevent or reduce the damage induced by DDoS attacks, various exposure, and mitigation methods can be used as a countermeasure. For instance, an in-house defense is implemented by the target system requiring dedicated hardware to analyze flow records and filter adversary traffic. An alternative to in-house defense is a cloud-based service, such as the ones offered by Akamai [1] or CloudFlare [10]. Acting as a proxy to load balance, reroute, or drop the adversary traffic. However, centralized defenses still lack hardware and software capabilities to counter large-scale DDoS attacks. Thus, if an attack is highly sophisticated and there are no countermeasures available, legitimate users may be impaired until the attack stops. An alternative to the centralized point of view is sharing the hardware and defense capabilities with other systems, an approach called cooperative DDoS mitigation. However, existing cooperative approaches involve the proposal of a particular distributed architecture and Protocols that usually require the modification of existing hardware and software in its support [41].

1.1 Motivation

Defending against large scale DDoS attacks on-premise is quickly becoming infeasible due to the increasing volume of traffic achieved by these attacks. However, a decentralized hardware agnostic, cooperative approach could enable more scaling and efficiency in order to defend a DDoS attack. A cooperative defense is an alternative to cope with large-scale DDoS attacks, where the mitigation takes place at the egress point of the attack. Advantages over traditional/on-premise defenses have been widely recognized in the literature [29, 33, 50]. It allows, for example, to combine detection/mitigation capabilities of different domains, to reduce the detection/mitigation overhead at a single point, and to block malicious traffic near its source. However, there is still no widespread deployment of such a cooperative defense system.

1.2 Thesis Goals

The goal of this thesis is to implement and test the Cooperative Signaling on an Ethereum Blockchain for cooperative DDoS mitigation. Based on the Blockchain Signaling System (BloSS), a Register and its Signaling Protocol [8] have to be extended. Rigorous testing from stage to stage ensures *correctness*, *security* and finally *scalability* for the Cooperative Signaling between the actors and Smart Contracts. Testing of this system has to be done locally as well as in geographically different locations, allowing for realistic *performance* results as well as operating costs.

1.3 Methodology

Based on the aforementioned goals, it is noted that this thesis has an important practical dimension, which is sought, through a bibliographic research, extend the current BloSS

signaling Protocol to a fully distributed approach. Therefore, this work is based on the hypothetical inductive method based on the existing code as well as related approaches described in references. In summary, the following steps define the process of this bachelor's thesis in order to achieve the goal:

1. Problem analysis and planning: In a first step comprehensive knowledge about BloSS and signaling systems is to be obtained. Possible signaling platforms have to be analyzed in order to benefit from the advantages of BloSS and the Ethereum blockchain.
2. Requirements and System Design: In a second step formal requirements for the selected signaling system as well as the measurement parameters for the evaluation of the signaling system are proposed.
3. Design and development of the prototype: In a third step a prototype consisting of exchange of messages as illustrated in the Cooperative Signaling Protocol as well as a management Register for multiple signaling Protocols have to be implemented. The prototype has to be deployed on the Ethereum blockchain and allows for interaction between multiple instances (target and mitigator).
4. Evaluation: With the measurement parameters *gas usage* and *processing time*, in a fourth step, concise evaluation of the implementation against the specification of the signaling Protocol and performance expectations needs to be delivered.

1.4 Thesis Outline

Chapter 2 summarizes used concepts and technologies and discusses related work at the end of this chapter. **Chapter 3** discusses and presents a possible design for a prototype of a Cooperative Signaling system based on the BloSS exchange of messages. The prototype consists of two Smart Contracts which offer different functionalities and services to a target and a mitigator. The Protocol Smart Contract offers specific interactions for target and mitigator, and the Register allows for registration of targets, mitigators, and Protocols in order to manage multiple combinations. **Chapter 4** explains the process of testing the performance on critical measurements and summarizes the results of the local tests as well as on two separate geographically separated instances. **Chapter 5** summarizes the prototype's gas usage and its performance depending on which scenario is executed on the presented blockchain variants. A conclusion and future work on the extended and evaluated prototype is suggested as the last part of this thesis.

Chapter 2

Background

This chapter provides an overview of the concepts and technologies used in the following work. Starting from Section 2.1 with the blockchain technology and the consensus algorithms Proof-of-Work (PoW), as well as Proof-of-Authority (PoA) in the following Section 2.2 the concept of Smart Contracts is presented. Finally, in Section 2.3 related DDoS mitigation concepts are mentioned, compared and discussed.

2.1 Blockchain

The blockchain technology is a publicly available, decentralized data storage [30, 42]. Different types of blockchains, for instance, public, private, and consortium-based blockchains, are used in different use cases. In public blockchains, any willing participant may look into and participate in the permissionless consensus process and use the blockchain whereas in consortium-based approaches the blockchain itself provides an intermediary level of trust with pre-defined, permissioned, authorized nodes [24]. Peers of a blockchain may generate transactions of digital assets which are collated blocks. Time-stamps of the transactions are cryptographically hashed into the prevalent blocks which build the blockchain. The public chain of data blocks where each block is chronologically referenced to the previous block, allows miners to create new blocks and update the shared chain on the network nodes. If such a block is generated multiple times, the consensus algorithm decides which branch of the fork is interpreted as the valid chain.

2.1.1 Consensus Algorithms

Public blockchains can be monitored by anyone and are not controlled or owned by any instance which creates the need for consensus algorithms. Different implementations of consensus algorithms with different use cases exist. The Ethereum blockchain is currently using PoW whereas the testnet Rinkeby is using PoA. The permissioned Rinkeby testnet is chosen for the prototype to be deployed and tested on, since no costs and relatively short block times are enabled through PoA, compared to the Ethereum mainnet.

2.1.2 Proof-of-Work (PoW)

PoW is a computational based consensus algorithm in which miners try to solve a mathematically difficult problem which is efficiently verifiable [7, 30]. Solving the problem first leads to the privilege of attaching the block to the blockchain and therefore receiving a reward for the work done. Finding this mathematical solution to the problem requires mining power in the form of investing CPU time. Changing already existing blocks cannot be done without redoing all the calculations after the targeted block. Redoing this work, and maintaining the longest chain for validity, would require that node to be at least as fast as the network of nodes combined in order to change a block in the blockchain. Hence, PoW provides security as long as the majority ($>50\%$) of nodes participating in the mining process are honest [17].

2.1.3 Proof-of-Authority (PoA)

A more energy-efficient, reputation-based consensus algorithm is PoA [13]. By allowing pre-defined authorities to be able to use their reputation, instead of a receipt of invested energy to attach a block, PoA is a more scalable approach in contrast to PoW [15]. However, authorities must be defined and therefore trusted to create and sign blocks, this approach is mostly used in private blockchains and is, therefore, more centralized as there is only a set of authorities that are able to sign and create a block.

2.2 Ethereum

Ethereum is a blockchain-based project which additionally offers a built-in Turing-complete programming language [49]. The underlying blockchain uses PoW to find consensus in scenarios where different versions of the blockchain exist. Solving the mathematical problem is rewarded with Ethereum's cryptocurrency ether (ETH) which is paid by the user interacting through a transaction or a state change within a Smart Contract. With Smart Contracts allowing for algorithmic enforcement of defined agreements, Ethereum offers the possibility to create decentralized applications of any arithmetic complexity.

The Ethereum blockchain started in 2014 was initiated by Vitalik Buterin. It uses a Javascript-based contract language called Solidity that runs in a sandboxed Ethereum Virtual Machine (EVM) [49]. Every operation executed in the EVM has to be paid for to prevent Denial-of-Service (DoS) attacks [4]. For example, storing a 256bit word costs 20'000 gas plus its memory usage, making storage an expensive operation, *e.g.*, storing 10 words cost at least 0.038 USD with the gas price of 20 Gwei and the current ETH price in Dollar exchange rate of 216 USD [11] as of writing this thesis. With a Turing-Complete language in the blockchain, new types of contracts can be created, *e.g.*, a fully distributed digital organization, such as the Decentralized Autonomous Organization (DAO) or storing rules to mitigate DDoS attacks.

2.2.1 Ethereum Virtual Machine

One of the most popular blockchain technologies that enable Turing-complete contract language is Ethereum [49]. To restrain DoS attacks, operational costs are required to be paid in the EVM. Furthermore, since the data should be stored and accessed worldwide, only public blockchains will be considered. The most popular is Ethereum, which has the second highest market capitalization of 1 billion USD after Bitcoin with 18 billion USD (at the time of writing). Counterparty or Rootstock allows to run the same contracts as Ethereum, but relies on the Bitcoin blockchain or can interact with Bitcoin transactions. Since Ethereum has been battle-tested recently with DoS attacks [4], the DAO hack, and four hard forks, it will be used as the underlying blockchain.

2.2.2 Smart Contract

Smart Contracts are digital software contracts which can be deployed on the Ethereum blockchain. After deployment on an Ethereum test network or the main network, immutability, in the sense that further variables or functions cannot be added or removed to that contract, is given. Although changes within the state of that contract itself may occur, *i.e.*, unlocking funds of a Smart Contract when a service is completed successfully, further code or more conditions may not be added because of immutability. However, without the underlying blockchain used as an infrastructure to implement, verify as well as ensure and enforce the negotiation or performance of a contract, a Smart Contract could be manipulated *i.e.*, changed and, therefore, lose its vital feature.

Ethereum runs contracts which are programs written in languages such as Solidity [49]. Contracts are made of functions which are defined by a set of instructions created by users. Contracts can transfer Ether between users and other contracts and are stored in the blockchain. Considering the economic value of contracts and various types of attacks that can affect the contracts, it is crucial to provide security in execution infrastructure.

There are different types of use cases for Smart Contracts such as voting, crowdfunding, multisignature-wallets, and applications for selling and buying products or services. In [7], Smart Contracts have been used to record the temperature of medical production during transportation and, for instance, decentralized sharing of applications for smartphones is enabled using smart contracts and the Ethereum blockchain.

2.3 Related Work

This Section provides an overview of related DDoS mitigation systems as well as a discussion of the related work in Section 2.3.5.

2.3.1 IETF DOTS

DDoS Open Threat Signaling (DOTS) orchestrates high-level interactions between different DDoS mitigation Protocols. The DOTS client, which is accessible by the attacked target requests the DOTS server for mitigation help, and in return, the DOTS server sends back the status of the mitigation process. The DOTS server itself is connected to a mitigator. The DOTS client defines which address is categorized into the drop or accept list for a particular timeframe, and the chosen action is forwarded to the target peer in order to perform the mitigation service. Offering such an Infrastructure as a Service (IaaS) in a network with "drop" and "accept" lists, allows the client to define the timeframe and the mitigator to deliver the quality and cost [29].

Active Internet Traffic Filtering (AITF) adds filtering mechanisms on routers that are near to the traffic source which block certain addresses. The need for extending the filtering into the internet core does not scale and aggregating IP prefix filters to enable more efficient state management at the same time sacrifices legitimate traffic to the DDoS target. Even optimizing the process using route record still requires states to be stored, which leads to decentralizing the filtering mechanism trying to find an AITF-enabled router as close as possible to the source traffic [3].

However, these approaches require that a certain level of trust be established between peers that request and peers that provide a mitigation service. For core Internet Service Providers (ISPs), a trust relationship between peer A and peer B is typically defined in Service Level Agreements (SLAs) to coordinate the mutual exchange of traffic, but a trust relationship becomes complex if a peer A desires to push filtering rules as close to the traffic source and there is not an SLA to coordinate the exchange of these services. In this regard, blockchain and Smart Contracts become an opportunity to express agreements in cooperative network defense.

In order to prevent or reduce damages caused by DDoS attacks, different detection and mitigation methods are available as a countermeasure. A typical implementation is called the in-house defense, which is implemented by the target system (the one being attacked) based on dedicated hardware appliances to analyze flow records and further filter malicious traffic or perform load balancing. An alternative is to use a cloud-based service such as the ones offered by Akamai [31] or CloudFlare [10]. These can take away the burden of detection and mitigation at the target serving as a proxy to load balance, reroute, or drop malicious traffic before it reaches the target. However, centralized defenses still lack hardware and software capabilities to counter large-scale DDoS attacks. Thus, if an attack is highly massive in terms of traffic or highly sophisticated, no countermeasure might be available at the target domain. Therefore, the alternative is the cooperative defense in which domains aligned with the same objective can cooperate to mitigate attacks. In this regard, Section 3.3 shows the proposed Signaling Process that shall establish the basis for cooperation between domains, defining mechanisms to reward and rate these domains.

2.3.2 CoFenCe

CoFence is a DDoS defense mechanism that simplifies the participation and collaboration of network function virtualization (NFV). With resource sharing, CoFence proposes a lower cost and higher flexibility solution compared to centralized approaches by allowing defined NFV-enabled domain networks to help an attacked network with the spare network resource. Therefore, instead of relying on specific hardware, commodity hardware can be used to not only benefit from the elasticity, but also reduce costs of the infrastructure. By directing excessive traffic to one or more trusted external domains, DDoS filtering is applied, and non-malicious traffic is again returned to the attacked domain, which also acts as a proof of mitigation [37]. Due to virtualization of networking components, *e.g.*, creation and upgrading of networks that join the alliance, the collaborative defense system can be extended efficiently.

2.3.3 Bohatei

As DDoS defense in a centralized approach often relies on specific hardware, Bohatei [16] aims to enable more *flexibility* using software defined networking (SDN) and NFV. By separately using *flexibility* to find where defense is needed and *elasticity* to match the scale of the attack, specific network needs can be addressed. This leads to a scalability advantage by utilizing SDN and NFV-based networking for DDoS defense. While in the proof-of-concept implementation an inter-domain DDoS defense is not integrated, incorporating Pushback [21] could enable access routers to relay traffic filtering to the upstream and, thus, allow multi-domain cooperative defense with the flexibility of SDN and NFV.

2.3.4 DefCOM

For the defence of a highly distributed DDoS attack, a similarly distributed defense mechanism is necessary. The DefCOM overlay combines the strengths of already existing defence mechanisms *i.e.*, attack detection, rate limiting and traffic differentiation and simplifies the collaboration between DDoS defense systems. One advantage of DefCOM is the highly distributed overlay communication network. However, the data packets still flow on the data links defined by the routing Protocols [32].

An attack mitigation can be initialized when the overlay is assembled. First, an alarm message is created that contains the IP address of the targeted victim with the defined resource limits. Rate limiters along the three structure from the source of the attack to the destination mitigate the traffic attack. During the attack, AtCK-CONT messages are sent to inform the nodes that the attack has not ended yet. As soon as no more ATCK-CONT messages are read by the peers during a specific timeframe, the rate limiters terminate [32].

2.3.5 Discussion

The IETF DOTS [29] focuses on standardizing a Protocol to facilitate its deployment across the Internet. However, its implementation complexity still exists in distributed and centralized architectures to support the different types of communication. In their architecture, fully automatic mitigation is possible, even across multiple domains by using recursive signaling. While signaling for help, all relevant information about the attack is provided by the client and can be distributed among all IETF DOTS instances. However, the signal, as well as data channels used for communication, can be congested by the attack. The client-server model makes a highly distributed defense network more complicated to set up and maintain.

CoFence [37] is an interesting solution, because it is based on a technology that is easily installed and operated. Once the service provider's infrastructure provides support for the execution of NFVs, a virtual function can be easily installed and operated to participate in the defense. Once operating, circumventing the resource intensive packet inspection by relaying the traffic to be cleaned is also an efficient defense approach. The use of NFV for deployment makes CoFence simple to install and maintain, however, collaboration among CoFence instances is limited to relaying traffic (*i.e.*, black-holing or traffic scrubbing in case of an attack), and thus, large-scale attacks could overwhelm a CoFence instance, rendering it unable to relay traffic for cleaning.

SDN-based solutions allow greater agility to enforce decisions that require a global network view. Bohatei [16] demonstrates the scalability and performance advantages of using SDN in conjunction with NFV and building a DDoS defense system on top of proven, existing mitigation components. Leveraging NFV with SDN makes Bohatei up to 5.4 times more cost-effective compared to fixed defense facilities [16]. Utilizing existing mitigation solutions like Snort and IPtables make sure that proven defense technologies are employed. The implementation, however, does not include cooperative aspects of the mitigation.

DefCOM [32] is one of the first proposals of cooperative network defenses. It uses an overlay network, based on a Peer-to-Peer (P2P) gossip-based Protocol to facilitate coordination among peers with inherent scalability. Their proposal includes pre-defined messages for communication (e.g., attacks signaling and mitigation confirmation) and malicious traffic is routed entirely on the underlying routing protocols, keeping the overlay network open for mitigation purposes. However, the overlay network still relies on the underlying data links and could be congested. Propagation of messages in the overlay network might take a long time based on the size of the network. These works focus on addressing the technical challenge, providing a collaborative defense scheme that facilitate communication among peers. This can be achieved either by using a novel technology (e.g., SDN and NFV) or a novel architecture, such as IETF DOTS [29] and DefCOM [32]. However, other challenges such as economic, social, and legal are not addressed in these proposals mentioned above. The technical challenge is the main challenge to be addressed as it not only should not impose additional costs regarding hardware and software, but also should be simple to deploy and operate. However, it is expected that the technical solution should encompass support so that incentives 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.

Also, many proposals for a cooperative defense are not only limited to the signaling of attack information, offering a complete framework including attack detection, signaling, and mitigation. On one hand, a complete solution has a positive side by offering the entire defense framework not only for the single domain, but also in a cooperative fashion. However, on the other hand, this imposes hardware and software requirements that may restrict the widespread adoption of the solution. As an example, the IETF DOTS [29] proposal has a complex architecture which can make its widespread adoption an issue, but it also has a high power of standardization which can facilitate the adoption of the Protocol in standard networking hardware.

Concerning the use of new technologies, it is important to note that none of the existing approaches explore the Blockchain technology as a platform for signaling attacks. Blockchain systems can foster trusted cooperation, because they operate on the principle of decentralization and not only help to eliminate third party intermediaries, but also provide incentives stimulating the cooperative behavior among service providers. Therefore, Blockchain capabilities can be leveraged for signaling mitigation requests across a Blockchain network in a similar approach than DefCOM [32] and serve as an immutable platform for the exchange of mitigation services, where each participant can express their needs in forms of incentives.

Chapter 3

Prototype Design

This thesis relies on the BloSS schema of message exchange and extends the Cooperative Signaling Protocol from [8]. Fortifying the Protocol by a Register enables management of targets and mitigators and therefore offers more usability by enabling specific searching for a known mitigator. Moreover, by providing a test suite split into the three stages correctness, security, and scalability, the prototype design is used to prove the signaling system a valuable extension in the environment of a blockchain. This chapter delivers more insight into BloSS and the two Smart Contracts building the Cooperative Signaling prototype by focusing on the design in Section 3.1 first, before explaining further details about the architecture in Section 3.2 the implementation. Then in Section 3.4, the testing approach is described in three stages.

3.1 Blockchain Signaling System (BloSS)

BloSS is a platform enabling the signaling of DDoS attacks in a cooperative fashion between a network alliance of organizations. The Protocol that regulates BloSS allows the distribution of incentives in order to boost the cooperative behavior of participating entities, as well as track the reputation of the members involved. The BloSS Protocol is defined as Smart Contracts, deployed in an underlying permissioned Ethereum Blockchain infrastructure. Smart Contracts can be used to signal attacks over several domains while at the same time managing the incentives.

In this regard, the prototype is envisioned to be deployed as a public permissioned instance, enabling the cooperation of members within an alliance. For example, Computer Security Incident Response Team (CSIRT) within the same region would be able to establish a private network, which is transparent only for its cooperative members (*i.e.*, CSIRT consortium). The used consortium based blockchain provides trust by definition, *i.e.*, assuming that shared information about signaled attacks would not be externally exposed. However, it is important to note that trust is an important issue involving the sharing of data between CSIRTs [5]. In this regard, a blockchain-based solution can provide, through its natural transparency, a higher degree of transparency between collaborative instances.

It should be observed, however, that the sole use of blockchain (or any technological solution) cannot ensure trust because it is a subjective concept not being able to be quantifiable/achievable in a straightforward manner. In general, trust is considered as human perception of the security triad Confidentiality, Integrity, and Availability (CIA), which in most cases can be verifiable. Yet, there is no general metric or standard set of properties that would make a particular verifiable indicator be classified in a general trust scheme. Nonetheless, the use of blockchain in a collaborative security context was explored in different industry and academia solutions [9, 18], as a platform for providing higher levels of confidence through transparency and unchanging the record of security-related events/incidents. It is noted, however, that this does not eliminate the need for a prior (or minimal) reliable establishment for the construction of a private network to report incidents, events and indicators. In this sense, solutions like MISP [48] provide a distributed platform, without the benefits of providing incentives or tracking the reputation of members.

In addition, the use of blockchain in the collaborative security context brings benefits that can be discussed and/or incorporated into existing platforms. Hence, this work has as a collateral implication of its main objectives to foment the discussion of alternatives that enable the use of defenses or platforms of intelligence sharing, on a large scale. Specifically, blockchain provide an alternative to established distributed solutions, such as MISP [48], by providing an out-of-the-box platform that can be simple to deploy and configure, being customized through the implementation of cooperative logic in Smart Contracts. Benefits and drawbacks can be summarized in Table 3.1:

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

Table 3.1: Benefits and Drawbacks of a Blockchain-based collaborative platform in the cybersecurity context

Considering the two different perspectives (blockchain vs traditional) of platforms for sharing security information, there are three highlighted points in the Table 3.1. These negative (drawback) aspects are detailed in the following subsections.

3.1.1 Blockchain Performance

The first drawback 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 blockchain 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 [13] and trust is already given in a consortium-based blockchain; 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, another limiting factor in performance (throughput) is the relation between the block size and the propagation delay, *i.e.*, latency. The blockchain throughput is defined as [12]:

$$TP = \frac{blockSize}{propDelay} \quad (3.1)$$

For example, considering a *blockSize* of 1MB and a *propDelay* of 10s, the throughput would be of t 0.1MB/s delivered in epochs defined by the block generation time. Similarly a *blockSize* of 2 MB and a *propDelay* of 10s would result in a twice better throughput t 0.2MB/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 3.1).

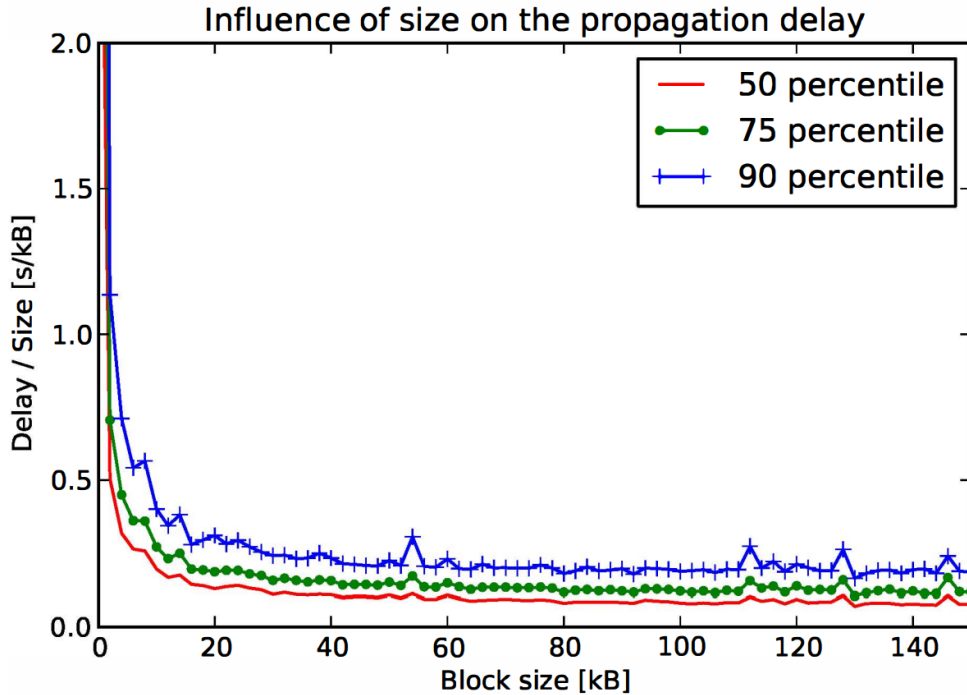


Figure 3.1: Influence of block size on the propagation delay [14]

In Figure 3.1, the authors studied the effects of increasing the block size on the propagation delay. 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 [14]. As reported, for

sizes below 20kB the round-trip delay caused by the Protocol (Bitcoin used as example) causes a major influence on delay, whereas in blocks larger than 20kB, each kilobyte represents an additional of 80ms.

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 seconds based on Ethereum, and a period of time in a day (in seconds) $T = 86400s$, 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 Stored Available in the Period (SAP) is described by:

$$SAP = \frac{T}{blockGenTime} \times blockSize \quad (3.2)$$

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 blockchain solution is the need for storage of blockchain history. This is to ensure that data previously entered into the blockchain 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 blockchain state into a snapshot (which needs to be hashed and compared by all members) and start a new fork of the same blockchain [51]. However, such an approach does not reduce by itself the need for storage, requiring snapshot compression and storage at a more efficient media.

3.1.2 Blockchain as an Enabler of Trust

Trust is a fundamental aspect of any cooperative environment and difficult to obtain since it may rely on many non-technical aspects [20]. In general, trust refers to the assurance and confidence that persons, data, entities, information or processes will function as expected, and, according to ITU-T, it is regarded as a consequence of progress towards security or privacy objectives [47]. Therefore, this 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.

According to [47] *apud* [35], the main characteristics of trust are defined as:

- **Trust is 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.
- **Trust is context-dependent:** trust applies only in a given context. 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). In this regard, trust may exist between organizations A and B only for the "threat indicators" context.

- **Trust is not transitive in nature but maybe transitive within a given context:** That is, 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.
- **Trust is an asymmetric relationship:** Thus, 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.

Therefore, the blockchain technology has a role of a "trust-enabler", providing transparency between cooperative organizations and thus possibly providing trust. However, it is not possible to quantify the role of blockchain as a trust enabler, being not possible to determine a "probability" in which the use of blockchain is a determining factor in ensuring trust between organizations. The role of blockchain in building trust has been studied by [19], in which the authors shed light on how these conflicting notions may be resolved and explore the potential of blockchain technology for dissolving the issue of trust.

Among the various (non-technical) facets of trust, trust in the cooperative platform plays a key role. This has been demonstrated in different e-commerce studies [23, 25], 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 evaluation dimensions to blockchain, we can state that the technology (based on the public deployment model, permissionless with a consensus necessarily open to the participation of all members), meets these requirements. The capability to create an immutable, consensually agreed and publicly available record of transactions is seen as an enabler of trust in the platform [19]. In addition, the definition of rules between participants through smart contracts allows all parties involved to verify the execution of the code that defines the cooperation, *i.e.*, the Cooperative Protocol. 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.

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 blockchain consensus on an equal basis, preventing certain members from *e.g.*, having the ability to censor certain transactions.

Finally, blockchain 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.

3.1.3 Existence of Financial Incentives

The CSIRT community has its current operating model based on trust [46], *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. And currently, there is no exchange of financial incentives for shared information or cooperative mitigation actions in case of DDoS attacks.

The use of blockchain 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 [34].

There are arguments both for and against the use of financial incentives [22, 46]:

- **Against:** one of the arguments against is precisely because of the change in the current model of how information is shared, in which the establishment of incentives could change the trust model among CSIRTs.
- **Favor:** incentives allow for greater engagement of collaborative organizations, working similarly to a bug bounty program. In addition, effectively mitigating a collaborative request demands time from professionals (*i.e.*, security analysts) in other organizations, which opens up an opportunity for free-riders.

Before incorporating incentives into any collaborative (especially financial) platform, it is important to carefully consider these two points of view. As trust is essentially a non-technical aspect and extremely relative to the security perception of cooperative members, establishing financial incentives can create a business model in which companies can change their behavior within the alliance to maximize their profitability. In this way, the old cooperative behavior, based on free will, would cease to exist, and with it, potentially the existing trust scheme.

The lack of financial incentives, however, discourages cooperation in the process of collaborative DDoS attack mitigation. It is important to note that the collaborative process does not end in the signaling, but also in the effective mitigation action (*e.g.*, dropping or blackholing attacking traffic from blacklisted addresses) [39] the required mitigation. This involves the use of resources (human and machinery) from third parties, as well as possible legal consequences of future mitigation acts in cases of false positive. The argument in favor is that there is "a lot to lose and little to gain" 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 blockchain 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.

3.2 Architecture

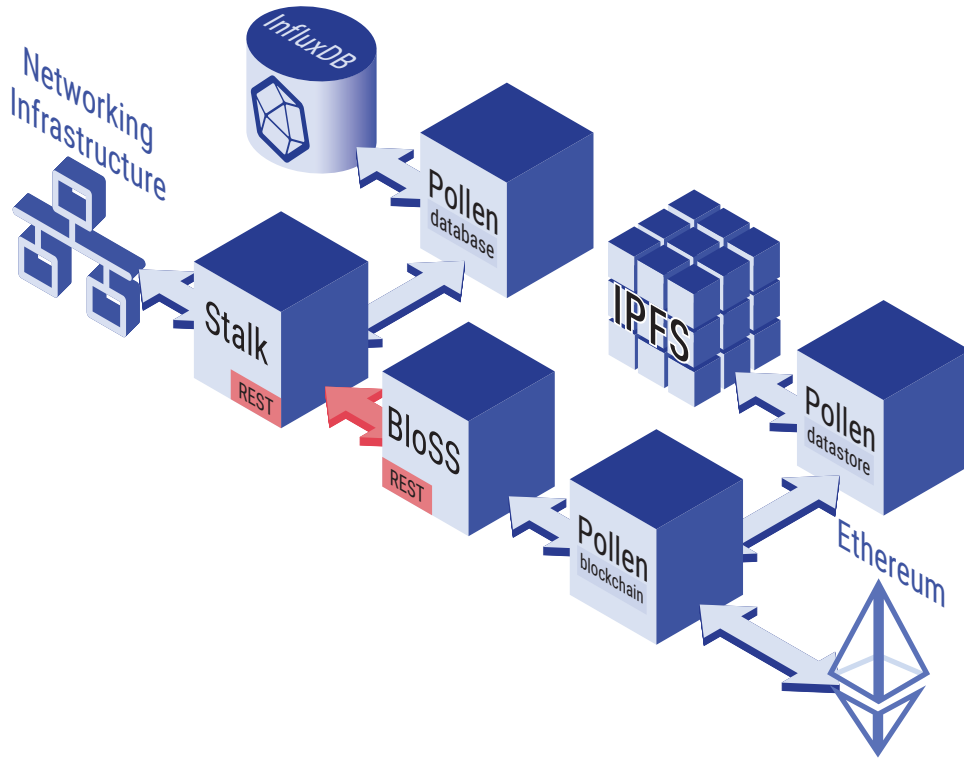


Figure 3.2: Architecture of the Blockchain Signaling System (BloSS) [27]

By running the BloSS component on each service provider that is participating in the cooperative defense, information whether an attack is ongoing or not may be posted to the Ethereum Blockchain. The REST interface is used to ease the separation of the BloSS module. It also leads to the encapsulation of the BloSS module with the Pollen Blockchain and datastore within a Virtualized Network Function (VNF) (*cf.* Figure 3.2). While the data is exchanged through the "Pollen" set of modules, the network-related tasks are handled by the "Stalk" module [27, 44]. Pollen includes modules for both, access to the Ethereum Blockchain as well as a module for managing on the Inter Planetary File System (IPFS) [6].

The blockchain itself is not used to store an attack report since the block size is limited and sensitive information on the blockchain would be public, which is unfavorable.

Instead, IPFS is used which not only offers scalability, but by also running the BloSS, decentralization of the storage is given. In summary, when a new attack has been started, IPFS is used for the data storage, and the hash key of the storage location within IPFS is used in order to store it into a block on the Ethereum blockchain.

Since the Pollen data store is indispensable for the module, it is also encrypted. Moreover, the encryption of the attack information posted to IPFS ensures confidentiality and integrity. While confidentiality is important among service providers because of sensitive information of mitigators or targets, the integrity of service providers ensures that liability of posted information to the blockchain is given and therefore falsification of attack information is impossible. The essential integrity-check consists of a public key published on to the blockchain to group the BloSS defense alliance. Without this feature, adversary actors may find attack information leading to specific IP addresses which could be used to discover the source of an ongoing attack. As a consequence flows from the discovered source to the target could be blocked [40].

3.3 Signaling Process

Before the DDoS defense of a victim (*i.e.*, a target) can be requested, this prototype consisting of a Register and Protocol Smart Contract must be deployed to the blockchain in order to enable the signaling between the defending mitigator and the target. One Register may be used for storing many targets, Protocols, and mitigators. Therefore, the Register is deployed before the Protocol and can be used for searching a specific mitigator. The proposed Protocol, on the other hand, handles the signaling process when a target is requesting defense as a first step of the mitigation process in BloSS.

3.3.1 Register

Storing participating targets and mitigators as well as the Protocol address in the Register as depicted in Figure 3.3, enables searching for mitigators and better management of active processes. This registry type Smart Contract extension is indeed important as it facilitates the process of finding a known mitigator which is already registered, waiting for a target to interact. When that mitigator could not be found in the Register, the default address value is received by the target and further searching for a specific mitigator can be done. But in case the known mitigator is found, the Protocol and target addresses can be observed by the mitigator in order to interact with the Protocol itself. At this point the Protocol has stored the address of the target and the mitigator in order to either allow or disallow access to functionalities or proceeding to change the states of the Protocol.

By allowing reusability of the Protocol and Register more efficiency, in terms of *gas usage* and *performance*, is enabled. After completing a DDoS mitigation, the Protocol's state allows the reuse by reinitializing the Protocol for a new DDoS mitigation with either the same mitigator as in the previous defense process or a new mitigator specified with a name. Therefore there is no need to either redeploy the Register or a Protocol which

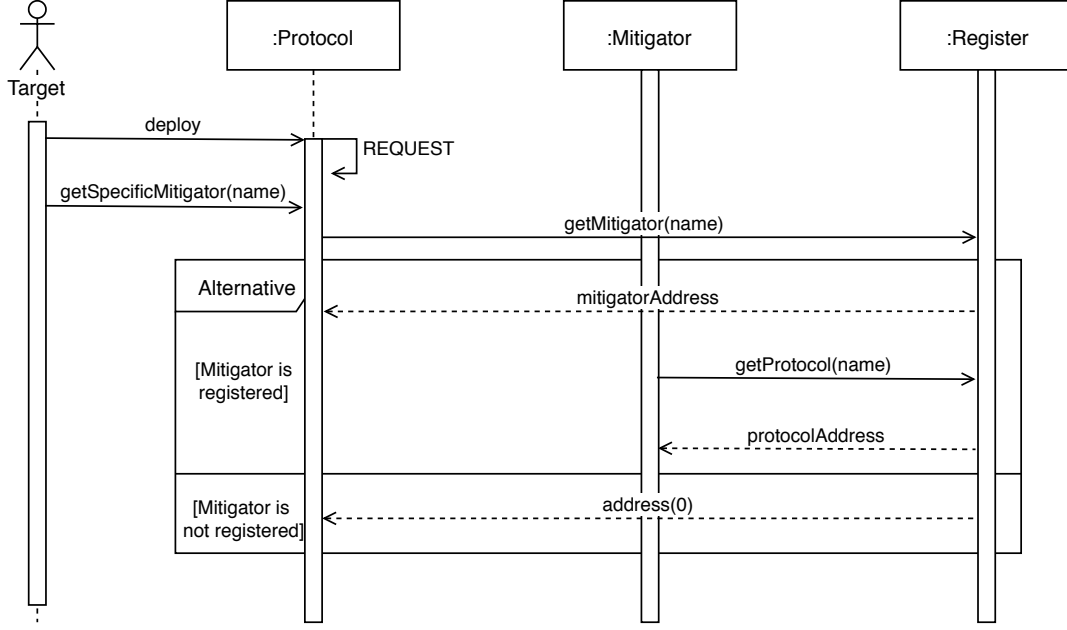


Figure 3.3: Registration Sequence Diagram

leads the deployment *gas usage* being saved. Also, enabling the search for a different mitigator of the following Cooperative Signaling process offers additional flexibility by providing the possibility to change the mitigator address if the newly proposed mitigator is registered. The flexibility of selecting a mitigator without redeployment also enables better *performance* on the blockchain, since one block time (*e.g.*, maximum 15 seconds by using the Rinkeby testnet) can be saved. If the target decides to change the mitigator, the searching process is repeated, and the new mitigator may accept and approve the mitigation request of the target.

It is important to note that the target itself may decide to get the address of a specific mitigator from the Register via the Protocol. As depicted in 3.3, if the mentioned name of the mitigator is registered, the mitigator's address is stored in the Protocol and not directly returned to the target. Although the mitigator is already known by the target beforehand, since the unique name must be sent as a parameter in the *getMitigator* function of the Register, the address of that mitigator is not important for the target at this point of time.

3.3.2 Cooperative Signaling Protocol

The already existing Protocol in [8] is modified and extended by a Register allowing the target and the mitigator to exchange messages to enable an efficient signaling process. But to enable the registration as well as reusability on the side of the target, further functions must be implemented. Also, in order for the functions to be only accessible by an authorized and permitted actor, message sender restrictions must be included.

Figure 3.4 depicts all possible states and transitions depending on the message sender (*i.e.*, caller) of the function. After the deployment of the Protocol on the blockchain, the

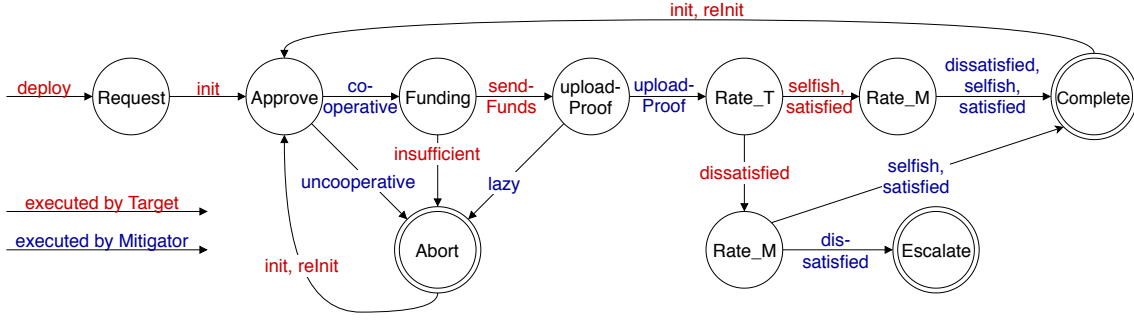


Figure 3.4: State Transition Behavior of the Protocol

default state *Request* is set until the target requests defense from a mitigator 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 Protocol is reused through initializing or reinitializing. During *Approve*, the chosen mitigator may cooperatively accept or uncooperatively deny the request which either leads to the state *Funding* or the endstate *Abort*. While in the *Abort* state, the target may choose to initialize with a different mitigator or reinitialize with the same mitigator to reach *Approve* again. However, if the mitigator is cooperative the *Funding* state begins, and after sending at least the amount of Ethereum defined during initialization, the state switches to *uploadProof* and the funds are locked into the Protocol Smart Contract until completion. The mitigator can upload proof in the form of a report that is used as evidence for work that has been done by the mitigator to mitigate the DDoS attack. If either the deadline interval defined during initialization is missed or there is no upload of a proof at all, the mitigator is marked as lazy, and the updated state is *Abort*. If, however, a proof is uploaded, the rating process of the target and the mitigator begins. During rating, only the case where both actors are dissatisfied leads to the *Escalate* endstate in which the actors themselves must find a solution manually to find consensus upon the service and incentive. All other combinations of the target or mitigator rating satisfied, selfish or dissatisfied, lead to the endstate *Complete*. When *Complete* is reached, the locked funds from the Protocol are released and transferred to either the target or the mitigator, depending on missed deadlines and ratings.

More precisely from the perspective of a target interacting with the mitigator through the Protocol being refunded or rewarded, different strategies and branches are possible. Considering that one party could be selfish or lazy creates possibilities which in each step of the Cooperative Signaling Protocol may lead to a separate rating branch, for instance, the target not sending the funds or the mitigator not uploading a proof of the actual mitigation. Also, if no reaction occurs at all and a deadline is therefore missed, the Cooperative Signaling process continues with no proof uploaded and the ratings being set to selfish. The Cooperative Signaling Protocol depicted in Figure 3.5 shows the possible scenarios and the corresponding deadlines and splits all possible branches from Figure 3.4 into eleven scenarios.

For instance, considering target "T" and mitigator "M" as example actors interacting on the Protocol, the Protocol can lock the funds and decide upon the reached state whether the funds are being kept in the Protocol for further escalation, or if the funds are transferred

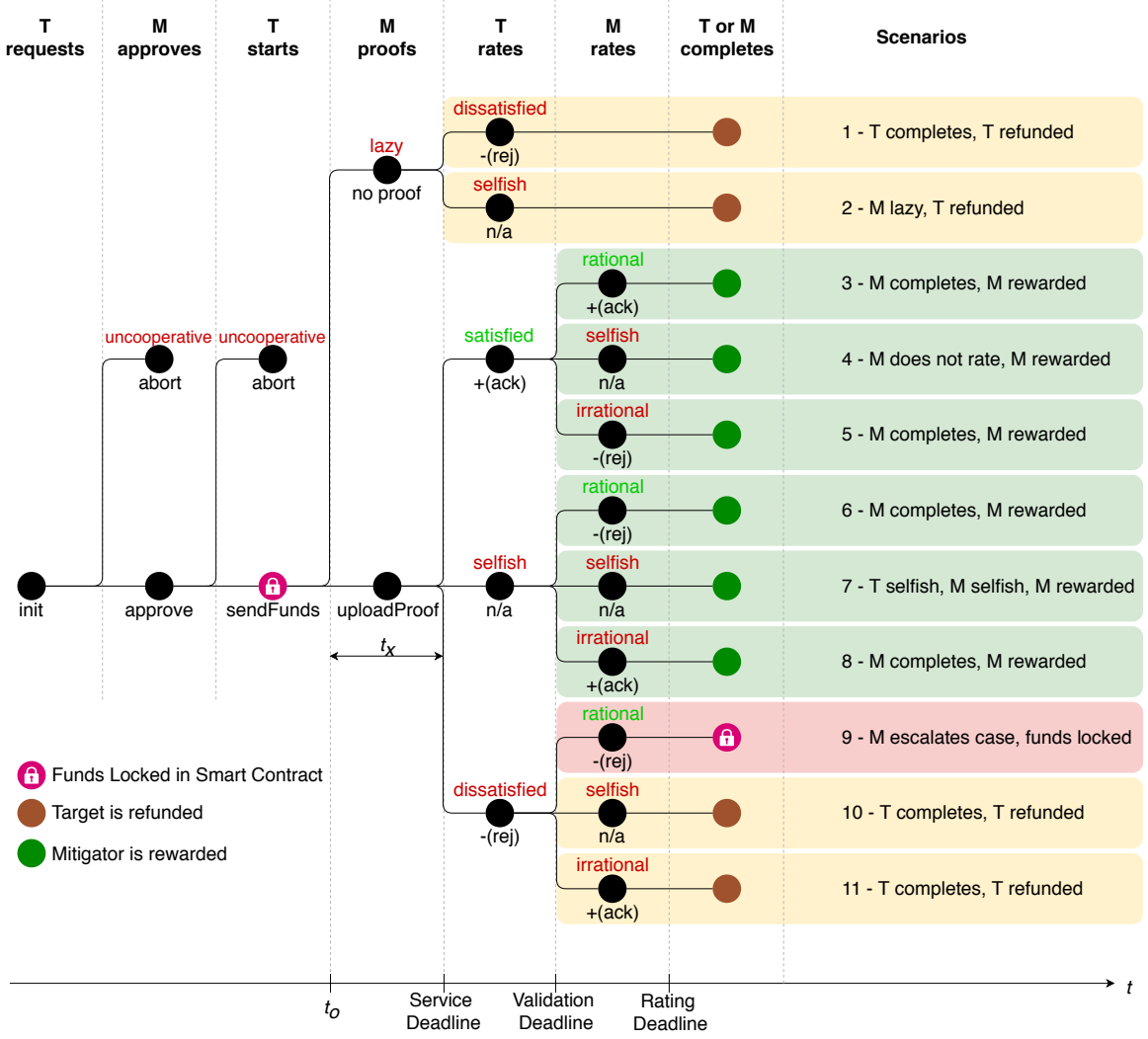


Figure 3.5: Cooperative Signaling Protocol

to one of the actors. The default starting point for T and M is defined as the requesting state, where T requests help from M. After the initialization, M can approve the conditions and the specified minimum incentive from T. Other information, for instance, the list of the networks as well as the deadline interval, starting from t_0 , are also defined during the initialization. After M approves the conditions, T transfers at least the designated funds to the Protocol to proceed to the actual mitigation process, which is a requirement for all scenario branches covered. During this first deadline, M is incentivized to mitigate the attack and upload a proof in the form of a receipt for the proof to be valid. If M misses the deadline of the upload or does not upload at all, M will not be rewarded. The quality of the truthfulness of this proof, however, can not be verified [26, 43] and therefore the lack of ground truth cannot be compensated by the transparency of the blockchain. Not only does this apply to the uploaded proof but also to the following ratings by T and M, which lead to the ending states. In the first scenario, T is dissatisfied as a result of M not uploading a proof, thus leading to the funds being refunded to T. Even when there is no reply from T, after no proof has been uploaded, T is refunded as depicted in the second scenario. In order to remove the possibility of ballot stuffing where T would rate satisfied

in case there is no proof uploaded, T is not allowed to rate positively in this particular case. If however a proof is uploaded before the service deadline ends, T can rate satisfied, be selfish or dissatisfied. In any of those cases, M can react and rate rational, be selfish or rate irrational, which enables further possibilities, although only three ending states can be reached after locking the funds.

The third scenario shows the basic case in which T rates satisfied and M reacts rationally with a positive rating as well, leading to the funds being transferred to M. Similar to this base case, each *scenario* from 4 to 8 of T rating satisfied or being selfish (*i.e.*, missing the service deadline) leads to M being rewarded, regardless of the rating by M. In case T is dissatisfied with the uploaded proof by M and M reacts in a rational way by rating dissatisfied as well, as depicted in *scenario 9*, the case escalates and further investigation is needed to resolve the case. The *scenarios 10* and *11* both lead to T being refunded due to rating dissatisfied and M either being selfish or rating irrationally satisfied.

Rational and irrational ratings by T or M must be considered. A rational response in *e.g.*, scenario 3 means that the signaling process of T is positive and, therefore, the rating by M is satisfied as well. However, in case T does not rate and misses the deadline, a rational response of M would be to rate negatively as depicted in scenario 6. In the case that a proof has been uploaded by M and T rates dissatisfied, the rational response is to rate dissatisfied which leads to an escalation in scenario 9. On the other hand, irrational cases may occur in scenario 5 when M rates dissatisfied, although the rating of T rating was satisfied. A possible use case for this scenario occurs, when the mitigation process consumed more resources and costs than expected, but the locked funds are not sufficient. However, if M does not agree to the initialized terms, the process could have been aborted during *Approve*. A different possibility to rate irrational can be observed when T misses the *Validation Deadline* and acts selfish, depicted in Figure 3.5. In this case the irrational rating is positive shown in scenario 8, and when T rates dissatisfied in scenario 11, M would irrationally rate satisfied and the funds would be refunded to T.

3.3.3 Implementation

The existing Protocol Smart Contract is extended by a feature that is able to call the Register for a specific mitigator by searching in the mapping of the string name in the Register. Extending and redirecting states as well as adding a prevention for ballot stuffing directly into the Protocol, are rather important additions. First the address of the Register can be set within the Protocol. Depending on further implementation the address of the Register can also be defined as a private constant in the Protocol, therefore eliminating the need for defining the address with a setter as the Register's address will always exist before any Protocol address.

Getting a specific mitigator (*i.e.*, getting a mitigator from the Register) requires the name parameter to be known by the target that is interacting with the Protocol. If the name and the mitigator address exist without having a Protocol address assigned to the same *struct*, the address of this Protocol is assigned and registered and that mitigator address is returned to the Protocol.

```

1  function setRegister(address _Register) public {
2      require(msg.sender==Target, "[setRegister] sender is not
        required actor");
3      RegisterAddress = _Register;
4  }
5
6  function getSpecificMitigator(string memory _name) public {
7      require(msg.sender==Target, "[getSpecificMitigator] sender is
        not required actor");
8      reg = Register(RegisterAddress);
9      // cast mitigator address from address to payable address
10     Mitigator = address(uint160(reg.getMitigator(address(this),
        _name)));
11 }
12
13 function init(uint _DeadlineInterval,uint256 _OfferedFunds,string
        memory _ListOfAddresses) public {
14     require(msg.sender==Target,"[init] sender is not required actor"
        );
15     require(Mitigator!=address(0),"[init] mitigator is not set.");
16     require(CurrentState==Enums.State.REQUEST ||
17         CurrentState==Enums.State.COMPLETE ||
18         CurrentState==Enums.State.ABORT, "[init] State is not
        appropriate");
19     Target = msg.sender;
20     DeadlineInterval = _DeadlineInterval;
21     OfferedFunds = _OfferedFunds;
22     ListOfAddresses = _ListOfAddresses;
23     CurrentState = Enums.State.APPROVE;
24     emit ProcessCreated(msg.sender,address(this));
25 }

```

Listing 3.1: Protocol Extension to get a Specific Mitigator from the Register

Enabling both Smart Contracts to interact with each other offers advantages, for instance, calling functions or retrieving other information from each other. The second advantage of allowing interactions is that the logic in the Protocol can be cleanly separated from the registration process. Therefore only one deployment of the Register is needed whereas the number of possible deployments of the Protocol is not limited, allowing every target to deploy such a Protocol and search for a mitigator in the Register. In the context of this Bachelor's Thesis and the BloSS exchange of messages, the callee would be the Register, and the caller would be the Protocol. There exist two approaches to call a function of a callee contract from a caller contract.

```

1 RegisterAddress.call.gas(62500).value(1 ether)("Register", "aName");

```

Listing 3.2: Call Executes Low Level Function of the Callee Contract

Calling a contract function with the address i.e. executing the function directly with the low-level call may be used for interacting with the callee. It must be mentioned that the callee contract may change and therefore also manipulate the storage in the caller contract which can be a security risk if no further authorization is implemented before using `delegatecall`. Both these direct interaction calls do not return any value which would facilitate the usability of the Protocol. Therefore a direct, low-level, call or `delegatecall` may not be an optimal solution for this specific use case.

```

1 contract Register {
2     function getMitigator(address _Protocol, string memory _name)
        payable returns(address);
3 }

```

Listing 3.3: Abstract Call Register Function from Protocol to Get a Specific Mitigator

Another approach to retrieve the mitigator from the Register is to create the Register instance locally, using the Register's address. This approach further requires to define the functions executed by the callee as shown in listing 3.3. By providing this abstract function as well as the Register address, function calls from the Register can be executed, and mitigator information can be retrieved through this interaction between the Protocol and Register. Sending the request with the Protocol address itself shown in Listing 3.1 returns the mitigator address which is cast into a payable address with the uint160 wrapper. By using this approach, the instance additionally allows more extensibility if more functions from the Register would be needed, although needing to redeploy the contract again each time a change is done in order to be able to test the functionalities. After the mitigator is found in the Register and its address is returned to the Protocol, the Cooperative Signaling continues on the Protocol's side.

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

```

1 pragma solidity ^0.5.8;
2 import "./Enums.sol";
3
4 contract Register{
5
6     // address of the creator of the Register
7     address private creatorRegister;
8     event LogNotValid(string);
9
10    struct MitigatorStr {
11        address Protocol;
12        address Mitigator;
13        bool isAdded;
14    }
15
16    mapping(string => MitigatorStr) private mitigators;
17
18    constructor() public{
19        creatorRegister = msg.sender;
20    }
21
22    // set new mitigator by a possible mitigator address
23    function setMitigator(string memory _name, address _Mitigator)
        public {

```

```

24         if(mitigators[_name].isAdded == false){
25             mitigators[_name].Protocol = address(0);
26             mitigators[_name].Mitigator = _Mitigator;
27             mitigators[_name].isAdded = true;
28         }else{
29             emit LogNotValid('Mitigator already registered.');
```

```

30     }
31 }
32
33 // gets called by the target via the Protocol and sets Protocol
34 // address to the mapping of the mitigator
35 function getMitigator(address _Protocol, string memory _name)
36     public returns(address){
37     require(mitigators[_name].Mitigator != msg.sender, "[
38         getMitigator] Protocol is not allowed to be a Mitigator."
39     );
40     if(mitigators[_name].isAdded){
41         mitigators[_name].Protocol = _Protocol;
42         return mitigators[_name].Mitigator;
43     }else{
44         emit LogNotValid('Mitigator not registered.');
```

```

45         return address(0);
46     }
47 }
48
49 // gets called by mitigator, returns address of the Protocol
50 function getProtocol(string memory _name) public view returns(
51     address){
52     require(mitigators[_name].Mitigator == msg.sender, '[
53         getProtocol] You have no permission to see this
54         information.');
```

```

55     if(mitigators[_name].Protocol != address(0)){
56         return mitigators[_name].Protocol;
57     }else{
58         return address(0);
59     }
60 }
61
62 function getCreator() public view returns(address){
63     return creatorRegister;
64 }
65 }
66 }
```

Listing 3.4: Register Smart Contract

The Listing 3.4 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.

3.4 Requirements for the Prototype

The testing of this Cooperative Signaling System prototype is reached in three stages (*cf. Figure 3.6*). In the first stage *correctness* of the BloSS based Cooperative Signaling Protocol [8] is to be verified. Possible state transitions as well as necessary transactions including gas costs and account balances must be tested. Assuming the Protocol behaves as expected, in the second stage *security* is to be reached. Using integration tests on Ganache for low response times and Rinkeby for more realistic testing regarding block times as well as changing the parameters to assess potential bugs is to be achieved. Finally, *scalability* is acquired in the third stage by allowing to find mitigators and managing Protocols and its corresponding target in a Register. Especially as soon as a target of a DDoS attack is requesting help, the mitigator must be known and informed about the state in order to take countermeasures and proceed the signaling. Testing the interactions between the Smart Contracts on different platforms *e.g.*, Ganache, Rinkeby and on Geth with a customized execution script for the target as well as the mitigator allows for performance tests and measurements that can be used and compared in the evaluation.

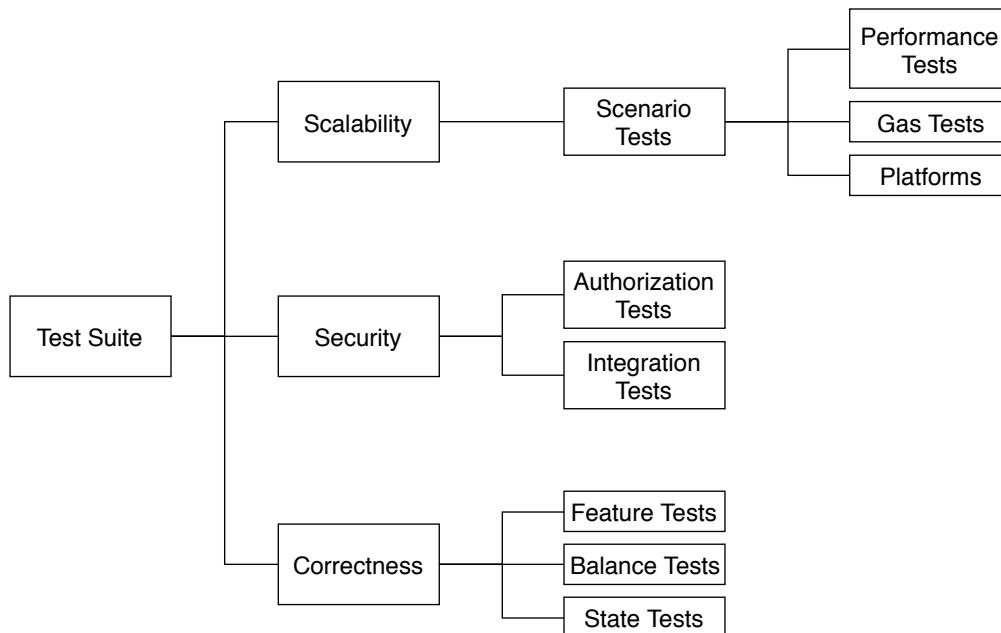


Figure 3.6: Cooperative Signaling Test Suite

Adding a network to the truffle configuration file as for instance Ganache can be done as shown in the following Listing 3.5. The addition of this development network is essential when using the truffle framework since every following used network must be defined and named to be able to choose the name of the network as a option in the command line expression when executing tests or deploying contracts with truffle.

```
1 development: {  
2     host: "localhost",  
3     port: 8545,  
4     network_id: "*", // Match any network id  
5 }
```

Listing 3.5: Localhost Addition to Networks in truffle-config.js

3.4.1 First Stage

Extending the test suite from the Cooperative Signaling Protocol in [8] by unit tests and in-depth scenario testing in integration tests builds the first assessment of the correctness on the underlying Smart Contracts. By utilizing Ganache and running the automining feature which enables instantaneous mining and therefore minimized block times on port 8545, a swift testing cycle is realized.

Not only does Ganache provide a method to remove the block time, it also offers a very convenient user interface with an excellent overview of accounts, blocks, transactions, contracts, events, and logs. Being able to search for specific transactions or contract creations with the gas price of each action attached has been proven to be very useful when searching for bugs and understanding the process flow. Not only the gas price but also the state transitions, balance tests, and feature tests are tested in this first stage to ensure correctness, which is a requirement for the following stages.

3.4.2 Second Stage

Security is the goal to be achieved in the second stage. By using authorization and reverting tests in all integration tests, each "require" statement which defines the party that has access to that function is executed in positive as well as negative possibility branches in order to ensure authorization security. The authorization is tested through require-style exception asserting done within the smart contracts and with accounts that do have access to the functions, for instance, a target starting the Cooperative Signaling with the initialization

3.4.3 Third Stage

Finally, *scalability* is tested and compared between the local and distributed networks. Scalability is only tested after *correctness* and *security* is ensured in order to enable an efficient test suite. Utilizing the PoA testnet Rinkeby for further comparison of the *performance* measures between the customized Ganache block time and the Go implementation of Ethereum (Geth) allows for a valuable variety of testing networks. And with different scenario testing on the mentioned networks processing timings are compared and evaluated on different starting points *e.g.*, the mitigator already being registered and the Protocol having retrieved the mitigator address versus starting from the registration process.

Chapter 4

Evaluation

This evaluation of the extended Protocol and Register Smart Contract should deliver confidence in the Cooperative Signaling on *scalability*, and therefore *performance* and *gas usage*, of the signaling process on different blockchain settings. Correctness as well as security are tested but not evaluated and are not included in this Section, since they are used to enable *scalability* as a last testing step. In order to compare the different infrastructures as well as the realistic usage, the *gas usage*, and the time to finish each scenario is considered in each case. Then a comparison between the times to run each scenario locally, on the testnet Rinkeby via Infura as a proxy, and finally deployed on global nodes, are evaluated.

4.1 Evaluation Setup

The Protocol evaluation involves the scenarios described in Section 3.3.2 to measure the processing time, *i.e.*, time required for a complete execution of each scenario. The test suite, as well as the scenario simulations for Ganache (local set-up) and Infura (proxy for global evaluation), were executed on the same computer and network. The network data-transfer rate was 10Gbit/s and the computer used is equipped with an Intel Core i7 with 4 cores @ 2.6 GHz and 16 GB of 2133 MHz RAM. As for the global evaluation, Amazon Web Service (AWS) instances were deployed in Ohio, Tokyo, and São Paulo. These EC2 *Amazon t2.medium* instances are configured with two threads on either a Intel Xeon or AMD EPYC-Core running at up to 3.0 GHz and 4 GB of RAM. All instances were synchronized with the Rinkeby blockchain in order to enable separation of the target and mitigator, which were also simulated in the local Ganache and Rinkeby Local tests. Each location was tested separately between the target being in Zurich and São Paulo and the mitigator being set to Ohio and Tokyo, respectively.

Depending on how the network is configured and what the average block times are, the test cases are adjusted (*i.e.*, deadlines for specific steps within the Protocol can be adjusted based on real-world latency). For instance, running the *performance* tests on Ganache with the block time set to automining *i.e.*, blocks are automatically generated whenever

there are new transactions and, therefore, the time it takes for the other actors on the network to update the blockchain is minimized. However, this is a completely different behavior than using the Geth client on the Rinkeby network, which is configured for an average block time of 15 seconds. Also, in order for an actor to be tagged as selfish or lazy *i.e.*, to miss a deadline or not upload a proof, the block time must be considered during the initialization of the signaling process. For instance, a deadline interval configured to 5 seconds on a PoA-based blockchain having a 15 seconds block time, would lead to a missed deadline after each block and would therefore not be a valid choice. A 20 seconds deadline interval would suit a 15 seconds block time better since the deadline would not be missed anymore in every block but rather when an actor is lazy or selfish. More precisely, when a missed deadline is simulated, a predefined timeout occurs such that the initialized deadline interval is missed by introducing a wait, which waits longer than the defined deadline interval. Finally, since the truffle test scripts are not executable on Geth, similar scripts were implemented to test the *gas usage* and the *performance* which leads to slight differences in the results although using the same Protocol.

4.2 Ganache

Testing the scenarios for *correctness*, *security* (in terms of authorization and integration tests, as defined in Section 3.4), as well as the best-case *performance* is achieved with the extensive test suite applied to the local personal Ethereum blockchain Ganache. Especially the *gas usage* and *performance* results, independently from the block times, are considered in the evaluation on Ganache.

Using the truffle framework for the deployment of the contracts on the private Ganache blockchain, as well as running the predefined registrations, scenarios and tests on that contract, delivers *gas used* in Tables 4.2 and 4.4. In addition, the averaged processing times of each scenario listed in the Tables 4.1 and 4.3 were also evaluated. The waiting times to simulate a lazy or selfish actor 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 Protocol.

	Average Processing Time [s]	Standard Deviation [s]
Target	0.081	0.012
Mitigator	0.047	0.012
Total Registration	0.128	0.019

Table 4.1: Ganache Processing Times [s] for Register

In this first evaluation of the *performance*, it was found out that the averaging process time for the registration process on Ganache with automining with $n = 20$, takes 0.128 seconds,

and the standard deviation averages at 0.019 seconds. The measured time includes the registration of the mitigator as well as the Protocol searching for a specific mitigator. The times in Table 4.1 represent the best possible performance for the registration process.

	Total Gas Used	Target Gas Used	Mitigator Gas Used
Deployment	952'459	-	-
Registration	145'209	88'866	56'343

Table 4.2: Ganache Gas Used in Gwei for Register

The Register contract will be deployed independently of a specific target or mitigator. It is assumed that all the actors already have the Register address and, therefore, the Register exists before a Protocol has been deployed. So, before the Cooperative Signaling starts, a third actor may deploy the Register and pay for the deployment in the same way the target pays the transaction fee in Formula 4.1. In this evaluation setting however, the target instantiates the Register in the Protocol to run a search of a specific mitigator, 70% of the "one-time registration costs" are paid by the target as shown in Table 4.2.

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

Table 4.3: Ganache Processing Times [s] for Protocol

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 only 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. The Table 4.3 shows that without including block times, the average processing time for the Protocol would be close to 0.37 seconds.

A second 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. It was found that the *gas usage* among the scenarios does not vary significantly when the same rating is given *e.g.*, scenario 3, 4 and 5 as shown in Table 4.4. 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. Since the gas price depends on the implementation, only the Gwei value of the *gas used* offers more comparability across Ganache, local Rinkeby and global Rinkeby.

Scenario	Total Gas Used	Target Gas Used	Mitigator Gas Used
Deployment	3'795'264	3'795'264	-
1	269'837	208'156	61'681
2	269'719	208'556	61'163
3	329'758	184'545	145'213
4	329'810	184'545	145'265
5	330'370	184'545	145'825
6	329'690	184'248	145'442
7	329'828	184'248	145'580
8	330'474	184'248	146'226
9	306'721	184'459	122'262
10	330'036	184'459	145'577
11	330'682	184'459	146'223
Average	326'993	188'770	128'223

Table 4.4: Ganache Gas Used in Gwei for Protocol

Due to the target being the actor that deploys the Protocol, the gas cost is paid by the target only. Once the Protocol is deployed, no further deployment cost is necessary since the same Protocol may be reused for every scenario except scenario nine (*i.e.*, leads to an escalation to be solved by the target and mitigator themselves). In the *gas used* evaluation shown in Table 4.4, the target *gas used* correlates with the different ratings, for instance, the target rating satisfied in scenarios three to five leading to the same amount of *gas used* of 184'545. Similarly, the target gas used when rating selfish or dissatisfied have the same *gas used* values of 184'248 and 184'459, respectively. The *gas used* by the mitigator in the first two scenarios does not reach the final rating by the mitigator since no proof has been uploaded and, therefore, the signaling process leads to refunding the target and results in a lower amount of *gas used*. The following formula is used to transform *gas used* and Gwei to ETH in order to receive the ETH transaction fee in Dollar.

$$TransactionFee[\$] = \frac{GasUsed \times GasCost \times ETHPrice}{10^9} \quad (4.1)$$

Since the gas used does not directly show the transaction fee costs in fiat currency *e.g.* dollar, the formula in 4.1 can be used to determine the transaction fee of the deployment or the interactions afterwards. For the deployment of the Protocol, the actual gas used is 3'795'264. Considering a high priority gas price, for instance, 20 Gwei with the price of one Ether at the point of time of writing this thesis costing \$216 Dollar, the transaction fee would roughly cost \$16.40.

$$\$16.40 \approx \frac{3'795'264 \times 20 \times 216}{10^9}$$

By lowering the gas price to the lowest possible price of 1 Gwei, the transaction fee would roughly result in \$0.82 with the tradeoff that the priority of the transaction might delay the deployment of the Protocol.

$$\$0.82 \approx \frac{3'795'264 \times 1 \times 216}{10^9}$$

4.3 Rinkeby Local

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 blockchain 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 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 4.2 as well as the aforementioned Protocol results on Ganache in Table 4.4. 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 blockchain, allowing the Cooperative Signaling to proceed to a final *endstate*.

	Average Processing Time [s]	Standard Deviation [s]
Target	14.848	0.576
Mitigator	14.115	0.619
Total Registration	28.963	0.810

Table 4.5: Local Rinkeby Processing Times [s] for Register SC

Registering a mitigator and calling the Register from the Protocol to retrieve specific mitigator addresses on the Rinkeby blockchain with $n = 20$, takes an average time of 28.963 seconds with a standard deviation of 0.81 seconds as shown in Table 4.5.

Since the average block time on Rinkeby is defined as 15 seconds and each scenario defined in Figure 3.5 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

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

Table 4.6: Local Rinkeby Processing Times [s] for Protocol SC

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 as depicted in Table 4.6.

4.4 Rinkeby Global

Geth is an Ethereum client implemented in Go. A synchronized node can utilize Geth in order to interact with a deployed Smart Contract. 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 Zurich and Ohio. A control condition is used between the nodes in Tokyo and São Paulo such that the results from Zurich-Ohio can be compared with either a private blockchain, a proxy instead of a full node and finally Tokyo-São Paulo. The focus on these global tests is to measure the *gas usage* and the *performance*, and whether the Register and 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.

From	Ohio			São Paulo			Tokyo		
To	Zurich	São Paulo	Tokyo	Zurich	Ohio	Tokyo	Zurich	Ohio	São Paulo
Average [s]	0.1192	0.13005	0.159	0.2234	0.13035	0.27055	0.276	0.159	0.2701
Std. Dev. [s]	0.0005	0.0002	0	0.0005	0.0011	0.0017	0.0009	0	0.0004

Table 4.7: Round Trip Time [s] between synchronized Nodes in Zurich, São Paulo, Tokyo and Ohio

Table 4.7 lists the Round Trip Time (RTT) test results executed on the AWS instances in Zurich, São Paulo, Tokyo and Ohio. With this table, the RTT times may be compared and used to evaluate whether potential statistical significances across the RTT times may be found during execution of Cooperative Signaling.

Since two instances, namely a mitigator and a target, are considered, the state on the blockchain is to be checked by the actors before continuing to the next state. Figure 4.1 depicts the beginning of each scenario by starting on a first block in a random timeframe, x seconds away from the next block which is set to the first state: *INIT*. It should be noted that x may vary from zero to that specific block time and, therefore, also leads (even with RTT added) to less than the estimated times only considering "full" block times. While the target is initiating and relies on the correct execution of the Protocol by the mitigator, the state is updated and signaled for the mitigator that the approve function is to be executed as a next step in the same fashion.

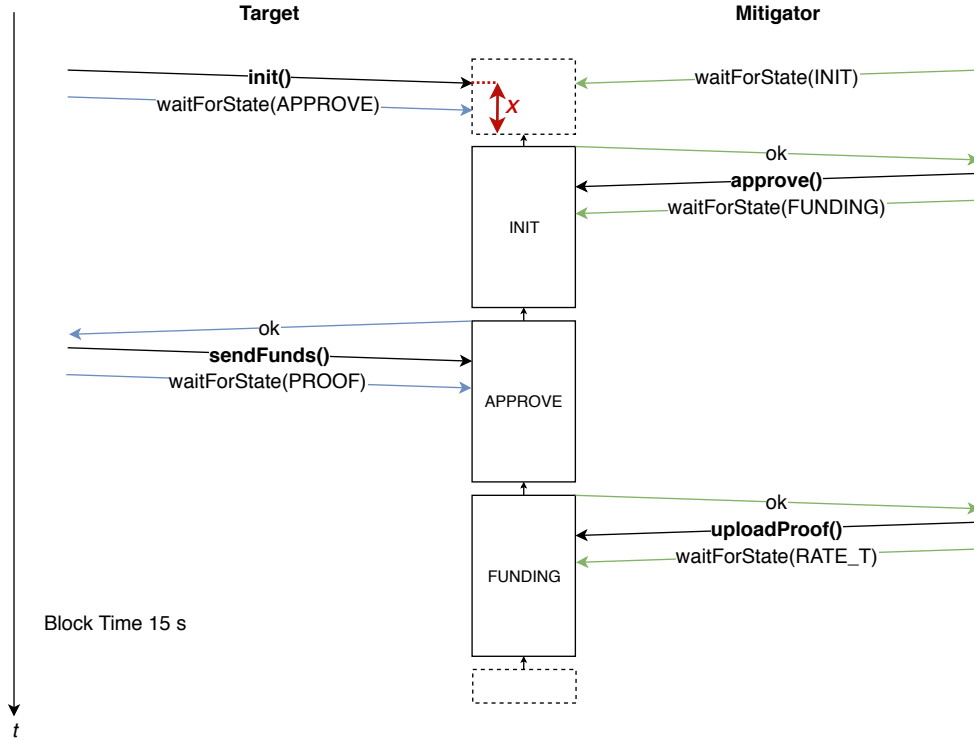


Figure 4.1: Target and Mitigator Geth Scripts

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 timeframe closer to the beginning of the starting block, *i.e.*, maximizing x , the global results become comparable to the results using Ganache and local Rinkeby. Since truffle deploys a new contract each time the truffle test command is run, the local Rinkeby tests start already synchronized with x maximized. Using the private blockchain on Ganache, obviously synchronization and, therefore, x is maximized by default as well. However, if there would be no synchronization of the measured timeframe, random times would be considered for the first block which would mean that on average each timeframe of the first scenario, for instance, would take 7.5 seconds less for completion. A second approach to retrieve the correct time measurement on the global performance tests is to run the tests $n = 20$ times at once such that the first run acts as a synchronization step. It should be pointed out that depending on the scenario and the missed deadlines, full block times represent the worst case in terms of performance of the Cooperative Signaling system.

By running the target script on an the AWS instance in Ohio and a mitigator script on

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

Table 4.8: Global Rinkeby Processing Times [s] for Protocol (Zurich-Ohio)

a node located in Zurich (both synchronized to the Rinkeby network), the average global Rinkeby processing time with $n = 20$ is 96.950 seconds and the average standard deviation is 1.146 seconds (*cf.* Table 4.8).

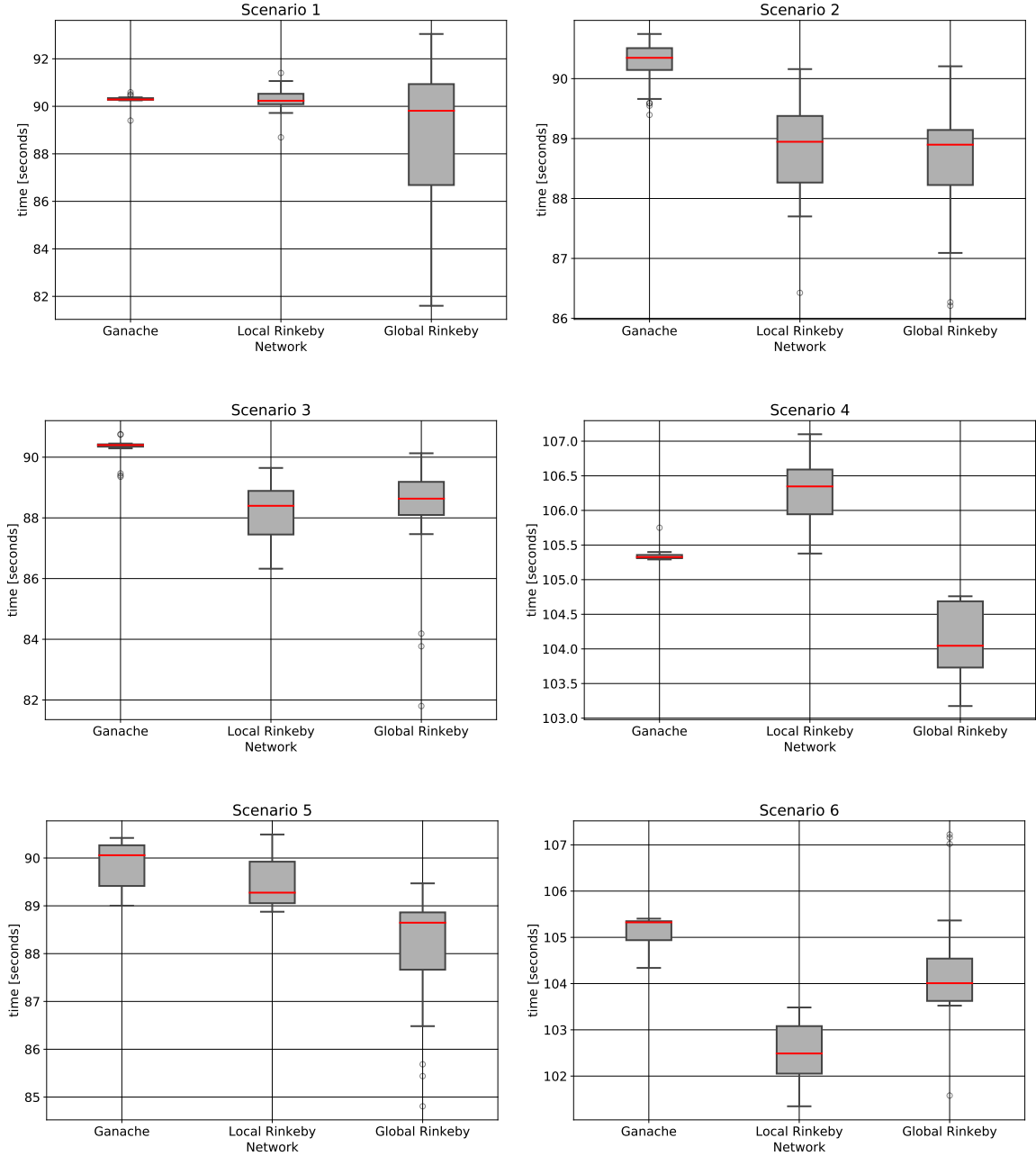
Since the control condition has been tested and evaluated as well, similar results in terms of average processing time and average standard deviation are expected. But although similar results could be reached as shown in the Tables 4.8 and 4.9, the nodes were not synchronized at all times due to timeouts and, therefore, missed deadlines. This is due to the full nodes, that are geographically in close proximity of the AWS instances in Tokyo and São Paulo, not being synchronized at all times [38].

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

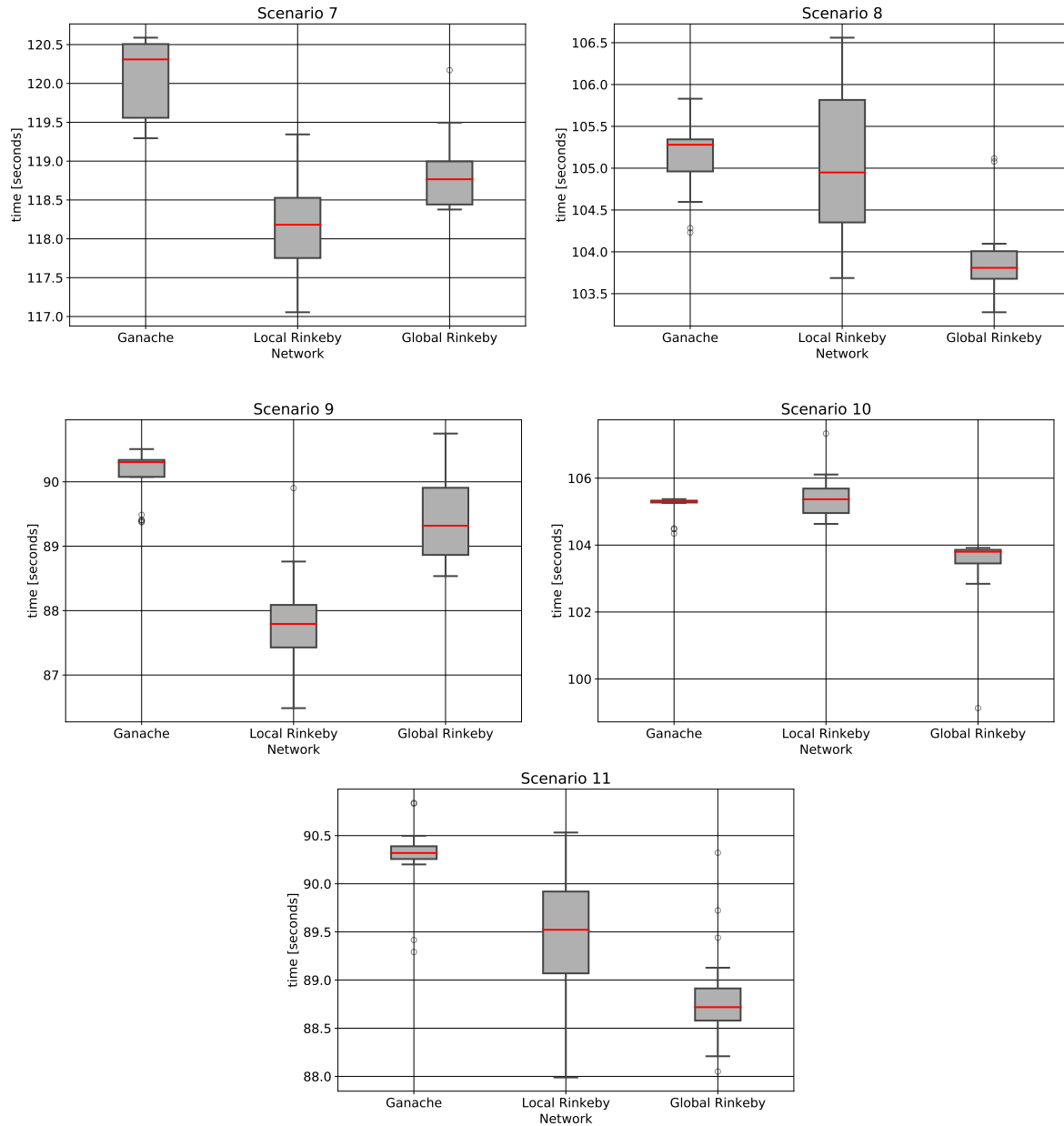
Table 4.9: Control Condition Rinkeby Processing Times [s] for Protocol (Tokyo-São Paulo)

For both global averages, the measured average times are similar with a difference in average processing time of 0.668 seconds which is a difference of approximately 0.7%.

By reaching these quite similar results in both global Rinkeby tests and removing the corresponding RTT shown in Table 4.7 the difference in average processing times for the scenarios is only 0.5171 seconds. It should be noted that 20 test runs per case may not lead to exact average values. Also, every test on Rinkeby was tested with varying (not exact) average block times of 15 seconds.



All listed *performance* results and differences between the standard deviations and median times, based on the used blockchains ganache, local Rinkeby or global Rinkeby, are depicted in the Figures Scenario 1 to Scenario 11. The tests run on Ganache were set to 15 seconds block time and both Rinkeby tests relied on the average 15 second blocktime. Clearly, the variance between each test case and each run across the different infrastructures exists. 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.



Gas used in the global Rinkeby tests showed slight differences compared to Ganache and local Rinkeby, which use the same truffle test scripts to evaluate the *gas used*. While the registration gas usage does not differ in any test, the total Protocol *gas used* on average differentiates 15'366 Gwei compared to local Rinkeby and Ganache. The global Rinkeby gas used results are shown in Table 4.10.

Scenario	Total Gas Used	Target Gas Used	Mitigator 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 4.10: Global Rinkeby Gas Used in Gwei for Protocol

Chapter 5

Summary and Conclusions

The goal of this thesis was to implement a Register and fortify as well as extend the cooperative signaling Protocol. The extension allows searching of a known mitigator in order for the signaling between a target and a mitigator to work seamlessly. During researching and testing the possible use cases, scenarios that have not been recognized in earlier implementations, were finally added as irrational, but still potential, possibilities as depicted in Figure 3.5.

Using the truffle framework to deploy the Protocol and Register on the private Ganache and public Rinkeby blockchain, a test suite on three levels was designed to ensure *correctness*, *security*, and *scalability* of the Signaling Protocol. While tests to ensure *correctness* include feature and necessary balance tests as well as testing for correct state transitions, *security* tests ensure to allow the access to contract functions only to authorized actors. Integration tests were used to test possible as well as restricted state transitions. Finally, the *scalability* tests involved testing the *performance*, measured in seconds for the Protocol to finish a particular scenario and the *gas usage* measured in Gwei. In order to calculate the costs for the registration and each following scenario, private as well as public blockchains were used to compare the results using similar client scripts.

The average processing times in Section 4.2 also showed that under perfect conditions *i.e.*, close to zero block times and no network latency, the average processing time for the Protocol execution takes 0.498 seconds. Test scenarios using the global Rinkeby setting showed, however, that on average across all scenarios, 96.943 seconds should be expected for the 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 as depicted in Figure 3.5. In addition to the main factor of the average block time, the latency between Zurich and Ohio also adds 0.1192 seconds to the total processing time of the 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 4.3 and 4.4, that the results are indeed similar to each other. While the processing tests on the private local blockchain 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.

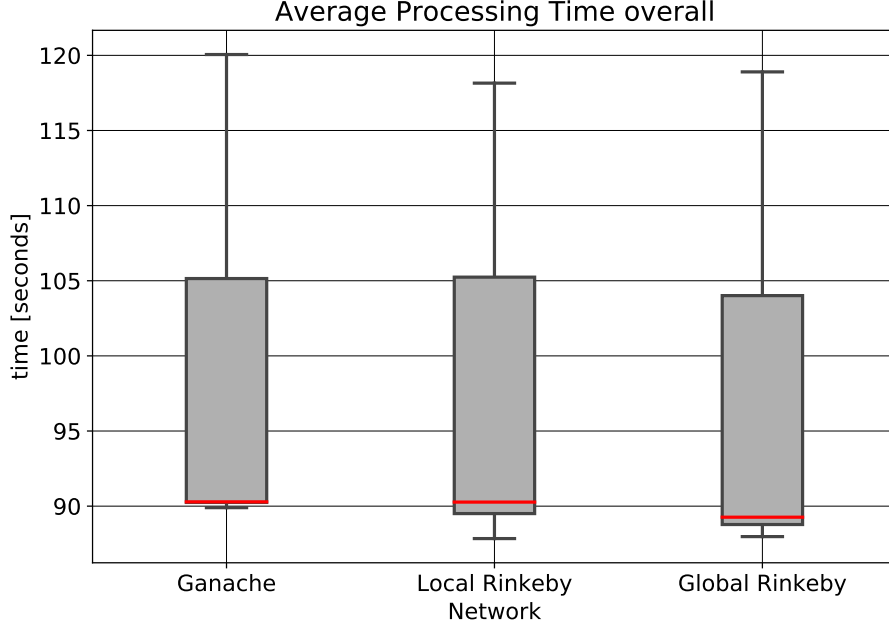


Figure 5.1: Average Times on Ganache, Local and Global Rinkeby

The *performance* results in Figure 5.1 depict the overall average times which summarizes the three infrastructures, eleven scenarios and for each scenario $n = 20$ test runs. In total, 660 performance scenarios were run to evaluate the worst-case *performance* of the signaling process depending on which client and blockchain were used. Overall similar processing times were evaluated, and it could be shown that by using Infura as a proxy, the behavior of the public testnet Rinkeby with a simulated actor in Ohio, could mostly be mimicked. This result implies that the *performance* of the Protocol Smart Contract 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 very similar results when both nodes were synchronized, which gives more confidence in the cooperative signaling.

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

Table 5.1: Total Average Global Rinkeby Costs [\$]

The overall *gas used* to deploy and utilize the Register is the same on all evaluated infrastructures, whereas the Protocol shows slight differences. On average a mitigator

would need to pay the gas costs that range from 0.04 USD, for the gas price of 1 Gwei, to 0.8 USD when prioritizing the transactions with a gas price of 20 Gwei. A target is required to pay more gas on average, which ranges from 0.06 USD to 1.13 USD. But not only does the Target have to pay for the processing of the state transitions, the deployment of the Protocol must be paid as well. The one-time costs for deployment of the Protocol range from 0.82 USD to 16.40 USD and the total costs to deploy both Smart Contracts require a payment from 1.03 USD to 20.51 USD (calculating with 216 USD/Ether).

Future runs in a real DDoS scenario are still to be tested. Depending on which scenario is run, deadlines and the deadline interval may be changed when a DDoS mitigation is executed. When testing the Cooperative Signaling in a real-world scenario, it might not be apparent during initialization of the Protocol how long the mitigation process may take. Therefore, it may be the case that a different mechanism should be implemented to predict how long an actual mitigation takes or the possibility to be extended such that a mitigator that is actively defending, will still be rewarded, instead of missing a deadline. Also, instead of storing raw names and strings in the Register, hashes of the data or even hashes of the storage address could be stored on the blockchain, since transparency should be taken into account. With this, ratings of the mitigator or the target could be extended in the Register in order to enable the possibility of ranking and separating positively rated actors from negatively rated ones as well.

The Cooperative Signaling was extended by a Register Smart Contract, and numerous modifications and extensions were incorporated into the Protocol. The irrational scenarios 5, 8, and 11 were added to the already existing scenarios in order to complete the scenario tree. While *correctness* and *security* tests enabled *scalability*, testing the *performance* and the *gas used*, was used to evaluate the behavior with different blockchains as infrastructures. All gas usages were compared and converted to more comprehensible fiat currency in order to evaluate the feasibility. In general, this prototype has proven to be a working solution for the signaling between a target and a mitigator in all the test scenarios tested. Nevertheless, to prove its usability and real-world usage, it could be tested in a real DDoS mitigation scenario.

Bibliography

- [1] Akamai. *How to Protect Against DDoS Attacks - Stop Denial of Service*. URL: <https://www.akamai.com/us/en/resources/protect-against-ddos-attacks.jsp> (visited on Apr. 12, 2019).
- [2] Manos Antonakakis et al. “Understanding the Mirai Botnet”. In: *26th USENIX Security Symposium (USENIX Security 17)*. 2017, pp. 1093–1110.
- [3] 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*. Vol. 38. 2005.
- [4] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. “A Survey of Attacks on Ethereum Smart Contracts”. In: *IACR Cryptology ePrint Archive* (2016), p. 1007.
- [5] Maria Bada et al. *Improving the Effectiveness of CSIRTs*. 2014.
- [6] Juan Benet. “IPFS-Content Addressed, Versioned, P2P File System”. In: *arXiv preprint arXiv:1407.3561* (2014).
- [7] Thomas Bocek et al. “Modum.io - A Blockchain Start-up”. In: *15th IFIP/IEEE International Symposium on Integrated Network Management (IM 2017)*. 2017, pp. 1–6.
- [8] Dominik Buenzli. *Cooperative Signaling Protocol*. Communication Systems Group, Department of Informatics, Mar. 2019. URL: <https://files.ifi.uzh.ch/CSG/staff/rodrigues/extern/theses/mbm-dominik-buenzli.pdf> (visited on July 25, 2019).
- [9] Vincent Chia et al. “Rethinking Blockchain Security: Position paper”. 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)*. IEEE. 2018, pp. 1273–1280.
- [10] CloudFare. *CloudFlare Advanced DDoS Protection*. 2016. URL: <https://www.cloudflare.com/static/media/pdf/cloudflare-whitepaper-ddos.pdf> (visited on Aug. 23, 2019).
- [11] Coinmarket. *Crypto-Currency Market Capitalizations*. 2016. URL: <https://coinmarketcap.com/currencies/ethereum/> (visited on Aug. 11, 2019).
- [12] Kyle Croman et al. “On Scaling Decentralized Blockchains”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2016, pp. 106–125.
- [13] Stefano De Angelis et al. *PBFT vs Proof-of-Authority: Applying the Cap Theorem to Permissioned Blockchain*. 2018.
- [14] Christian Decker and Roger Wattenhofer. “Information Propagation in the Bitcoin Network”. In: *IEEE P2P 2013 Proceedings*. IEEE. 2013, pp. 1–10.

- [15] Tien Tuan Anh Dinh et al. “Blockbench: A framework for analyzing private blockchains”. In: *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM. 2017, pp. 1085–1100.
- [16] Seyed K Fayaz et al. “Bohatei: Flexible and Elastic DDoS Defense”. In: *24th USENIX Security Symposium (USENIX Security 15)*. 2015, pp. 817–832.
- [17] Arthur Gervais et al. “On the security and performance of proof of work blockchains”. In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. ACM. 2016, pp. 3–16.
- [18] Roman Graf and Ross King. “Neural Network and Blockchain Based Technique for Cyber Threat Intelligence and Situational Awareness”. In: *2018 10th International Conference on Cyber Conflict (CyCon)*. IEEE. 2018, pp. 409–426.
- [19] 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”. In: *Electronic commerce research and applications* 29 (2018), pp. 50–63.
- [20] D Henshel et al. “Trust as a Human Factor in Holistic Cyber Security Risk Assessment”. In: *Procedia Manufacturing* 3 (2015), pp. 1117–1124.
- [21] John Ioannidis and Steven Michael Bellovin. *Implementing PushBack: Router-based Defense Against DDoS Attacks*. 2002.
- [22] 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*. IEEE. 2008, pp. 82–85.
- [23] Jeonghye Kim, Youngseog Yoon, and Hangjung Zo. “Why People Participate in the Sharing Economy: A Social Exchange Perspective.” In: *PACIS*. 2015, p. 76.
- [24] Zhetao Li et al. “Consortium blockchain for secure energy trading in industrial internet of things”. In: *IEEE transactions on industrial informatics* 14.8 (2017), pp. 3690–3700.
- [25] 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”. In: *Electronic Commerce Research and Applications* 9.4 (2010), pp. 346–360.
- [26] S. Mannhart et al. “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)*. 2018, pp. 362–367.
- [27] Stephan Mannhart. “Mitigation as a Service in a Cooperative Network Defense”. MA thesis. University of Zurich, July 2018.
- [28] Jelena Mirkovic, Gregory Prier, and Peter Reiher. “Attacking DDoS at the source”. In: *10th IEEE International Conference on Network Protocols, 2002. Proceedings*. IEEE. 2002, pp. 312–321.
- [29] Andrew Mortensen et al. *Distributed-Denial-of-Service Open Threat Signaling (DOTS) Architecture*. Internet-Draft draft-ietf-dots-architecture-06. Internet Engineering Task Force, Mar. 2018. 32 pp.
- [30] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008.
- [31] Erik Nygren, Ramesh K Sitaraman, and Jennifer Sun. “The Akamai Network: A Platform for High-Performance Internet Applications”. In: *ACM SIGOPS Operating Systems Review* 44.3 (2010), pp. 2–19.

- [32] George Oikonomou et al. “A Framework for a Collaborative DDoS Defense”. In: *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*. IEEE. 2006, pp. 33–42.
- [33] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. “Survey of Network-based Defense Mechanisms Countering the DoS and DDoS Problems”. In: *ACM Computing Surveys (CSUR)* 39.1 (2007), pp. 03–15.
- [34] PolySwarm. *A Decentralized Cyber Threat Intelligence Market*. 2019. URL: <https://polyswarm.io/polyswarm-whitepaper.pdf> (visited on July 23, 2019).
- [35] Ilung Pranata, Geoff Skinner, and Rukshan Athauda. “A Holistic Review on Rust and Reputation Management Systems for Digital Environments”. In: *International Journal of Computer and Information Technology* 1 (2012), pp. 44–53.
- [36] The Associated Press. *Hackers Used 'Internet of Things' Devices to Cause Friday's Massive DDoS Cyberattack*. 2016. URL: <http://www.cbc.ca/news/technology/hackers-ddos-attacks-1.3817392> (visited on Aug. 23, 2019).
- [37] 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)*. 2016.
- [38] Rinkeby. *Rinkeby Testnetwork, Voluntarily Listed Full Nodes*. 2019. URL: <https://www.rinkeby.io/#stats> (visited on July 22, 2019).
- [39] B. Rodrigues et al. “Evaluating a Blockchain-based Cooperative Defense”. In: *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2019)*. 2019, pp. 533–538.
- [40] Bruno Rodrigues, Thomas Bocek, and Burkhard Stiller. “Enabling a Cooperative, Multi-domain DDoS Defense by a Blockchain Signaling System (BloSS)”. In: *Demonstration Track*. 2017, pp. 1–3.
- [41] Bruno Rodrigues, Thomas Bocek, and Burkhard Stiller. “Multi-Domain DDoS Mitigation Based on Blockchains”. In: *IFIP International Conference on Autonomous Infrastructure, Management and Security*. Springer. 2017, pp. 185–190.
- [42] Bruno Rodrigues, Thomas Bocek, and Burkhard Stiller. “The Use of Blockchains: Application-driven Analysis of Applicability”. In: *Advances in Computers*. Vol. 111. 2018, pp. 163–198.
- [43] Bruno Rodrigues and Burkhard Stiller. “Cooperative Signaling of DDoS Attacks in a Blockchain-based Network”. In: *The ACM SIGCOMM 2019 Conference Posters and Demos*. 2019, pp. 39–41.
- [44] Bruno Rodrigues et al. “Blockchain Signaling System (BloSS): Cooperative Signaling of Distributed Denial-of-Service Attacks”. In: *Journal of Network and Systems Management* 28.3 (2020), pp. 1–27.
- [45] Rishikesh Sahay et al. *Towards Autonomic DDoS Mitigation Using Software Defined Networking*. 2015.
- [46] Isabel Skierka et al. “CSIRT Basics for Policy-Makers”. In: *The History, Types & Culture of Computer Security Incident Response Teams* (2015).
- [47] Tai-Won Um, Gyu Myoung Lee, and Jun Kyun Choi. “Strengthening Trust in the Future Social-Cyber-Physical Infrastructure: an ITU-T Perspective”. In: *IEEE Communications Magazine* 54 (2016), pp. 36–42.
- [48] Cynthia Wagner et al. “MISP: The Design and Implementation of a Collaborative Threat Intelligence Sharing Platform”. In: *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security*. ACM. 2016, pp. 49–56.

- [49] Gavin Wood. “Ethereum: A Secure Decentralised Generalised Transaction Ledger”. In: *Ethereum project yellow paper* 151 (2014), pp. 1–32.
- [50] Saman Taghavi Zargar, James Joshi, and David Tipper. “A Survey of Defense Mechanisms against Distributed Denial of Service (DDoS) flooding attacks”. In: *IEEE communications surveys & tutorials* 15.4 (2013), pp. 2046–2069.
- [51] Zibin Zheng et al. “Blockchain challenges and opportunities: A survey”. In: *International Journal of Web and Grid Services* 14.4 (2018), pp. 352–375.

Abbreviations

ABI	Application Binary Interface
AITF	Active Internet Traffic Filtering
AWS	Amazon Web Service
BloSS	Blockchain Signaling System
CIA	Confidentiality, Integrity and Availability
CSIRT	Computer Security Incident Response Team
DDoS	Distributed Denial of Service
DoS	Denial of Service
DNS	Domain Name System
DOTS	Distributed-Denial-of-Service Open Threat Signaling
ETH	Ether (Cryptocurrency)
EVM	Ethereum Virtual machine
IaaS	Infrastructure as a Service
IETF	Internet Engineering Task Force
IoT	Internet of Things
IPFS	Inter Planetary File System
ISP	Internet Service Provider
NFV	Network Function Virtualization
P2P	Peer to Peer
PoA	Proof-of-Authority
PoW	Proof-of-Work
REST	Representational State Transfer
RTT	Round Trip Time
SDN	Software-Defined Networking
SLA	Service Level Agreement
VNF	Virtualized Network Function

Glossary

Authorization Authorization is the decision whether an entity is allowed to perform a particular action or not, e.g. whether a user is allowed to attach to a network or not.

BloSS The Blockchain Signaling System utilizes the Ethereum blockchain as an infrastructure for cooperative DDoS mitigation.

Target A victim that is being attacked by a DDoS attack.

Mitigator An actor that offers mitigation services to a target.

Testnet A testnet simulates the behavior of the public Ethereum blockchain, without costing real Ether.

Rinkeby A testnet using PoA with average block times of 15 seconds.

Infura Relying on a full node of Infura as a proxy to the blockchain relieves a client not having to synchronize the blockchain himself and leads to more centralization.

DDoS Distributed denial of service follows the goal of bundling the distributed resources comprised of packet streams in order to attack a target such that legitimate traffic is not possible anymore [28].

List of Figures

3.1	Influence of block size on the propagation delay [14]	15
3.2	Architecture of the Blockchain Signaling System (BloSS) [27]	19
3.3	Registration Sequence Diagram	21
3.4	State Transition Behavior of the Protocol	22
3.5	Cooperative Signaling Protocol	23
3.6	Cooperative Signaling Test Suite	28
4.1	Target and Mitigator Geth Scripts	37
5.1	Average Times on Ganache, Local and Global Rinkeby	44

List of Tables

3.1	Benefits and Drawbacks of a Blockchain-based collaborative platform in the cybersecurity context	14
4.1	Ganache Processing Times [s] for Register	32
4.2	Ganache Gas Used in Gwei for Register	33
4.3	Ganache Processing Times [s] for Protocol	33
4.4	Ganache Gas Used in Gwei for Protocol	34
4.5	Local Rinkeby Processing Times [s] for Register SC	35
4.6	Local Rinkeby Processing Times [s] for Protocol SC	36
4.7	Round Trip Time [s] between synchronized Nodes in Zurich, São Paulo, Tokyo and Ohio	36
4.8	Global Rinkeby Processing Times [s] for Protocol (Zurich-Ohio)	38
4.9	Control Condition Rinkeby Processing Times [s] for Protocol (Tokyo-São Paulo)	38
4.10	Global Rinkeby Gas Used in Gwei for Protocol	41
5.1	Total Average Global Rinkeby Costs [\$]	44

Listings

3.1	Protocol Extension to get a Specific Mitigator from the Register	25
3.2	Call Executes Low Level Function of the Callee Contract	25
3.3	Abstract Call Register Function from Protocol to Get a Specific Mitigator	26
3.4	Register Smart Contract	26
3.5	Localhost Addition to Networks in truffle-config.js	29

Appendix A

Contents of the CD

The enclosed CD contains the following content:

Code

Both Smart Contracts as well as a migration file and the truffle folder structure that is needed for running the tests is included in this folder.

Documentation

This folder contains the LaTeX source code of this document as well as all used Figures and full Tables for the evaluation.

Literature

This folder contains all referenced papers.

Presentation

The Powerpoint slideset for the final presentation is included in this folder.