



University of
Zurich^{UZH}

A Reputation and Reward Scheme for a Cooperative, Multi-domain DDoS Defense

*Andreas Gruhler
Zurich, Switzerland
Student ID: 12-708-038*

Supervisor: Bruno B. Rodrigues
Date of Submission: March 19, 2018

Abstract

Distributed Denial-of-Service (DDoS) Attacken nehmen an Häufigkeit, Vielfalt und Intensität zu [14]. Die Hard- und Software-Konfiguration von bestehenden, zentralisierten Verteidigungsansätzen ist jedoch oft unzureichend [24]. Ein kooperatives, domänenübergreifendes DDoS-Mitigationssystem bietet Verteidigungsdienste auf einer bestehenden, dezentralen Infrastruktur an [9]. Den Teilnehmern dieser Systeme fehlt es jedoch an Anreizen für eine erfolgreiche Zusammenarbeit. Diese Masterarbeit entwirft, implementiert und evaluiert ein Reputations- und Belohnungsschema für das Blockchain-Signalisierungssystem (BloSS). Reputations- und Belohnungsprotokolle schaffen Anreizstrukturen, welche für die Zusammenarbeit zwischen Dienstleistern und Verbrauchern nötig sind. Die Blockchain ermöglicht nicht nur eine effiziente Signalisierung von Angriffen, sondern ist auch nützlich für das Reputationsmanagement [37, 54]. Deshalb schlägt diese Masterarbeit einen Prozess vor, welcher mit Smart Contracts umgesetzt wird, um die Reputations- und Belohnungszahlungen zu automatisieren. Das binäre Reputationssystem wurde durch Simulation verschiedener Kundenstrategien getestet und evaluiert. Diese Arbeit kommt zum Schluss, dass unter anderem Beta-Reputation den Kunden die notwendigen Informationen liefern kann, um faule und selbstsüchtige Teilnehmer im Voraus zu erkennen. Nichtsdestotrotz kann ein Reputationssystem nicht alle Mängel in der kooperativen Abwehr von DDoS-Angriffen beheben. Das Reputationssystem benötigt Instanzen, welche dazu befugt sind, die Identitäten der Kunden zu überwachen, um betrügerische Absprachen zu verhindern. Ausserdem ist eine Drittpartei notwendig, um potenziell böswillige Kundschaft aufzuspüren und vom System auszuschliessen, indem Reputations- und Belohnungskonflikte unparteiisch geschlichtet werden. Schlussendlich wurde in der Arbeit auch ein inhärenter Interessenkonflikt zwischen einer nachweisbaren Reputations-Engine und vollständiger Kundenanonymität identifiziert.

Distributed Denial-of-Service (DDoS) attacks rise in frequency, diversity and intensity [14]. Often, centralized defense approaches lack hard- and software capabilities [24]. A cooperative, multi-domain DDoS mitigation system provides defense services on top of an existing, distributed infrastructure [9]. However, the participants in these systems lack incentives for cooperation. This thesis designs, implements and evaluates a reputation and reward scheme for the Blockchain Signaling System (BloSS). Reputation and reward systems provide the necessary incentives for cooperation among service providers and consumers. Blockchain does not only allow for efficient attack signaling, but is useful to manage reputation as well [37, 54]. Thus, the thesis promotes a process implemented with smart contracts, in order to automate reputation and reward payments. The binary reputation system was tested and evaluated by simulating different customer strategies. Among other metrics, Beta reputation provides the necessary information to detect lazy and selfish participants in advance. Nonetheless, a reputation system cannot solve all deficiencies in cooperative defense. The reputation system is in need of authorities overseeing identities to prevent collusion. Moreover, a third party locating and expelling potentially malicious customers by resolving reputation and reward disputes is indispensable. Last but not least, there exists an inherent conflict of interests between a verifiable reputation engine and full customer anonymity.

Acknowledgments

I want to thank my supervisors Bruno Rodrigues and Dr. Thomas Bocek for their guidance and assistance during the thesis. I also want to thank my fellow student Lukas Eisenring, for providing me with his IP address generation snippet for use in the simulator script. I would like to thank Benedikt Bleyer for repeatedly proofreading my thesis. Thanks goes to all these people, and Mervin as well, for always having time to discuss about the issues at hand. I thank the ICU fellows for participating in relaxing garage skateboarding sessions. Last but not least, I would like to give thanks to my family and friends for their emotional support.

Contents

Abstract	i
Acknowledgments	iii
1 Introduction	1
1.1 Thesis Goals	3
1.2 Terminology	4
1.3 Thesis Outline	4
2 Background and Concepts	6
2.1 Concepts for Reputation Propagation	8
2.1.1 Tokens	8
2.1.2 Event Reputation Factors	9
2.1.3 Thresholds	9
2.1.4 Multi Signature Transaction (Multi-Sig)	10
2.1.5 Optimistic Fair Exchange	11
2.1.6 Anonymous Feedback	12
2.1.7 Incentive to Vote	13
2.1.8 Insurance Models	14
2.2 Reputation Engines	15
2.2.1 Probabilistic Engines	15
2.2.2 Fuzzy Engines	16
2.2.3 Graph and Flow Engines	16

2.3	Reaction and Service Differentiation	18
2.4	System Properties	18
2.5	Agents and Identity Management	19
2.6	Data Storage and Exchange	20
2.7	Evaluation Frameworks and Metrics	21
3	Design	23
3.1	Incentive Design for Cooperative Defense	23
3.2	System Architecture	27
3.3	Reputation Exchange, Computation and Storage	29
3.4	Reputation Metrics	29
3.5	Metric Verification Process	30
4	Implementation	31
4.1	Construction of the Mitigation Task Index	31
4.2	Mitigation Tasks	32
4.3	Mitigation Protocol	33
4.4	Storage of Reputation Data	35
4.5	Reputation Data Format	36
4.6	Reputation Metrics	37
4.7	Metric Verification Process	38
4.8	Methodology	41
5	Evaluation	42
5.1	Simulation Setup	42
5.2	Evaluation Metrics	43
5.3	Customer Profiles and Strategies	44
5.4	Customer Behavior in Test Mode	44
5.5	Customer Reputation in Acceptance Mode	47

6 Discussion	50
6.1 Evaluation Results	50
6.2 Robustness against Rating Fraud	51
6.2.1 Sybil- and Collusion Attacks	51
6.2.2 Ballot Stuffing and Bad-mouthing	51
6.3 Verifiability and Anonymity	52
6.4 Limitations	52
7 Final Considerations	54
7.1 Future Work	55
Bibliography	55
Abbreviations	60
Glossary	61
List of Figures	62
List of Tables	64
A Implementation	66
A.1 Documentation of the Reputation API	67
A.2 Test Coverage and Continuous Integration	68
B Evaluation Results	69
C Contents of the CD	76

Chapter 1

Introduction

Distributed Denial-of-Service (DDoS) attacks are a major threat to the Internet availability that remains unmitigated despite many commercial and research efforts. A large number of unsecured devices that range from connected cameras to smart fridges are being connected to the Internet and their growing processing capacity allows attackers to take control of a vast amount resources to launch malicious attacks. Many of these devices are insecure by design and not often are impossible to be secured due to their hardware and software constraints.

To prevent or reduce damages caused by these DDoS attacks, different detection and mitigation methods are available, being mainly organized in a single-domain defense or, in-house. These defense methods are mainly implemented based on dedicated ASIC-based (Application Specific Integrated Circuit) appliances to analyze flow records exported from edge routers, and further filtering or load balancing traffic. Similarly, cloud services such as the ones offered by CloudFlare [14] or Akamai [1] can take away the burden of detection and mitigation, serving as a proxy able to load balance, reroute, or drop the traffic in case of DDoS attacks.

However, in-house defense systems have shown to lack hard- and software capabilities to detect and mitigate the attacks themselves [24]. These trend attacks can easily exceed the defense capacity of a single system in the aspects of volume and frequency. Thus, as DDoS attacks become progressively sophisticated and coordinated, the defense from such attacks likewise needs distribution and coordination. By utilizing other organization's resources, the burden of the protection can be shared, and defense capabilities can be extended through the different protection systems participating in the distributed defense. The Internet Engineering Task Force (IETF) drafted a Distributed-Denial-of-Service Open Threat Signaling (DOTS) architecture for this purpose [24].

While providing many benefits, a cooperative defense also poses many challenges. For example, why such organizations should help each other. In a competitive environment, trust needs to be established. Solely relying on voluntary contribution (*i.e.*, accepting defense requests) creates a favorable environment for free riding peers (consuming resources without contributing). This situation, and the social dilemma that the business partners

find themselves in is illustrated in Figure 1.1. First, the attack target publishes malicious IP addresses. Second, multiple mitigators adjust the configuration of their network devices to filter and drop the malicious packets. This second step is referred to as the actual “mitigation service” in this thesis. In a third step, the attack target evaluates the effectiveness of the mitigation service.

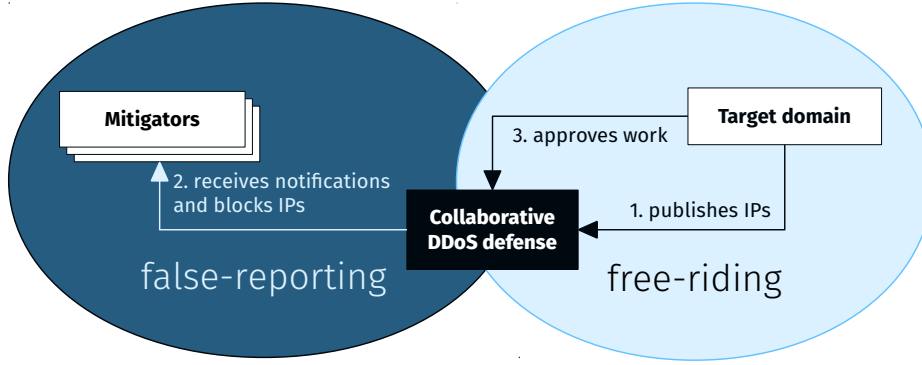


Figure 1.1: The social dilemma of false-reporting and free-riding exists in crowdsourcing applications as well as in collaborative DDoS defense (own illustration, based on [9, 53]).

A reputation scheme allows contributors and consumers of the network to rate entities that request protection in a cooperative defense. These systems have already been proven useful for e-commerce websites to incentivize peers to contribute with relevant information and establish fairness among peers. Moreover, similar social dilemmas exist in other research areas, *e.g.* crowdsourcing [53]. Even in large Peer-to-Peer (P2P) networks, peers maintain lasting business relations and transact repeatedly [48]. This increases the potential benefit of such type of systems in P2P related domains. In mobile ad hoc networks (MANET), researchers also identified the same need, to provide incentive and credit-based mechanisms for cooperation among peers [17, 27]. Thus, incentive or reward mechanisms are required. The DOTS technical draft does not touch on these issues, which are of administrative nature or part of business agreements between the cooperating domains.

The design of reputation and reward systems is still a challenge, because they are vulnerable to gaming attempts by malicious users [10, 18]. These attacks against reputation systems are also referred to as “rating fraud” and can occur in different forms [11]. Attackers can use these strategies to inflate their own reputation (*i.e.* ballot-stuffing) or cause the defamation of others (*i.e.* bad-mouthing) [11]. In collusion and Sybil attacks, multiple identities are misused to manipulate reputation [18]. When using multi-dimensional and subjective user-ratings, an unfair rating can dilute the reputation of other peers, independent of their actual behavior [18]. Therefore, self-balancing and credible reputation metrics based on well-defined inputs other than subjective ratings are needed [50]. Similarly, a self-policing reputation systems can discipline the participants behavior without the need for a central authority [26]. Another problem identified in [18] is, that reputation systems are prone to re-entry attacks if the cost to obtain a new identity is too low. A

carefully chosen initial reputation score and reputation function can present newcomers trustworthy in their first transactions, but still discourage from whitewashing the identity [8]. The cost to create a new identity must be higher than the benefits that come with the initial reputation score. The blockchain-based attack signaling architecture proposed in [9] already leverages IP verification to proof IP ownership. Using the scarce IPV4 address space has shown to be a good way to raise the cost barrier of re-entry and to decrease the risk of Sybil and whitewashing attacks [11, 18].

Blockchain technology not only offers new possibilities in attack signaling, but also emerges as a trust-worthy and distributed solution for reputation management [9, 54]. Since reputation is earned in interactions between peers, it can be modeled or even attached to transactions. The blockchain could then be used to build consensus about the reputation of all peers in the system. Decentralized ledger technologies such as the New Economy Movement (NEM [37]) calculate importance scores for peers. According to NEM (2015), “importance cannot be arbitrarily manipulated or gamed” and “importance scores are useful for purposes other than just block chain consensus. They can be interpreted as a form of reputation” (p. 35).

1.1 Thesis Goals

The goal of this thesis is the design and development of a reputation and reward mechanism applied to a cooperative, multi-domain DDoS defense. Therefore, the thesis extends existing functionality proposed in [9] as well as develops requirements imposed by such mechanisms, which are essential capabilities toward the development of the reputation and reward functionality. However, as the development requires knowledge of the existing framework, more detailed requirements and goals of the aforementioned tool were discussed during the meetings with the supervisors. Driven by the description of work outlined, the following stages are defined for this master thesis in order to achieve the main goal:

1. **Problem analysis and planning:** This step requires understanding the problem and possible requirements further analysing existing solutions. Thus, it is required to search main related works listed in state-of-the-art, discussing and evaluating if these works (or parts of their proposed solutions) can be used in the design phase of the proposed solution.
2. **Requirements and System Design:** This stage also comprises, as an initial stage, the formalisation or proposal of requirements establishing parameters for the operation and evaluation of the proposed system. At this stage it is necessary to formalise an architecture capable of achieving the main objective.
3. **Produce a working prototype:** This stage consists on the integration of the designed and implemented elements into the whole system, producing results and material to be analysed and contrasted with the previously specified requirements. Source code needs to be open source, well documented, and readable. The prototype needs to be tested with appropriate testing practices.

4. **Write a well-structured report** that includes motivation, background information, related work, design decisions, implementation details, deep evaluation (also in comparison to related work), and conclusions. Two reports are required, one mid-term report describing current progress and the final report.

1.2 Terminology

Reputation and reward have no universal definition [48]. Nevertheless, we use a standardized terminology to refer to objects of investigation. Because the terms and definitions from the DOTS architecture draft are close to the research area of DDoS attacks and already provide a basic set of terms, this thesis leans on that terminology as much as possible and only extends it where necessary [36]. For the purpose of this thesis, the terms “mitigation contract” and “mitigation service” are used synonymously with a “mitigation task”, short “task”.

Fairness is a term encountered in protocols for fair exchange [32]. An exchange protocol is considered fair, when in case of an abortive transaction either both parties have what they want or neither party has an advantage compared to the status quo. A fair exchange is a foundation to grow trust in almost all long-term business relationships.

1.3 Thesis Outline

Chapter 2 surveys concepts leveraged in current reputation systems and related research. The topics discussed in this chapter are the result of a literature review.

Chapter 3 presents the design of a reputation and reward system for cooperative DDoS defense. More precisely, this chapter develops a process for rating and rewarding the contributors in a DDoS mitigation scenario. The process was designed with the properties of a fair exchange protocol in mind [32]. Further, this chapter integrates knowledge obtained through the literature review to embed this central process into a holistic reputation system.

Chapter 4 documents the experimental test setup and development process for the reputation system prototype. This chapter also explains the software architecture and central parts of the program code.

Chapter 5 develops an evaluation method using a software simulator. The reputation measurements aim answering the question, if reputation values of the developed system can lead to better decisions in the process of cooperative defense. The discussion of these measurement results is held in **Chapter 6**.

Finally, **Chapter 7** reflects the current state of this work and makes final considerations about future directions.

Appendix A contains a high-level description of the reputation API endpoints used for reputation aggregation and summarization and smart contract test coverage results.

Appendix B holds the detailed reputation results of the evaluation.

Chapter 2

Background and Concepts

This chapter emerged from a literature review about reputation and reward schemes in P2P and blockchain networks, in order to gather information about how to design such as system for cooperative DDoS defense. A brief overview about the structure of this review is given in Table 2.1. The objects of investigation were selected by analyzing the most prevalent topics in literature. Google Scholar served as the main source of information. Among others, the queries included the following main keywords: “Blockchain”, “Reputation”, “System”, “Engine”, “Propagation”.

Reputation systems are deployed in different business areas and application contexts. There exists a multitude of use cases for reputation and incentive mechanisms: Crowdsourcing and -sensing [16, 53], MANETs [17], Border Gateway Protocol (BGP) routing [51], e-commerce [43], file-sharing [48], mobile data plan sharing [52], prediction markets [40], various P2P and blockchain networks and cooperative DDoS defense [9, 41]. These use cases will be referenced wherever appropriate during this chapter, to explain a certain concept or mechanism.

Reputation and incentive-based mechanisms have been widely explored for P2P systems [8]. However, the blockchain technology introduces a novel way to decentralize trust creating a certification infrastructure that records all traces of transactions, allowing users to look up and record histories of transaction outcomes. By relying on the public ledger to form reputation scores and reward contributors, reputation and rewards of users are tightly coupled to their actual network activity. The process of reputation aggregation, propagation and metric calculation is presented in Sections 2.1 and 2.2. These sections discuss several reputation propagation mechanisms and reputation engines because these are two major building blocks for reputation systems [5]. Furthermore, blockchain provides the underlying structure towards a financial reward scheme through cryptocurrency. This currency represents an incentive to behave reputable, if actions are rewarded accordingly. Financial rewards and reputation can be treated independently or combined. The possibilities for such system reactions (*e.g.* reputation adaptive pricing) based on users reputation score are discussed in Section 2.3. The inherent properties of blockchain allow to secure reputation systems against weaknesses mentioned in the introduction [18]. Section 2.4 highlights possible gaming attempts on reputation systems, their proposed

solutions and additional robustness properties. Also, storage, calculation and communication of reputation can be formalized in smart contracts. These issues are addressed in Section 2.6. Because reputation is always bound to an identity, Section 2.5 explores how identity management is related to reputation systems [10]. Lastly, this thesis also aspires to integrate with current research. Thus, evaluation frameworks and metrics are discussed in Section 2.7. This last section shows established methods to evaluate and benchmark a reputation system.

Object of investigation	Description
Reputation Propagation (Section 2.1)	Reputation propagation, also known as reputation management [8], can be centralized or decentralized [5]. The focus of this thesis lies on the decentralized design concepts used when building reputation systems for blockchain applications or P2P networks more generally.
Reputation Engine (Section 2.2)	The engine defines how reputation scores or metrics are calculated based on the reputation input [5]. The input can originate from direct user feedback, contributions or from transactions if applied in a blockchain context.
Reaction and Service Differentiation (Section 2.3)	Incentives and rewards can be bound to the current reputation of a participant. Since a reputation metric is always bound to a specific context, it is disputed how far services in different contexts can be differentiated based on the same reputation score. Carelessly designed relationships between reputation and reward can lead to vicious cycles [20].
System Properties (Section 2.4)	The required properties of a reputation system are self-policing, a foundation of trust, strong incentives, robustness against a variety of attacks and an accurate and verifiable scoring system [26, 51]. In some applications (<i>e.g.</i> e-commerce or mobile crowdsensing) there is also a high focus on privacy or anonymity [16, 26, 43, 46].
Agents and Identity Management (Section 2.5)	Different types of agents have different competing (economic) interests in the system. An agent reputation is inherently bound to its identity [10]. Pseudonyms can serve as a proxy to guarantee a certain degree of anonymity if required by the application context [16].
Data Storage and Exchange (Section 2.6)	There are few standards when it comes to reputation exchange. The IETF proposed “reputon” as the standard media type for reputation exchange [30]. Also, the Incident Object Description Exchange Format (IODEF) is used to transport reputation scores between domains in Software-Defined Networks (SDN) [22].
Evaluation Frameworks and Metrics (Section 2.7)	Since reputation is context dependant and multi-faceted it poses a challenge to compare and evaluate the different models. Schlosser <i>et al.</i> propose a simulation framework to compare different reputation systems [44].

Table 2.1: Literature overview about reputation and incentive system design

2.1 Concepts for Reputation Propagation

2.1.1 Tokens

Electronic tokens are a common way to represent reputation. Liu *et al.* propose an identity and reputation management system on the Ethereum blockchain [33]. A reputation coin termed RpCoin can be earned by completing tasks, which is like reputation itself, a non-transferable coin. The authors define two different types of tasks:

1. The reputation task contains a reputation claim beneficial to the publisher. All task participants cast a binary vote. Votes corresponding to the final result of the task will increase the reputation of the voter. The reputation of the task publisher is only increased if the result of the voting is positive.
2. The incentive task is used to discipline the peers. Without these tasks, the system is vulnerable to ballot stuffing and speculation of voting peers about final results. Therefore, these incentive tasks contain a negative claim about another peer in the system. The voting rules are similar to the reputation task but voters are rewarded for detecting fraudulent peers.

Reputation and reward are omnipresent in academia, citations and degrees can be interpreted as indicators for relevance. Sharples and Domingue analyzed the potential of blockchain for educational record management [45]. In the educational economy, credits and degrees are stored in the blockchain. This economy is fueled by Kudos, an educational reputation currency. In comparison to RpCoins, Kudos is envisioned to be traded according to rules enforced by smart contracts. Concrete projects to realize such a system have already been undertaken in the Ethereum community.

Further from an academic point of view, the Work.nation project is currently designing a similar infrastructure for decentralized portfolio management and skill attestation for working professionals [49]. Contributions of work during projects are verified by team members and the transparent system allows for rapid team building based on attested skills. The proof of concept is built using the Ethereum blockchain.

As of another example targeting the decentralized prediction market, Augur is used to weight the reports about real world events [40]. Much of the theory behind Augur was developed in Sztorcs work on Truthcoin [39]. A prediction market in Truthcoin is called “oracle corporation” and consists of a customer and an employee layer with different currencies. Reputation is represented by VoteCoins (VTC) and is the currency transacted on the employee level. Another coin (called CashCoin) on the customer level is solely used to buy/sell shares of an expected outcome. The purpose of prediction markets is to forecast outcomes of real world events based on share prices. The market price of these shares reflect the expected probability of the outcome. VTCs are tradeable between the employees of the oracle corporation, but the total supply of reputation tokens in the system is fixed. VTCs are gained when a report about a real-world event is consistent with the consensus and lost otherwise. Only reputation owners can create reports about

such events. VTCs are also withdrawn if the user does no longer participate in event-reporting. In such a case, the voter can maximize its utility by selling the VTCs, because they are liability as much as asset.

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

2.1.2 Event Reputation Factors

In the trust module proposed by Moinet *et al.*, blockchain payloads trigger particular events [35]. The module is applied to establish trust in decentralized sensor networks. Every event is associated with a positive or a negative reputation factor, depending on the nature of the event from the originating transaction. Then the cumulative reputation score can be calculated by stepping through all past events of all blocks and process the associated reputation factors. Moinet *et al.* use an exponentially decaying function to present recent events more strongly (reputation decay). With this approach, the required minimum reputation of an agent to execute a specific action can be calculated based on the reputation factors. These minimum requirements can be designed trust defaultive for small sized networks and scale well with network size. A practical benefit of using events for reputation aggregation is that common blockchains and smart contract languages have built in support for event management (*e.g.* events in Ethereum or Hyperledger Fabric chaincode event listeners).

2.1.3 Thresholds

In mobile crowdsensing, workers smartphone sensors are used to aggregate knowledge. The reputation can indicate reported measurement data quality and help to validate data [16]. Zhang and van der Schaar propose a threshold based incentive protocol as seen in Figure 2.1. This system is designed to be robust against “false-reporting” and “free-riding”.

The incentive protocol works like a state machine. Assume a newly activated user starts with a reputation θ at the social norm threshold h_k . At the end of each time period the requester evaluates the action (a) of the user. If the service requester is satisfied with the action, the evaluation will result in $a = H$ and the user is given one additional unit of reputation ($\theta = h_k + 1$). If the task is not solved at the end of the time period, then the service requester rates $a = L$ and one unit of reputation is deducted from the user. We can assume ex-ante payment and thus a truth-telling requester.

As long as the users reputation (θ) is in the active region above the social norm threshold ($\theta \geq h_k$), the user is allowed to work and actions will be evaluated by the requester. In this active region the worker need to be in compliance with the social norm. Still, whenever

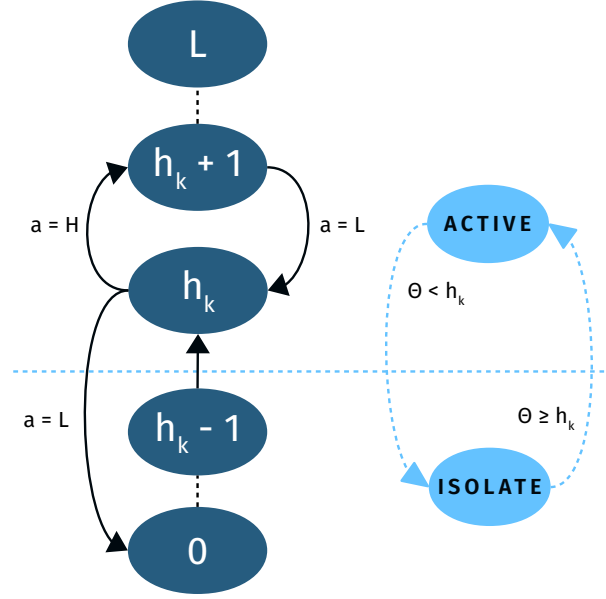


Figure 2.1: Threshold based reputation model for crowdsourcing applications (illustration based on [53])

$\theta = h_k$ and $a = L$, the user will be forbidden to solve tasks and its reputation will be reset. An isolated user receives one unit of reputation each time period and automatically builds up reputation over h_k time periods. Then, the user is reactivated and its actions are evaluated. The users reputation is again adjusted by the requester from this point onwards.

2.1.4 Multi Signature Transaction (Multi-Sig)

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

A smart contract protected by a multi-sig schema presents another solution to mitigate the same problem (see Figure 2.2). Considering that a service provider (P) and requester (R) both pay a deposit (D_R and D_P). Additionally, the requester has to pay the service price to the service provider. Then, funds D_R and D_P are protected with the Multi-sig, which means that they can only be spent if all parties agree. Kopp *et al.* call this concept Mutual Assured Destruction (MAD) transaction and use it to build a decentralized file storage with financial incentives [29]. As both parties store a safety deposit in the contract, they both have the incentive to resolve the transaction and retrieve the locked funds. Again, under the assumption of rational agents who want to retrieve the deposit, this contractual

agreement can reduce counterparty risks. After the successful termination of the contract, the service provider receives the price (p) for the service and both parties retrieve their deposits.

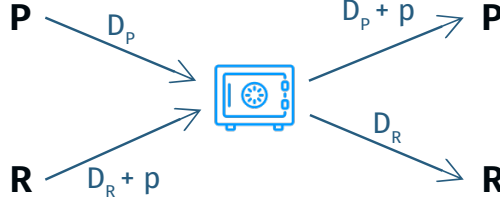


Figure 2.2: MAD transaction: The price (p) stays locked inside the contract until user (U) and provider (P) agree to payout (illustration based on [29]).

2.1.5 Optimistic Fair Exchange

The order of service delivery, reward payment and change in reputation need to be clearly defined. Otherwise, misbehaving peers could delay delivery or reward payments. Liu *et al.* propose a blockchain based payment-for-receipt protocol to design a fair exchange between producer and consumer [32]. Such a design has timeliness and fairness requirements. Timeliness means that either party should be able to abort the protocol any time (strong-timeliness). If that happens, a fair protocol requires the exchange of service and reputation/reward to be completed, or both parties return to the same state before the service request (strong-fairness). Deadlines and predefined timeouts can serve as means to achieve timeliness. However, configuring a short timeout (*e.g.* a few milliseconds) can render the exchange impossible and thus ineffective. Also, the blockchain needs some time to include the transactions in the next block which can delay the protocol and introduce time lags. Furthermore, the strong-timeliness is not guaranteed at all times because the parties may wait until the deadlines to proceed in the exchange.

The weaknesses of timeouts and deadlines in fair exchange can be mitigated with an Optimistic Fair Exchange (OFE) protocol [32]. This protocol usually depends on a Trusted Third Party (TTP) to act as mediator between the two parties. This TTP knows the key to items i_A and i_B of both parties, Alice (A) and Bob (B) participating in the fair exchange. Such a basic scenario is presented in [32] and illustrated in Figure 2.3.

In this OFE, first Alice sends Bob a verifiable encryption of her item (c_A) and her expectations about Bob's item (e_A). Bob verifies c_A in the second step. In the third step, both of them are free to abort the protocol. If Bob sends c_B to Alice in the fourth step, she can verify the item i_B in the fifth step. Afterwards, both parties are free to abort or resolve the protocol. It is important to notice that, the TTP not only knows about the items, but has to maintain state as well. Once one party decides resolve or abort, the other party cannot choose otherwise. In the scenario depicted in Figure 2.3, Bob is able to resolve in step seven and sends his item i_B to the TPP. The TPP decrypts c_A and safely stores both items as long as not both parties resolved. When Alice resolves and sends i_A

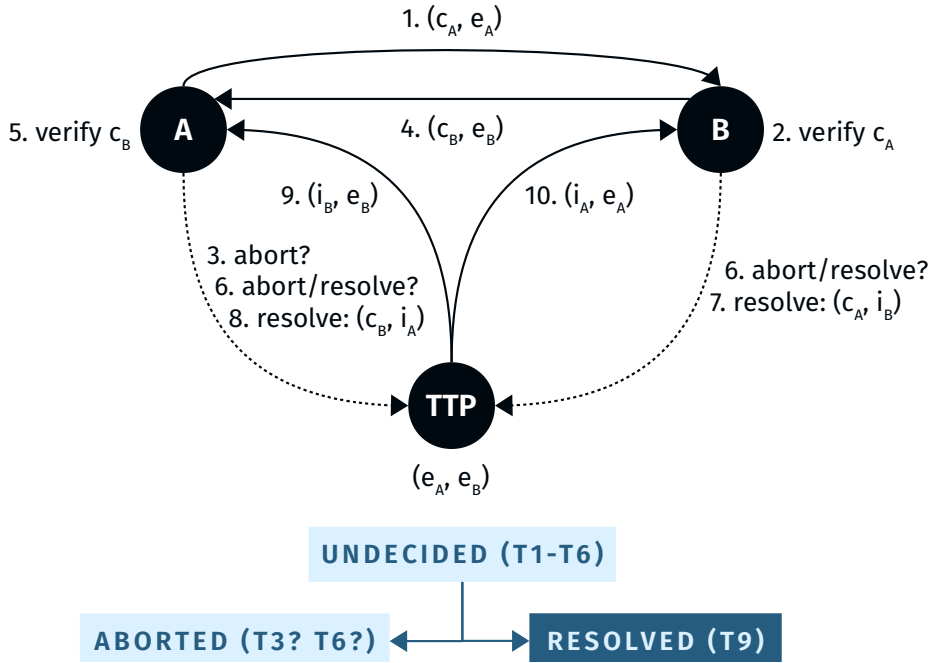


Figure 2.3: A basic optimistic fair exchange scenario with TTP [32]

in the eight step, the TTP can verify and compare both items with the expectations and hand out both items to the respective new owners.

The blockchain allows for OFE without TTP. The exchange can be automated with a smart contract similar to a MAD transaction (see Figure 2.2) with the difference that the provider (P) also needs to include the message expected by the user (U). This message can be a signature of a past transaction, namely a receipt. As the signed receipt which confirms service delivery needs to be fully confirmed on the blockchain first, the provider may experience a small delay before receiving the automated payout from the smart contract.

2.1.6 Anonymous Feedback

Some applications require the reputation system to be privacy preserving [43]. In e-commerce or anonymous marketplaces [46], it might pose an issue that the producer knows which consumer gave a particular bad rating. Also, anonymous feedback mechanisms allow for honest feedback and remove the bias towards positive ratings [25]. An anonymous reputation system is not a contradiction since reputation can also be bound to temporary pseudonyms [16]. Schaub *et al.* designed a blockchain-based trustless reputation system which preserves the privacy and anonymity of the party giving the rating. This is achieved with blind-signatures on key pairs created for each transaction. In the end, the service provider does not know from whom he received the rating. This process is illustrated in Figure 2.4.

First the consumer (c) prepares a public/private key pair for the transaction. Before the transaction happens, the consumer requires a blind signature on the public key for

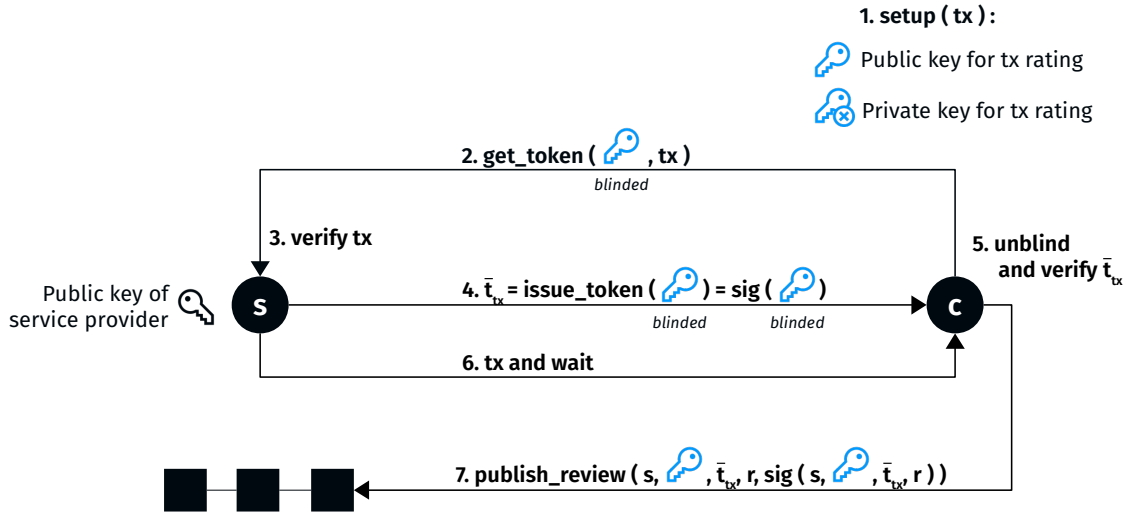


Figure 2.4: Anonymous feedback with blind signatures [43]

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

Limited token issuance for s can be an effective method to fight against ballot stuffing [43], because according to Cai and Zhu (2016) in [11], “this creates a tradeoff between rating and profit for the seller” s . For every token, s can either choose to use it for a real transaction or pay the opportunity cost and use it to inflate its own reputation [43]. The opportunity cost equals to the unrealized profit from a legit sale of goods or services. Even though a limited supply of tokens can incentivize s to use the tokens for genuine purposes, s can never be completely discouraged from “buying” its own reputation [43].

2.1.7 Incentive to Vote

If the feedback in a reputation system is not mandatory, it might be required to provide the rater an incentive to leave feedback. Otherwise, peers might rather not give feedback. Carbonis feedback-based reputation system on top of the bitcoin blockchain incentivizes the rater with a voucher to give feedback for a transaction [12]. Figure 2.5 depicts the situation.

The voucher is linked to the past payment (P) and contains a vote fee and an incentive.

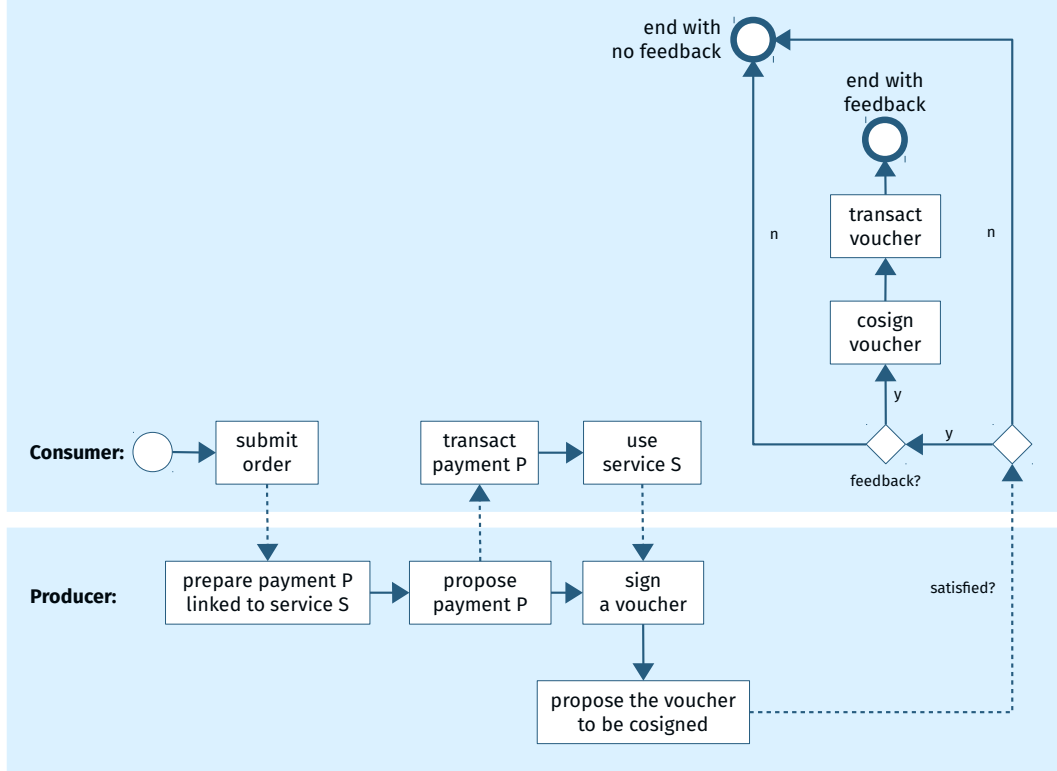


Figure 2.5: The cosigned voucher creates a monetary incentive to leave feedback (illustration based on [12]).

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

2.1.8 Insurance Models

Trust can be established based on reputation information [25]. A trust model helps the user to take decisions in concrete situations [31]. A reputation model however is more holistic, as it includes the view of all peers in the system about a given user [25]. Nevertheless, reputation and trust are closely related concepts.

Some trust models work in a similar fashion to insurance models. To better understand these models, the example of a supply chain with upstream suppliers and downstream customers is helpful. Peers trust each other up to some amount of money. In TrustDavis for example, peers can obtain weighted references from their neighbours [15]. If a user gives a reference, she would become liable to pay the reference price to any customer, if the supplier she gave the reference to cannot deliver the product or service. These references can be modeled as edges in a trust graph. In a trust graph each edge represents the maximum amount of money, that the originator is liable for or trusts the target of the reference with. A similar approach has been presented recently by Litos and Zindros

[31]. Trust and reputation are not directly linked to rewards, but can be expressed in monetary terms using such schemes.

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

Such a network of liabilities can serve as basis for concrete financial decisions. In a DDoS cooperative defense scenario, the target domain could assess if it is possible to trust a mitigator domain to solve a task, given the target domain can estimate the expected mitigation costs correctly. However, the reputation of a peer cannot be directly derived with this scheme, as reputation is mostly defined through a more collective view [25].

2.2 Reputation Engines

Besides reputation aggregation protocols, another important aspect of a reputation system is the reputation computation engine, where scores and metrics are calculated based on the inputs [25]. Most reputation systems in the P2P space are single-dimensional, meaning that only one factor (*e.g.* number of contributions) serves as input for the system [18]. For the output metric there are a broad range of different models. Surveys and classification about the different types of reputation engines are presented by Jøsang *et al.* in [25] and Schlosser *et al.* in [44]. The overview presented in this section is mainly based on these two works, with an emphasis on engines that seem suitable for the design of reputation systems in cooperative DDoS defense.

2.2.1 Probabilistic Engines

The reputation score is mostly accumulated linearly, exponentially or calculated with help of probabilistic density functions (*e.g.* the Beta reputation system) [5, 8]. In probabilistic engines, reputation is expressed as a probability. The expected value of the beta distribution forecasts the probability of a positive, future event x , based on the past binary events x and $\bar{x}$. The binary events represent positive and negative historical reputation ratings. The expected value of this distribution can be interpreted as a reputation score [5]. One particular example of a Beta reputation system applied to blockchain is found in the Topl protocol [13]. Topl is a proposed blockchain protocol to create profit sharing agreements with producers in emerging and frontier markets. The protocols reputation engine “Divine” builds upon a Beta reputation engine that facilitates due diligence and reduces counterparty risk [28].

2.2.2 Fuzzy Engines

Fuzzy reputation engines like REGRET [42] and FIRE [23] store multi-dimensional reputation data [25]. The peers behavior can be analyzed through fuzzy queries on the data stores holding this reputation data. A reputation score is stored in tuple-form. $rep = (a, b, i, d, v)$ is the rating from a about b , in interaction or transaction i , on dimension or skill d , with v being the actual rating value (*e.g.* in range $[-1, 1]$). With a fuzzy query $q = (a, -, -, quality, -)$ an agent can retrieve and aggregate all ratings concerning the *quality* dimension from peer a , to any other peer (“-” is the wildcard) in any interaction and with any score value. Insights on the different interaction dimensions like quality, price and service time can be gained by executing and aggregating the results of such queries on the data store. Since most reputation systems in P2P research are single-dimension, the thesis will not further investigate on the benefits of using such fuzzy models in a cooperative defense scenario.

2.2.3 Graph and Flow Engines

A graph structure is a suitable format to capture blockchain transactions and associated reputation scores. Reputation metrics can be calculated by following a path or flow in such graphs. This is why flow and graph based reputation engines are particularly interesting for incentive systems on blockchains. Current reputation management systems use centrality measures to indicate the importance of nodes. Table 2.2 summarizes goal, input and procedure of the flow algorithms discussed in this section.

	Network Flow [8, 10]	EigenTrust [26]	Proof-of-Importance[37]
Goal	Find selfish peers or compute maximum reputation between two peers	Detect malicious peers	Blockchain consensus
Input	Bitcoin transaction graph or reputation graph with reputation constraints	Local trust values	NEM transaction graph
Procedure	Net flow convergence rate or max flow	Aggregation of local trust values	NCDawareRank

Table 2.2: Comparison of flow engines

In EigenTrust, each user holds a local trust vector which is the net total of satisfactory and unsatisfactory transactions [26]. By weighting the local trust values with other values of trusted peers, the EigenTrust algorithm can assign each member in the system a trust value that converges to a global value after iterating through all the peers (iterative trust). The global trust vector contains the reputation scores of all the peers in the system, and is the same for every peer in the system. The convergence property of the algorithm enables each peer to have the same, global view of the network, independent of their local trust vector used as an input for the algorithm.

The New Economy Movement (NEM) is a novel, distributed ledger that leverages a modified version of EigenTrust to identify malicious nodes and reduce unsuccessful transac-

tions [37]. Additionally, the algorithm adjusts for node credibility. This “balance factor of trust”, proposed by Xiong and Liu in [50], is calculated based on the number of similar feedback ratings from two nodes. This approach protects the reputation system against attacks stemming from colluded local trust values, which are still subjective user data. The Proof-of-Importance (PoI) consensus algorithm in NEM determines who forges the next block (also known as harvesting or mining). The importance of a peer is calculated with the NCDawareRank algorithm, which works similar to PageRank. This algorithm requires the transaction graph as input. PoI is a metric that indicates how much a peer makes use of the network and with whom he interacts. Nodes that are very active in the network are assigned a higher importance and have a higher probability of harvesting new blocks.

Buechler *et al.* show, that an analysis of network flow in a transaction graph can produce reliable measures for reputation [10]. In comparison to EigenTrust, the net flow algorithm does not depend on subjective user input (ratings), but is applied on the transaction graph produced by the bitcoin transaction network. The goal of the algorithm is to find selfish peers which transact only among themselves.

The idea of the net flow algorithm is as follows. The Directed Acyclic Graph (DAG) of the bitcoin transaction network has a net in- and out flow (*i.e.* net flow) of zero, because no bitcoin can enter or leave the closed system. Consequently, there exists a positive or negative net flow in every subgraph of the DAG. The net flow convergence rate indicates, how fast the net flow converges to zero, when an incident edge is added to the edge set of the subgraph of interest. The assumption is, that a fast convergence towards zero is an indication of selfish peers that only transact among themselves. While iteratively increasing the edge set of the subgraph, the money flows of colluding subgraphs will converge faster towards zero than it is the case for honest peers and subgraphs. This holds, because honest peers generally transact more with other peers which results in a higher net flow. One benefit of using net flow to compute reputation scores is the lower computational complexity in comparison to other graph algorithms like Dijkstra’s shortest path between two nodes.

The max flow algorithm calculates the maximum flow between two nodes in a graph network. When the weighted edges between the nodes are interpreted as reputation constraints, this algorithm can return the maximum reputation between two distant nodes (or peers) in the graph [8]. The interpretation of such a metric would then be analogous to the interpretation of the maximum trust in an insurance model (see Section 2.1.8).

This section only presented a subset of available reputation engines. Other engines average or blur reputation scores over time. Amazon for example builds an average over all star ratings from all consumers for a specific product [25]. Engines with multi-dimensional scores could also average across different dimensions. In blurred engines, recent ratings have more weight. If only the latest reputation value is valid, the engine is called an “only-last” system. This system is an extreme instance of a blurred system [44]. The reputation engine in Moinet *et al.* is an example for such a blurred system, because reputation decay is modeled with an exponentially decreasing function [35] (see Section 2.1.2 for an example). In adaptive engines, the newly assigned score depends on the current reputation of the contributor [44]. The Sporas system for example defines a damping function which

slows the reputation increase for very reputable peers [21]. Thus, the most reputable peers converge towards an upper reputation limit.

2.3 Reaction and Service Differentiation

This section discusses, how reputation can be used to trigger a certain reaction in the system or to differentiate services. These mechanism should be applied with caution, because rewarding or penalizing users in the system based on their reputation value could lead to negative feedback loops [20]. For example, if job applicants are selected according to their credit rating, this can lead to a vicious cycles, where some applicants never find a job. But if job applicants are selected based on work performance scores, this is a reliable reputation signal because it is based on objective assessment of the applicants work history. INDAPSON [52] and PaySense [16] are incentive systems that define objective reputation metrics, which allow to couple reputation and rewards. In INDAPSON, prices depend on reputation. In PaySense, rewards are interpreted as reputation.

INDAPSON is an incentive data plan sharing system based on self-organizing network [52]. The incentive mechanism in this system is called reputation adaptive pricing (RAP). This pricing scheme was developed to enable fair sharing of mobile data plans. The reputation of a user is the ratio of downloaded and relayed (or shared) data. The price depends on the remaining energy level and on past reputation of the contributing mobile device which is sharing its data plan with other users in the network. This credit system makes it possible to soften the rigid time and quantity constraints when using mobile data plans.

PaySense is an integrated reward and reputation system for mobile crowd sensing [16]. Integrated means that reputation and reward are coupled. If the sensor earns a high reward, it also gains much reputation and vice versa. In this case, the money flow (reward) in the transaction history determines the reputation of a sensor. This principle is also applied by Carbonis feedback system on top of bitcoin, where the reputation increase is equal to the payment amount [12].

2.4 System Properties

Reputation systems have a few desired properties [51]:

- **Foundation of trust:** Repeated interactions and a transparent interaction history build a reliable foundation of trust.
- **Self-policing nature:** The system should be self-policing, with the social norms defined by the users and not by a central authority [26].
- **Carrots and sticks:** With incentives and penalties, users are more likely to behave according to the social norm.

- **Robustness:** The system should be robust against gaming attempts in reputation systems [18]. It needs not be fool proof, but reasonably secure against collusion and Sybil attacks, bad-mouthing (unfair ratings), ballot stuffing and identity whitewashing through re-entry. Current literature examines rating fraud, as a type of information fraud (see Figure 2.6). The goal of an attacker is to either increase the reputation of itself (ballot stuffing) or to decrease the reputation of others (bad-mouthing). An attacker can achieve his goals using a variety of attack models. A constant attacker behaves consistently evil, whereas a disturbing agent camouflages his actions skillfully.

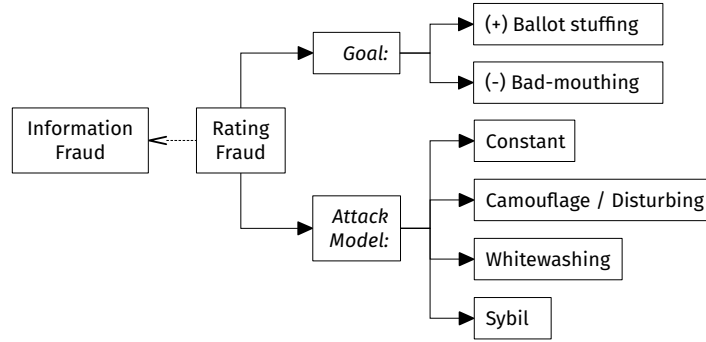


Figure 2.6: A fraudulent rating can be classified by goal and attack model [11, 44]

- **Accurate and verifiable scoring engine:** The scoring engine should provide accurate and verifiable metrics.
- **Anonymity and Privacy:** The subjective user feedback should be collected anonymously or bound to a pseudonym, to ensure honest feedback and guarantee some degree of privacy [16]. In a decentralized anonymous marketplace (DAM) [46] the reputation system requires additional properties to ensure a fair and complete listing of items, payments and reviews. Most importantly, such a marketplace ensures the unlinkability of reviews and payments (with associated customer information), without compromising the legitimacy of the review [46].

2.5 Agents and Identity Management

The types of agents participating in cooperative defense can be diverse: Attack targets, mitigators, intelligence firms and researchers contribute to the mitigation in different forms. According to Akamai (2017) in [2], “every player has a different piece of the puzzle” in cooperative defense. The competitive landscape in cooperative defense is diffuse, because all agents have different economic interests.

The framework defined by Schlosser *et al.* defines different categories of users and simulates their behavior in different types of reputation systems [44]. The honest user is always truth-telling. Malicious users randomly provide good and bad information. Evil

users interact only honestly with other evil users (collusion). The selfish user is a typical free-rider, which never gives any feedback and only takes action for its own benefit. Disturbing agents look honest initially, but can suddenly switch strategy and become malicious.

In most reputation systems, the users identity is bound to a pseudonym. This can be a contract or account address on the blockchain, which was created and activated in advance by an administrator or authority [9, 16]. This pseudonym can be enriched with additional information to construct a complete view of a users identity. In Liu *et al.*'s identity and reputation system, the users identity consists of the hashed physical identity information (*e.g.* biometric data or text) and the reputation coins in addition to the Ethereum public account address [33].

Iudex is a smart contract based reputation system, that was built during a hackathon [3]. This system accumulates various identities from different online communities in one account. Different scoring algorithms calculate reputation metrics based on the activity data (number of posts, followers, etc.) of the connected accounts.

2.6 Data Storage and Exchange

Consensys, a company creating software for the Ethereum blockchain, developed a set of recommendations for programming Ethereum smart contracts in Solidity ¹. The Ethereum blockchain “distributed supercomputer” comes at an expense. Since computation and storage on the blockchain are expensive, developers are advised to only store the minimum necessary information on-chain. For example, a division operation could be processed locally with only the numerator and denominator stored on-chain.

Colony is an Ethereum based protocol that allows to create decentralized organizations. The costs for huge amounts of data and expensive computations are the reason, why the Colony project stores the reputation of the users off-chain in a Merkle tree structure [4]. Only the root hash of the tree is stored on-chain. The tree as a whole is stored on external storage.

Work.nation, the blockchain powered portfolio for professionals (see Section 2.1), stores the reputation scores on the InterPlanetary File System (IPFS) and creates a reputation index in a traditional PostgreSQL database [49]. This reputation cache is updated every 15 seconds with new claims from the blockchain. Work.nation makes use of reputons, the IETF standard media type for reputation exchange [30]. Such a standard enables the interpretation and exchange of reputation claims. Claims are a way to register a reputation statement that a source entity makes about a target entity [20]. The reputon media type represents these claims in JavaScript Object Notation (JSON). Work.nation stores the IPFS hash of each claim in a smart contract. This is illustrated in Figure 2.7. The users of Work.nation first create claims about themselves. Afterwards, other peers confirm these claims by creating a new claim, with the `rated` property of the reputon pointing to the rating to confirm (*i.e.* the IPFS hash of the reputon) and `rating` value

¹<https://consensys.github.io/smart-contract-best-practices/>

0.5. Each reputon IPFS hash is recorded on the blockchain and mapped to the owner of the claim.

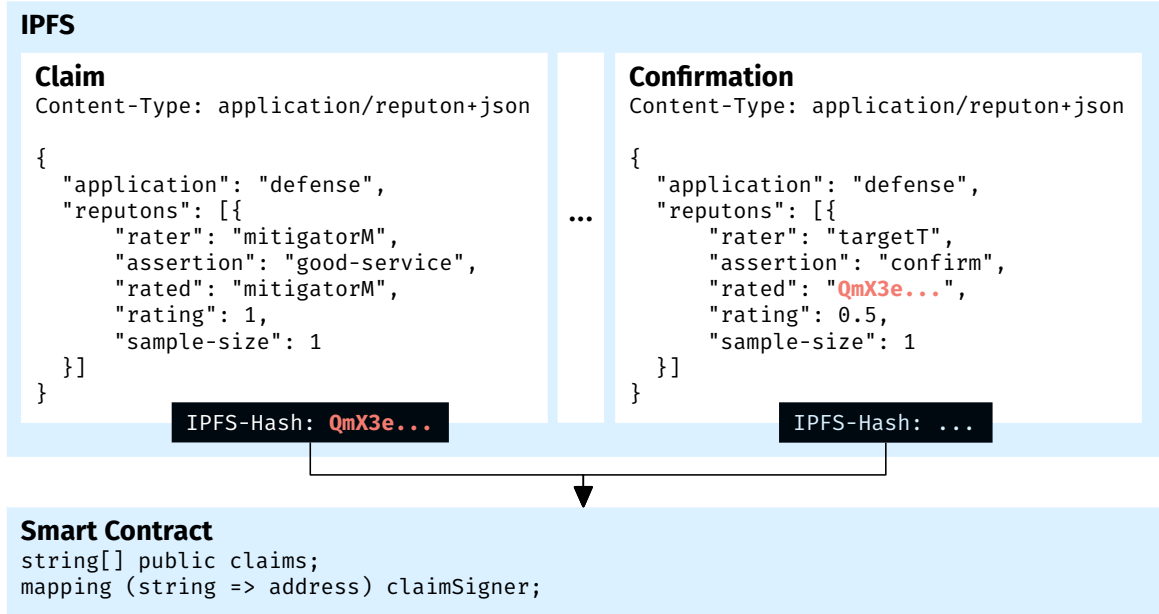


Figure 2.7: Storing reputons on- and off-chain [49]

In comparison to Colony, the Work.nation project does not organize reputation inside a tree, but provides a server component (API) which serves as reputation index. These two projects store reputation data differently, in order to optimize access times. On Work.nation, reputation can be claimed and confirmed without any particular restrictions. In Colony however, only the Colony token (short “CLNY”) holders are allowed to update the reputation state tree and submit a new reputation tree root hash. If different opinions about the next valid reputation state (*i.e.* reputation of all users in the next time period) exist, the smart contract can trace the transaction history. When the algorithm finds the disputed historical actions and reputation updates, it decides which submitter provided the correct next reputation state by comparing the action with the reputation update. Such an algorithm can run in logarithmic time, due to the tree structure of the data store. Hence, this approach called “reputation mining” reduces the computational effort on-chain, especially for complex reputation models, where reputation is computed across different groups, reputation dimensions and time periods (reputation decay). Since the Work.nation reputation model is less restrictive, data can be stored in a relational database.

2.7 Evaluation Frameworks and Metrics

Reputation systems are mostly built for a specific system context and evaluated independently from each other. Literature emphasizes the need for experimental evaluation of trust and reputation models [50]. Schlosser *et al.* presented a formal model to abstract

reputation system metrics [44]. Further they made use of the agent-based modeling and simulation suite “Repast” to assess the effectiveness of current reputation models.

The (repeated) Prisoner’s Dilemma is a game theoretic scenario used to evaluate reputation systems and describe the interaction between peers in a P2P network [8, 44]. This dilemma describes the situation where two partners in crime are offered a shorter time in prison if they confess and tell the truth. Rational peers choose to betray each other for the sake of higher payoffs. This always leads to the outcome where both could be better off if they kept silent (see Table 2.3).

	M silent	M betrays
T silent	$(-1, -1)$	$(-3, 0)$
T betrays	$(0, -3)$	$(-2, -2)$

Table 2.3: An exemplary payoff matrix for the general Prisoner’s Dilemma

To investigate fairness problems in the DDoS mitigation process, a basic scenario can be divided into three stages. Such a scenario is depicted in Table 2.4. Fairness problems can potentially arise between all of these stages. Firstly, between the mitigation request by the DDoS attack target T and the actual blocking of the malicious IP addresses by a mitigator domain M . Secondly, after the delivery of the mitigation service and the payment by domain T . Table 2.4 illustrates three possible mitigation histories and outcomes, depending on how T and M act.

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

Table 2.4: Scenarios in cooperative defense with DDoS attack target T and mitigator M

It is reasonable to assume that T ’s costs inflicted by the attack are higher than the costs of domain M , which is providing the mitigation service. The penalty pattern of the Prisoner’s Dilemma can be recognized in Table 2.4. A rational M would be better off not providing the costly mitigation service in the short term (*i.e.* betray). Also, T has strong incentives to refuse payment (*i.e.* betray). Since in a repeated game the roles of T and M could be swapped, they are better off cooperating in view of future attacks. A reputation and incentive scheme can incentivize the peers to do so, because it records and stores past behavior. This data can be analyzed by other peers before committing new transactions and can help them making better decisions. The peers will preferably transact with reputable colleagues. This leads to an increase of successful transactions overall, *i.e.* it increases social welfare. The percentage of successful interactions can serve as useful performance measure to evaluate a reputation and reward scheme [48].

Chapter 3

Design

This Chapter explains the design aspects related with processes of integrated payment and reputation ranking in cooperative defense. Furthermore, it considers exceptional and edge cases (*e.g.* what happens on abort if one party defects) and system flow (*i.e.* where data resides and how system states can change).

3.1 Incentive Design for Cooperative Defense

The blockchain-based OFE described in Section 2.1.5 has limitations when applied to cooperative DDoS defense. Whenever a mitigator submits a proof of blocking, there is no automated way for other peers to build a consensus on the quality of the service delivered. In other words, the receipt in a “payment-for-receipt” exchange process cannot be automatically issued by a smart contract, because in the worst case, any upload is accepted as successful delivery and the DDoS target domain would pay for a worthless receipt. Attack size and amount of payment are relevant factors that could result in serious financial losses for the attack target. In order to avoid this problem, the reputation process depends on the attack target to validate and rate the outcome of the mitigation service until an a priori agreed deadline. The limitations of designing exchange protocols with fixed delays are discussed in Section 2.1.5. The defense process illustrated in Figure 3.1 presents the protocol for mitigation or defense services by a mitigator M (*i.e.* the requested domain), in response to a DDoS attack at the target domain T (*i.e.* the requester or attack target).

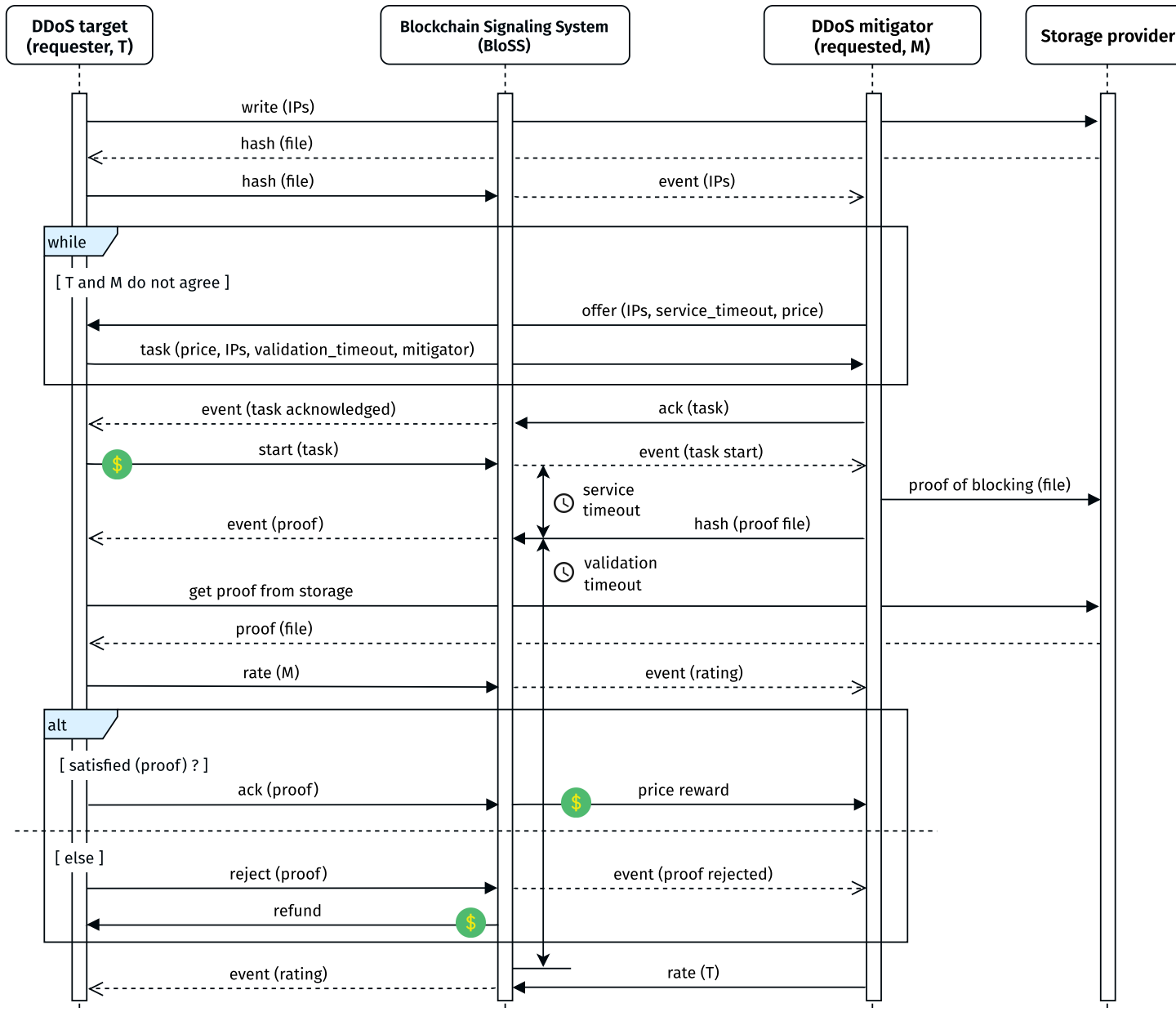


Figure 3.1: Illustrative DDoS mitigation service process which integrates payment and reputation ratings from/to both parties.

T writes the attack information (*e.g.* IP address ranges, packet captures, request headers, notes and other patterns [2]) to the external storage (*e.g.* IPFS). M is already subscribed to the incident report and attack information broadcast. If M decides to join forces on a particular mitigation request, it sends T an offer containing scope (*e.g.* IPs) and price. The negotiation of these contract parameters can take place off-chain, through a direct communication channel. For example, the Whisper protocol (Ethereum) could be used to agree on these contract parameters. This negotiation phase could as well be designed two-ways, allowing T to actively submit a “request for service” to M . Once the offer reaches T , the domain under attack can draft a genuine mitigation contract. In Figure 3.1 this contract is referred to as the “task” smart contract. When T creates this contract, it also pays the contract creation costs, *i.e.* the cost for having the task confirmed on-chain through a consensus algorithm. This task needs to be acknowledged (*i.e.* signed) by both parties. When T signs the contract, it also sends the specified price to the smart contract. The agreed price from the payable start-transaction (marked green in Figure 3.1) is locked in the smart contract, until the mitigation service is completed. In the best case scenario, both parties keep their promises and deliver money and service on time. The offers and contract proposals can be made repeatedly by the two domains until they find an agreement. During this process, M and T also agree on the deadlines for service delivery, validation, payment and rating. Each mitigator needs to agree with the service delivery and validation deadlines in order to contribute to the defense. Like the IP expiration in the underlying signaling system, these deadlines are measured in blocks [9]¹.

During the service time window, M is supposed to upload a proof of service. M will have an incentive to submit a proof (even a forged one), since time works against M . T is obliged to validate the effectiveness of the service and rate M during the validation time window. T will have an interest to vote and proceed in the mitigation process, since the clock works against T during this time window. Playing against the rules will result in a financial loss for both parties, as depicted in Figure 3.2. For example, if T misses to acknowledge or reject the proof in response to the delivered service, M will be rewarded. Therefore, T is penalized and has no chance to retrieve the payment. Both parties are free to abort the protocol if the counterparty did not deliver the expected result during the agreed time-frame. The decision flow that shows who gets rewarded on abort is visible in Figure 3.2. If M did not deliver the service, T will retrieve the payment. Similarly, if T did not respond during the validation time window, M will be reimbursed. M is not allowed to rate, if no proof was uploaded. This ensures that a users reputation is only changed through valid interactions which impedes bad-mouthing [11]. In order to bad-mouth a competing domain and deteriorate its reputation, the attacker would need to buy at least as many mitigation services from this competitor. This would increase the competitors profit and is therefore irrational [11]. In contrast to M , the process allows T to rate, even if no proof was uploaded during the service time window.

It is necessary to have an escalation and dispute resolution process in place in case T rejects a valid proof and fools M (*i.e.*, T is not honest). Apart from detecting dishonest validators, this complaint management system could serve to disclose rating fraud [11].

¹A transaction scheduler similar to <http://www.ethereum-alarm-clock.com/> can be used on Ethereum.

A dispute can then be escalated to a third party, like a court or the Precedent Protocol [34]. A democratic voting is also an option to resolve in the case of a dispute, but the challenge of building a trustworthy quorum remains. A “reputation-at-stake” mechanism as in Colony or Augur can incentivize the members of the quorum to vote with the consensus [4, 40]. Still, as mentioned by Swan (2015) in [47], these dispute resolution protocols only “decide the justiciability of a dispute—the point of law, not the point of fact” (p. 51). However, a negative rating by M can signal others the malicious behavior of T , and in this way partly restore justice (in terms of facts) for the betrayed mitigators in the long run.

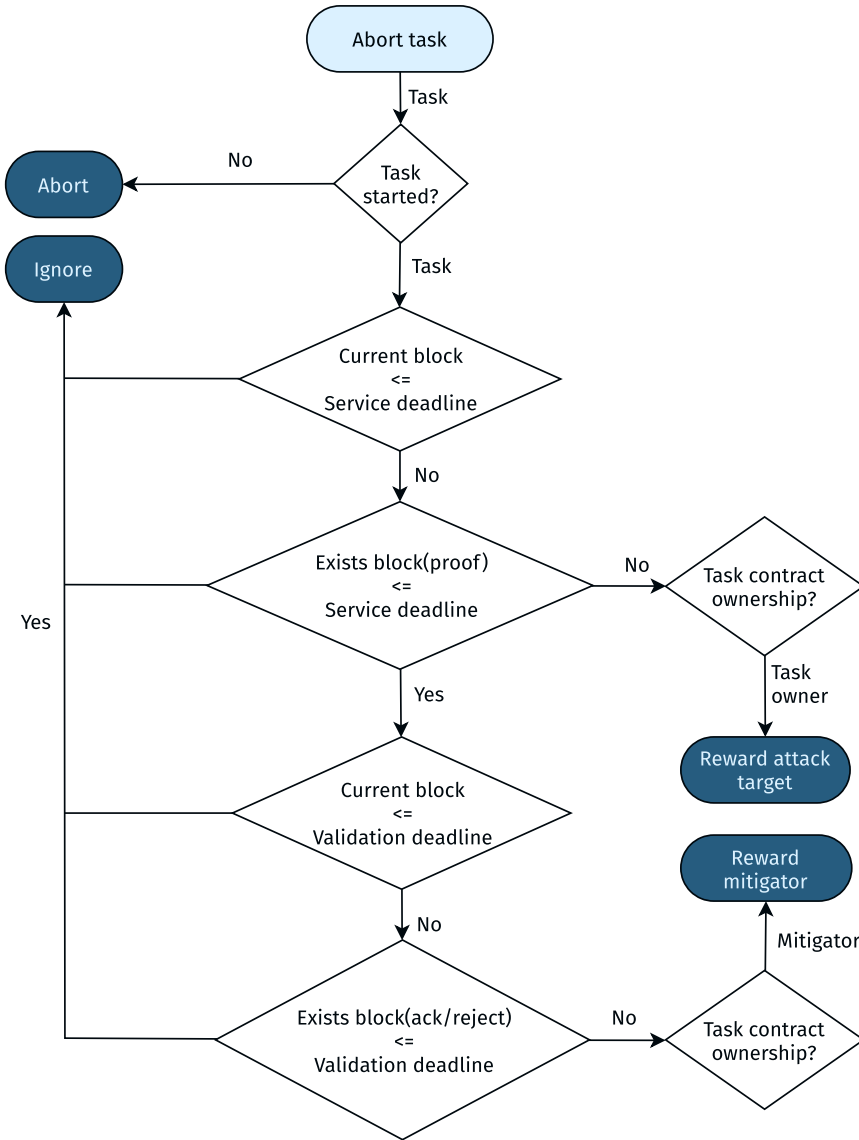


Figure 3.2: Payout process, where the target is refunded, if no proof was uploaded and the mitigator is rewarded, if the target does not validate in time

If M needs an incentive to give feedback, the process can be improved with a monetary “incentive to vote”. A deposit as presented in Section 2.1.4 or even a cosigned voucher (see Section 2.1.7) can be used in these situations. The former can be achieved by holding

back a fixed share of the price as collateral. In this way, M is encouraged to rate T in the end to receive the full amount of the reward. The latter option includes T paying an additional incentive on top of the already paid price. Holding back a collateral is more efficient from a cost perspective, so that additional expenses for creating a cosigned voucher can be avoided.

Liu *et al.* [32] mention timeliness and fairness as important properties of blockchain-based exchange protocols. According to the authors, a protocol is timely when “honest players can be certain that the protocol will be completed at a certain finite point in time. At completion of the protocol, the state of the exchange is final from that player’s perspective” (p. 4). Long term, either T wants the money back if services have not been delivered after the service deadline, or M wants to receive a rating and reward after the validation deadline expired. Any of the two parties with a desire to resolve the exchange can do this after deadline expiration, which is a well-defined point in time. So, the exchange process will conclude at one time. The decision to abort or conclude the exchange is final from the perspective of the domain and cannot be reverted.

Similarly, Liu *et al.* (2016) define a strongly fair protocol in [32] as a protocol where “at the time of protocol termination, either both players get what they want, or neither of them does” (p. 4). The protocol for reputation and reward presented in this section is not entirely fair according to this definition. The fairness property is not maintained in case T is malicious and deliberately rejects all services from M . However, since M can give a negative feedback for each disappointing transaction, the malicious T will get less reputable over time. This will serve other mitigators as a signal for caution in future mitigation contracts.

3.2 System Architecture

The high-level structure of the proposed reputation and reward architecture is shown in Figure 3.3. This Figure shows data flows and where reputation is stored and aggregated.

As in [9], it is necessary for each party to have an identifiable address on the blockchain. The customers in the prototype are identified by their Ethereum account addresses for sake of simplicity. An identity framework like Uport [19] would come with increased maintainability and security. For example, if a customer’s private key is compromised, the user is able to generate a new private key, but is still identifiable as the same customer [19]. A customer owned proxy contract (*i.e.* the “self-sovereign identity” on chain) always acts on their behalf and users never use the key directly, to interact with the network [19]. The identity smart contract holds the identities of all participating autonomous systems or domains. A limited **anonymity** (*i.e.* pseudo-anonymity or pseudonymity) of the rater is preserved, by disguising the real identity of the domain (see Section 2.4) and maintaining a pseudonym in form of an ID. However, for a particular reputation claim, the claim owner, the rated customer and the interaction are traceable. The reputation system for the Beaver marketplace in [46] ensures rating-interaction unlinkability by means of ring-signatures and zero-knowledge proofs. That level of anonymity was not achieved during the thesis and these concepts are not further investigated.

3.3 Reputation Exchange, Computation and Storage

Reputation is enriched with meta and context information. For example, when using the IETF standard media type for reputation exchange (“application/reputon+json”), reputation information contains additional attributes like the rater (*i.e.* origin of the rating) or a confidence level [30]. There is no need to store this additional information on the blockchain. Colony [4], an Ethereum based protocol for creating and managing decentralized organizations, creates a reputation Merkle tree and only stores the root hash on the blockchain. This strategy, to move reputation storage and computation off-chain for cost optimization, was explained in Section 2.6. We can assume that such off-chain storage strategy is used for any kind of data, that potentially grows large (*e.g.* attack information) or is computationally expensive.

Reputation computation can be a resource intensive procedure, being not suitable for execution in current blockchain environments (*e.g.* Ethereum VM). Thus, storing and computing these values should occur locally and the results of the computation can be synced with the blockchain [4]. Because the reputation data defined in this thesis is single-dimensional and binary (*e.g.* good or bad mitigation service provided), a single rating will not require computationally expensive operations (*e.g.* reputation computation across domains or reputation-dimensions as in Colony). The reputation merely serves as feedback mechanism to track the peers satisfaction and signalize this satisfaction to others. For this reason, each rating with reputation score is stored separately on the blockchain and no reputation mining procedure is necessary (see Section 2.6). Nevertheless, the binary value of each rating will be recorded in the reputation smart contract. This information needs to be accessible in smart contracts, to correctly resolve the mitigation case in the end.

3.4 Reputation Metrics

Reputation points used to rate target and mitigator are preferably not the same, because a task owner with a good rating does not necessarily need to be a good mitigator and vice-versa. Therefore, reputation earned as mitigator is stored separately from the reputation earned as attack target or task owner. A simple metric gives the analyzing peers a clear understanding of a peers reputation. Added to this, a complex metric might lead to feedback loops (see Section 2.3). There are two types of reputation sources, subjective and objective. While the subjective reputation is composed of positive and negative ratings from other peers, the objective, historical metrics can be obtained from the public blockchain history. For example, the reputation system prototype in this thesis allows to compute the following metrics (among others):

- **Customer age:** The customer age can be derived from the block timestamp when the customer ID was created.
- **Number of interactions:** Completed tasks (negative, positive and unknown rating) and the number of interactions for a customer are obtained through the public task states.

- **Average satisfaction:** The average satisfaction with a peer (ratio of positive and negative ratings) signalizes the general satisfaction with this customer.
- **Number of completed tasks:** Observing the amount of completed tasks over time helps to compare historical with current customer performance.

This system leverages the objective (*e.g.* number of transactions/interactions) and subjective metrics (*e.g.* ratings) proposed in Xiong and Liu [50]. The scores earned by ratings from other peers (subjective ratings) and the total amount of transactions (objective metrics) are equally important to have a good understanding of a peers trustworthiness. This approach allows peers to calculate a simple trust metric, which helps them to decide on trustworthy business partners. A “balance factor of trust” to correct for the trustworthiness of the subjective ratings is not implemented as part of the thesis.

3.5 Metric Verification Process

Section 2.4 mentioned the need for a verifiable scoring engine. Therefore, the data consumed through the reputation API should be presented in a way that any consumer is able to verify the data integrity. Since all data is publicly available on the blockchain, peers can check its integrity. Thus, we can safely assume that every peer trusts information (*i.e.* reputons and claims) confirmed on the blockchain and referenced on IPFS. The reputation engine (*i.e.* the REST based reputation API, see Figure 3.3) periodically collects reputation data from the ledger and aggregates this information. Such an aggregated reputation summary about a customer, always references the historical mitigation contracts in which the reputation was originally claimed. This leads to a situation where we have no more than exactly two reputation claims per mitigation task. However, there might be less than two claims if one contract party refused to give feedback. The implementation details of the reputation metric (*i.e.* the reputation summary) and a concrete example for the metric verification process will be presented in Sections 4.6 and 4.7. Using such a verification procedure, every client can trust the aggregated reputation data by comparing with the reputation claims on the blockchain. Furthermore, each change to the reputation smart contract (see Figure 3.3) generates a transaction receipt. The transactions and their receipts can be accessed and used to recover the complete change history of the contract. An example on how to perform such a transaction trace to verify changes to the reputation contract is provided in Section 4.7 as well.

Chapter 4

Implementation

Reputation and incentive systems have already been successfully applied to blockchain systems. Some projects considered reputation management as integrated feature when constructing the network (*e.g.* NEM). Thus, higher-level applications can potentially use these functionalities by default. Other projects pursued a layered approach and implemented protocols and extensions on top of existing blockchain networks. This work follows the latter approach, because the Ethereum blockchain does not come with built-in reputation mechanisms. For the implementation of this thesis, the smart contracts serve as an abstraction layer on the blockchain, and the reputation mechanisms are not directly built into the blockchain core. There exist plenty of examples and reference implementations for different blockchains, that implement reputation management in this layered fashion. Bitrated for example, is a reputation platform for bitcoin [7]. Since the system architecture in Figure 3.3 uses an API to aggregate reputation and answer reputation queries, the Bitrated API can be used as source of inspiration on how to design the endpoints. Other projects like Colony, Augur, Work.nation, Backfeed and Iudex are more relevant when it comes to the design of the smart contracts, because these systems are built on the Ethereum blockchain and most of them open source their code on GitHub. Backfeed is a pre-alpha reputation system, but does not seem to be actively maintained [6].

4.1 Construction of the Mitigation Task Index

The software architecture of the prototype is inspired by the Work.nation project [49], with slight modifications. Whereas Work.nation uses a PostgreSQL database as reputation cache, this thesis creates a mitigation task cache in a MongoDB database. Based on that data, the reputation API provides reputation metrics to interested consumers over well defined endpoints. Figure 3.3 shows, how task index and reputation API are embedded into the overall system architecture.

The reputation API was built with the Ruby on Rails (RoR) web-application framework. The `sidekiq-scheduler`¹ appends new contract information to the local index at

¹<https://moove-it.github.io/sidekiq-scheduler>

an interval of 15 seconds. Since we assume that data on the ledger is immutable (*e.g.* mitigation contract participants never change), new data can be appended to the local database in this periodic manner. The database is updated using a simple counter (`known_task_count`), to mark the position of the last update (see Listing 4.1). Thus, information appended previously will not be stored or processed twice. The same caching procedure could be applied to cache reputons.

```

1  def get_latest_tasks
2    ...
3    task_count = contract.call.task_count
4    known_task_count = Integer(REDIS.get('known_task_count') || -1)
5
6    # append only new tasks
7    (known_task_count + 1...task_count).each do |task_index|
8      get_task(task_index, contract)
9    end
10  end
11
12  def get_task(task_index, contract)
13    # fetch mitigation contract from the ledger
14    task = contract.call.tasks(task_index)
15    REDIS.set('known_task_count', task_index)
16
17    # create a new task entry in the database
18    MitigationTask.create(mitgn: MITIGATION_CONTRACT_ADDRESS,
19                          _id: task_index,
20                          target: task[0],
21                          mitigator: task[1])
22  end

```

Listing 4.1: The ruby class `Decentral::Task` in the file `/api/app/lib/decentral.rb` is periodically executed by the `sidekiq-scheduler`. New tasks are periodically appended to the local database. The order of the mitigation task attributes (*e.g.* `task[0]` or `task[1]`) is defined in Listing 4.2

4.2 Mitigation Tasks

This section describes, how the states of all mitigation tasks were implemented in a smart contract and how they will be used throughout the thesis. The task state is represented by an integer in range $[0, 6]$ (see Table 4.1). The relevant excerpt from the mitigation smart contract is given in Listing 4.2. This contract stores the state of all mitigation tasks in the public `tasks` contract variable.

State Code	Task State		State Description
0	Completed		A task can only be completed before payment/start and after the final rating deadline. The “complete” action triggers payment according to the diagrams in Figures 3.2 and 4.1.
1	Initialized		Every new task is created in this state.
2	Approved		The task is approved once the mitigator accepts all contract parameters (<i>i.e.</i> signs the mitigation contract)
3	Started		The attack target pays the price to start the mitigation task. This action initializes the task <code>startTime</code> in Listing 4.2. An event is emitted to notify the mitigator.
4	Proof uploaded		The mitigator uploaded a non-empty proof of service
5	Validated	Acknowledged	The attack target is satisfied with the mitigation service and the payout is made at completion time.
6		Rejected	The attack target is dissatisfied with the mitigation service and refunded at completion time.

Table 4.1: Task states derived from the Solidity enum in Listing 4.2.

```

1  struct Task {
2      address attackTarget;
3      address mitigator;
4      uint serviceDeadline;
5      uint validationDeadline;
6      uint ratingDeadline;
7      uint startTime;
8      uint price;
9      State state;
10     string proof;
11     string scope;
12 }
13
14 enum State { completed, init, approved, started,
15             proofUploaded, acknowledged, rejected }
16
17 Task[] public tasks;

```

Listing 4.2: Excerpt from the mitigation smart contract in file `/contracts/Mitigation.sol`

4.3 Mitigation Protocol

Figure 4.1 depicts the (technology-indepdent) mitigation protocol developed in Chapter 3. It defines how the customers should behave in a cooperative defense. The different types of customers are marked green if it is a desired behavior, red otherwise. These customers types implemented in the simulator are described in more detail in Tables 5.1 and 5.2. Throughout the next chapters, the rational and satisfied customers are referred to as the “desired customer strategies”.

In addition to the service and validation deadline (see the process design in Figure 3.1), the prototype implements a final rating deadline. During this last rating time window, a dissatisfied mitigator M can still dispute a rejected service. When this time window has passed, M is assumed to no longer be interested in the payout or having accepted the rating from the target T . The rules of the smart contract define a positive rating by M as the only allowed response to a positive rating (ack) from T . In the same way, receiving negative or no feedback from T cannot (altruistically) be rewarded with a positive rating by M . Nevertheless, a dispute process should be in place for customers to challenge any payout or rating.

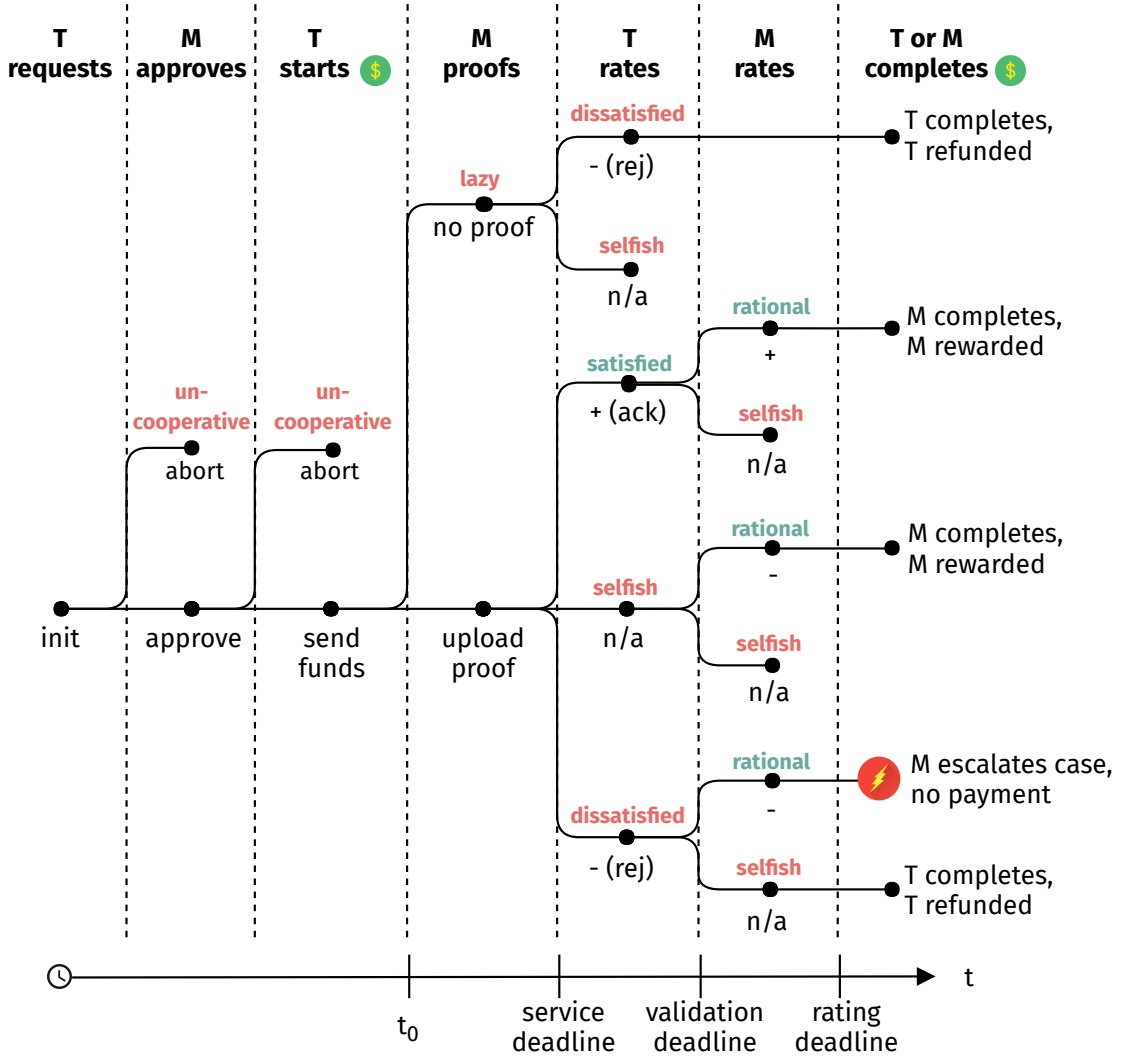


Figure 4.1: Protocol that shows the different customer strategies from Tables 5.1 and 5.2. The timeline at the bottom shows the start time t_0 and the three deadlines. An escalation process is required, since M might want to reclaim the payout of a rejected service (end state marked in red).

As independent observers, we cannot make a statement about the truthfulness or quality of the (proof of) service. Even though blockchain technology can preserve an audit trail for all transactions, it cannot compensate for a lack of ground-truth [11]. This holds for the uploaded proof of service as well as for user-defined, subjective ratings. There is

no automated way to fully determine the truthfulness of a proof or rating. Hence, the correct input of customers remains of paramount importance for the quality of the system as whole. Nonetheless, as soon as T decided on the quality of the delivered mitigation service, this information is published by rating and validating the mitigation service. In this sense, T cannot be malicious, only satisfied or dissatisfied. If T is honest, it either rates positively and acknowledges (satisfied) or rates negatively and rejects the service (dissatisfied). Then, an rational M always rates according to the expectation of T . This rational M responds to a rejected service by rating T negatively. However, if the rational M received a positive feedback from T , it will also respond positively. When there was no feedback (*i.e.* T selfish), a rational M is only allowed to rate negatively.

4.4 Storage of Reputation Data

Reputation storage is implemented partially off- and on-chain. The actual reputation data is not stored in the database of the API, but retrieved directly from the IPFS gateway. The task index of the API contains the task IDs and contract participants. To obtain reputation data, the API controller sets off a read request to the blockchain. An example to fetch all reputon IPFS keys for a customer (in the role of attack target) is given in Listing 4.3.

```

1 def get_target_reputons(addr)
2   target_tasks = MitigationTask.where(target: addr)
3   # reputons[:task_id][0] is claim of target about the mitigator
4   # reputons[:task_id][1] is claim of mitigator about the target
5   target_tasks.map do |t|
6     { ipfs_key: CONTRACT.call.get_reputon(t._id, 1),
7       task_id: t._id }
8   end
9 end

```

Listing 4.3: Obtains all IPFS reputon keys from all tasks from the ledger, in which the customer with address `addr` was enrolled as attack target.

The reputation smart contract stores the IPFS hash of every reputation claim (see Listing 4.4). The `reputons` mapping avoids duplicate claim registration. The Solidity state variable `interactions` maps every mitigation task ID to an array of size two. Every task can have at most two reputons assigned. First, the reputation statement from the target about the mitigator and second, the reputation statement from the mitigator about the attack target. These reputons hold complete reputation information, including meta-data. The `ratings` state variable only records the binary reputation value to correctly conclude mitigation tasks.

```

1 uint256 public reputonCount;
2
3 // reputon hash => reputon owner
4 // detect duplicate claims
5 // and register claim owners
6 mapping(string => address) reputons;

```

```

7
8 // interaction => reputon IPFS hash
9 //   ratings[i][0] is claim of attack target about mitigator
10 //   ratings[i][1] is claim of mitigator about attack target
11 mapping(uint => string[2]) interactions;
12
13 // interaction => rating
14 //   0: no (neutral) rating
15 //   1: positive rating
16 //   2: negative rating
17 mapping(uint => uint[2]) ratings;

```

Listing 4.4: Excerpt from the reputation smart contract in file `/contracts/Reputation.sol`

4.5 Reputation Data Format

As discussed in Section 2.6, reputation claims can be stored as reputons. These JSON objects are stored on IPFS. Listing 4.5 and 4.6 show this format for reputation statements about the mitigator and the target of an attack. The reputation API provides endpoints (*e.g.* `/customers/:customer_addr/reputons`) to obtain the IPFS reputon hash. Afterwards, a reputon can be fetched from the IPFS gateway. The inability to evaluate the rating value of a reputon in a smart contract is a severe disadvantage.

The properties of such a reputon can be interpreted as follows:

- The `assertion` value indicates, whether the reputation statement was made about an attack target or mitigator. In Listing 4.5 for example, the reputation statement was made about the mitigation service delivered by a mitigator. The receiving side (*i.e.* the “rated” customer) of the claim is implicitly recorded in the mitigation smart contract, since every mitigation contract is created for exactly two customers (see Listing 4.2).
- A rating value of zero is a negative feedback. The rating value of one is interpreted as a positive reputation statement. Intuitively, when no reputon is supplied for a mitigation task and customer, there exists no rating property and the rating for this customer is interpreted as “neutral”.
- The `rated` property of the reputon records the mitigation task ID.
- The `sample-size` specifies “the number of data points used to compute the rating” [30] and is fixed to 1, since the reputation statement originates from exactly one customer. Even though not being a required attribute by the media type specification [30], it improves the information quality of the reputation statement.

```

1 {
2   "application": "mitigation",
3   "reputons": [
4     {
5       "rater": "0xf43c5e7fd6997d10d3ad04f4ae5a8106c5803a73",

```

```

6     "assertion": "proof-ok",
7     "rated": 20,
8     "rating": 0,
9     "sample-size": 1
10  }
11  ]
12  }

```

Listing 4.5: Data format for a reputation statement about a mitigator.

```

1  {
2    "application": "mitigation",
3    "reputons": [
4      {
5        "rater": "0xa84eeae5c210933f8465c4c32e74463030b04393",
6        "assertion": "target-ok",
7        "rated": 21,
8        "rating": 0,
9        "sample-size": 1
10     }
11   ]
12 }

```

Listing 4.6: Data format for a reputation statement about an attack target.

4.6 Reputation Metrics

The reputation API presents reputation summaries to consumers. An example of such a summary for an arbitrary customer is given in Listing 4.7. Since this thesis operates with binary reputation, a rating value (see Section 4.5) of zero is a negative feedback. The `rating_summary` map of this summary reveals, that customer `0x113...` earned more negative than positive reputation for his mitigation service. In light of the design requirements in Section 3.4, the average satisfaction with the mitigation services of this customer amounts to $sat(c) = \frac{positive(c)}{negative(c)} = 0/2$ which is 0%. The reputation summary also shows, that this customer received no feedback and thus neutral ratings in three of total five interactions. These are only the interactions for which this customer was enrolled as mitigator. A similar summary can be obtained from the `/targets/:target_addr/reputation` endpoint, to learn about the reputation claims obtained in tasks, where customer `0x113...` was enrolled as the attack target. Additionally, the `/customers/:customer_addr/reputation` endpoint summarizes reputation earned as mitigator as well as reputation earned as attack target. This metric should be interpreted with caution, because it obfuscates the distinction between target and mitigator reputation (see Section 3.4).

```

1  {
2    "rating_source": {
3      "negative": [20, 26],
4      "neutral": [2, 8, 14],
5      "positive": []
6    },
7    "rating_summary": {

```

```

8      "negative": 2,
9      "neutral": 3,
10     "positive": 0
11   }
12 }

```

Listing 4.7: Exemplary reputation summary for the arbitrary mitigator `0x113...` received from the `/mitigators/:mitigator_addr/reputation` endpoint of the reputation API. negative reputation is computed from ratings with a rating value of zero, whereas positive reputation is calculated from ratings with rating value one.

The reputation computation is based on the task IDs in the `rating_source` map of the summary. The next section gives an example of how this information can be used to verify the correctness of the reputation metric. Further, one could obtain additional information about whether or not the task was successfully validated with a request to the `/mitigation_tasks/:id/fetch` endpoint of the API. A high-level documentation of the API endpoint is appended in Figure A.1.

4.7 Metric Verification Process

Since the reputation API is centralized, a consumer needs to be able to independently verify a requested reputation metric (*e.g.* Listing 4.7). The reason for this design feature was given in Section 3.5. The remaining part of this section gives an example how to verify a reputation summary consumed through the reputation API.

An HTTP GET request against the `/infos` endpoint of the reputation API returns a set of environment variables. These environment variables are used by the reputation engine to connect to the various services and aggregate reputation information. An example result of such a request is given in Listing 4.8.

```

1 {
2   "identity_contract": "0xd122c06c5569566ed42782278ae43b463d6332c1",
3   "reputation_contract": "0x21f8aa14692e6645c5ac2726d5e020bc721edde4",
4   "mitigation_contract": "0x414002558a7f6d0f114a1a98ae7f12bcdb4d0d2",
5   "eth_rpc_url": "http://172.17.0.2:8545",
6   "ipfs_gateway": "172.17.0.5:8080"
7 }

```

Listing 4.8: Result of an HTTP GET request to the `/infos` endpoint of the reputation API. The reputation information consumed through this API instance will be computed based on data stored at addresses `0xd12...`, `0x21f...` and `0x414...` on the blockchain.

With the connection information and the details of the `rating_source` in Listing 4.7, every consumer of the API is able to verify the integrity of reputation summaries independently by comparing with the data stored on the blockchain. Listing 4.9 presents an algorithm for this purpose. This verification can happen in two steps. First, the public smart contract interface can be invoked to retrieve the reputation data and compare the result with the reputation summary obtained from the API endpoints. This is verification step is done in `validateWithSCInterface()`. Second, changes for a particular reputation contract can be traced to find the transaction, in which the reputation

update for the customer account was confirmed on the blockchain. This happens in `validateByTxLookup()`. This function loops through all transactions in all blocks in a given time window (*e.g.* between start and validation time) and filters the blocks that alter the customers reputation. Naturally, this does only work in the case of positive or negative ratings, since for neutral ratings no feedback is supplied. Because this reputation system works with binary feedback, a positive rating has a rating value of one and a negative rating a rating value of zero (see Section 4.6). This algorithm is written in JavaScript, but verification can happen in a programming language of choice. The exemplary output (applied to the same customer `0x113...` and ledger state as in Listing 4.7) is provided in Listing 4.10. This output proofs the two negative ratings to be truthful. They originate from the two customers `0xf43...` and `0x786...`. Further, these addresses can also be compared with the contents of the reputon. The reputon in Listing 4.5 confirms, that customer `0xf43...` made this negative reputation statement (*i.e.* not satisfied with the proof of service) about customer `0x113...` for the mitigation task with ID 20.

```

1  var taskIdsVerifyNegative = [20, 26];
2  var taskIdsVerifyNeutral = [2, 8, 14];
3
4  var negative = neutral = 0;
5
6  new Promise(async (resolve, reject) => {
7    for (var _id of taskIdsVerifyNegative) {
8      var rating = await validateWithSCIFace(_id);
9      var reputon = ReputationInstance.getReputon(_id, 0);
10     await validateByTxLookup(_id, reputon);
11     if (rating === 0) negative++;
12   }
13   for (var _id of taskIdsVerifyNeutral) {
14     try {
15       await validateWithSCIFace(_id);
16     } catch (e) {
17       if (e.message.startsWith('multihash too short')) neutral++;
18     }
19   }
20   resolve();
21 }).then(() => {
22   console.log("Verified", negative, "negative feedbacks");
23   console.log("Verified", neutral, "neutral feedbacks");
24 });
25
26 function validateWithSCIFace(_task_id) {
27   var task = MitigationContractInstance.tasks(_task_id);
28   // reputon 0: claim about mitigator
29   // reputon 1: claim about target
30   var reputonHash = ReputationContractInstance.getReputon(_task_id, 0);
31   return new Promise((resolve, reject) => {
32     ipfs.files.cat(`/ipfs/${reputonHash}`, (err, file) => {
33       if (err) reject(err);
34       var reputon = JSON.parse(file.toString());
35       var rating = reputon['reputons'][0]['rating'];
36       resolve(rating);
37     });
38   });

```

```

39 }
40
41 function validateByTxLookup(_task_id, _reputon) {
42     var task = MitigationInstance.tasks(_task_id);
43     var targetAddr = task[0];
44     var mitigatorAddr = task[1];
45     var startTime = task[4].toNumber();
46     var validationDeadline = task[3].toNumber();
47
48     for (var i = startTime; i <= startTime + validationDeadline; i++) {
49         var block = web3.eth.getBlock(i, true);
50         if (block == null || block.transactions == null)
51             continue;
52
53         for (var trx of block.transactions) {
54             if (reputationAddress === trx.to
55                 && ( trx.from == targetAddr || trx.from == mitigatorAddr ) ) {
56                 var input = decoder.decodeData(trx.input);
57                 var txReputon = input['inputs'][2];
58                 if (_reputon === txReputon) {
59                     console.log("Valid rating");
60                     console.log(" from customer addr:", trx.from);
61                     console.log(" to contract addr:", trx.to);
62                     console.log(" tx hash:", trx.hash);
63                     console.log(" blockHash:", trx.blockHash);
64                     console.log(" blockNumber:", trx.blockNumber);
65                     console.log(" time:", block.timestamp,
66                                 new Date(block.timestamp * 1000).toGMTString());
67                     console.log(" with ipfs reputon hash: ", txReputon);
68                 }
69             }
70         };
71     }
72 }

```

Listing 4.9: Excerpt from the metric verification example in `/fixtures/evaluation_snippets/verification`. The IPFS gateway returns the expected error, if the reputon hash is empty (*i.e.* neutral rating feedback). `validateWithSCIFace()` relies on the contract instance addresses obtained through the `/infos` endpoint of the API (see Listing 4.8). `validateByTxLookup()` filters reputation relevant blockchain transactions.

```

1 Valid rating
2   from customer addr: 0xf43...
3   to contract addr: 0x21f...
4   tx hash: 0x854...
5   blockHash: 0x59d...
6   blockNumber: 8924
7   time: 1517047422 Sat, 27 Jan 2018 10:03:42 GMT
8   with ipfs reputon hash: QmNYT71P1M2mMKswWnuapCCKoJGs5o1a7aAc6ksDga27wX
9 Valid rating
10  from customer addr: 0x786...
11  to contract addr: 0x21f...
12  tx hash: 0x320...
13  blockHash: 0x649...
14  blockNumber: 8996
15  time: 1517048627 Sat, 27 Jan 2018 10:23:47 GMT
16  with ipfs reputon hash: QmYQaqXCBwcsT2LfVX5o29inHmhj2L6Zpbj91wnAKRTiEP

```

17 Verified 2 negative feedbacks
 18 Verified 3 neutral feedbacks

Listing 4.10: Example result of the verification algorithm in Listing 4.9. There are no reputons available for neutral feedbacks, since this is the exact definition of such.

4.8 Methodology

All software was tested and developed on Docker containers before deployment on the evaluation hardware in Chapter 5. A snapshot of the Solidity test coverage results is appended in Section A.2. Table 4.2 shows the components of these containers. The management script `/fixtures/manage.sh` allows to recreate and start the fully containerized development environment (or only selected parts of it).

Image Name (Prefix prototype/)	Container Instance (Prefix prototype_)	Description
geth	peer1	Ethereum mining node, using the geth command line interface (CLI)
geth	peer2	Second full Ethereum node, but non-mining
bootnode	bootnode	Stripped down version of a full Ethereum node, using only the bootnode executable, acting as a “lightweight bootstrap node ² ” for automatic node discovery
ipfs	ipfs	IPFS gateway
mongo	mongo	MongoDB instance to cache new mitigation tasks
api	api	Ruby on Rails API for reputation aggregation
redis	redis	Redis in-memory database, to store sidekiq task schedule (the pipeline) and the known_task_count
simulator	simulator	Node.js instance with simulator script
eth-netstats	netstats	Ethereum network status monitor
eth-netstats-intel	netstats-intel	Ethereum network intelligence API, the backend of the monitoring fronted

Table 4.2: Docker containers used to develop and test the prototype

²<https://github.com/ethereum/go-ethereum>

Chapter 5

Evaluation

5.1 Simulation Setup

The simulation experiments were performed in the Communication Systems Network Lab at the University of Zurich. More precisely, the setup comprised two identically equipped Dell XPS machines, with Intel Core i7 (4 Cores, 3.40 GHz) CPUs and 8GiB of RAM each. One machine hosted an Ethereum bootnode and one mining peer. The second machine hosted the second, non-mining `geth` instance and the Node.js simulator script. In addition, an Ethereum network statistics dashboard (`eth-netstats`) has been deployed for monitoring purposes on this second machine.

The simulator uses the Web3 Ethereum JavaScript API and runs in two different modes. The test mode was used to verify the correct behavior of the simulated customer strategies. The acceptance mode simulates an environment close to reality. The tests in acceptance mode include improvements and feedback from earlier experiments, because they were executed near to the end of the thesis. The reminder of this chapter is structured as follows:

1. **Test mode (see Section Section 5.4):**

The test mode lets all customers complete a mitigation contract with each other only once. The behavior of the different types of simulated customers is described in Section 5.3. All 16 possible combinations of customers (4 targets $\times$ 4 mitigators) are tested, to ensure that they run as expected. Section 5.4 explains, how the test mode was used to verify the correct behavior of these customer profiles.

2. **Acceptance mode (see Section Section 5.5):**

This mode simulates continuous DDoS attacks and the creation of mitigation tasks. The acceptance mode allows the simulated customers to make decisions based on past reputation values. Mitigation contracts are created on a regular basis and two randomly chosen players play the repeated game against (or with) each other. There exists an acceptance criteria in this mode, *i.e.* the two assigned players start the game considering past reputation scores. In a real world scenario customers are

unwilling to interact with un reputable colleagues. The acceptance function takes as input the past reputation of the counterparty [44]. Based on this input, the acceptance function evaluates whether or not to advance the mitigation task. The simulated customers continue with the mitigation actions, if the Beta reputation of the counterparty exceeds a value of 30%. Varying time-windows were chosen for the deadlines, such that they might lead to situations, where customers miss to meet a deadline. For example, the code that simulates a mitigator uploading a proof with the minimum service deadline of three blocks might be not executed fast enough, because the block time of the blockchain increases faster due to other transactions being processed. The service, validation and final rating/escalation deadlines are sampled randomly for each task in acceptance mode. Service deadlines are chosen in a range of $[3, 13]$ blocks and validation deadlines in range of $[17, 27]$ blocks. Final rating deadlines are sampled in the range of $[32, 42]$ blocks. Before the new mitigation contract is created, all peers are funded with 10 Ethers. The mitigation contract price was fixed to one Ether for all tasks.

Methodology wise, the experiments were performed in a top down approach. The goal was to find a setting, where consistent, desired customer behavior is distinguishable from undesired or even malicious behavior (by looking at reputation values).

5.2 Evaluation Metrics

The raw metrics to describe reputation are the amount of positive ($positive(c)$), neutral ($neutral(c)$) and negative ($negative(c)$) ratings of customer c . These metrics were obtained through the reputation API. Listing 5.2 has been modified to request these three metrics every 200 seconds. The average of these positive, negative and neutral reputation values by time and customer strategy for all experiments are presented in Appendix B.

In acceptance mode, the players of a mitigation contract consider the Beta reputation of the counterparty before making their first move in the game. Beta reputation is assumed to be a suitable model for the purpose of this evaluation, since the reputation system of this thesis works with binary reputation feedback (see Section 2.2.1 and 3.4). The Beta function outputs reputation scores in range of $[0, 1]$, where 0.5 is the initial, neutral score of every new customer. In acceptance mode, no customer is willing to work with a counterparty that has a Beta reputation lower than 0.3. Beta reputation for a customer c was calculated as the expected value $E(p) = \frac{\alpha}{\alpha + \beta}$, with $\alpha = positive(c) + 1$ and $\beta = negative(c) + 1$ being the accumulated positive and negative reputation values at one point in time [5]. This value reflects the probability of a future positive interaction with customer c , based on its past positive and negative ratings. Likewise to the raw metrics (*i.e.* positive, negative and neutral reputation), the Beta reputation in this chapter is averaged by time and customer strategy.

5.3 Customer Profiles and Strategies

The behavior of the different customers in the simulator is summarized in Tables 5.1 and 5.2. These customer profiles were selected with the goal in mind to build a reputation system that reveals freeloaders and false-reporters. Revealing and expelling lazy customers can reduce free-riding. Similarly, exposing untruthful customers could potentially reduce false-reporting. However, this is more difficult than detecting free-riding, because such a fraudulent case needs close examination by a tribunal.

Strategy	Description	Behavior:		
		<i>abort</i>	<i>start</i> (<i>payment</i>)	<i>rate</i>
Uncooperative:	Aborts before payment	✓		
Selfish:	Pays, but does not leave feedback		✓	
Satisfied:	Pays and gives positive feedback if a proof was uploaded		✓	+ (ack)
Dissatisfied:	Pays, always gives negative feedback		✓	− (rej)

Table 5.1: Attack target (T) strategies: The satisfied T acknowledges (ack) and accepts the service if a proof was uploaded. The dissatisfied T always rejects (rej) the service.

Strategy	Description	Behavior:			
		<i>abort</i>	<i>approve</i>	<i>proof</i>	<i>rate</i>
Uncooperative:	Aborts before approval	✓			
Lazy:	Delivers no service (no rating allowed)		✓		
Selfish:	Signs contract and uploads proof, but never rates		✓	✓	
Rational:	Signs contract, uploads proof and rates according to T 's expectations		✓	✓	+/-

Table 5.2: Mitigator (M) strategies

5.4 Customer Behavior in Test Mode

In order to conduct experiments with randomly assigned customer strategies and repeated tasks (*i.e.* the acceptance mode of the simulator), the correct behavior of every strategy has to be reviewed first. If only one customer profile behaves differently than expected, this can lead to misleading and skewed results in the evaluation of the acceptance mode. Table 5.3 lists all possible combinations of mitigator-target strategies and their assigned mitigation contract ID in test mode. The task end states can be obtained with Listing 5.1. This snippet performs a HTTP GET request against the `/mitigation_tasks/:id/fetch` endpoint of the reputation API.

Task ID	End State	Target Strategy	Mitigator Strategy
0	Completed	Uncooperative	Uncooperative
1			Lazy
2			Selfish
3			Rational
4	Started	Selfish	Uncooperative
5			Lazy
6			Selfish
7	Completed	Satisfied	Rational
8			Uncooperative
9			Lazy
10		Dissatisfied	Selfish
11			Rational
12			Uncooperative
13			Lazy
14			Selfish
15	Rejected		Rational

Table 5.3: Simulator in test mode: Execution order, customer configuration and end states of mitigation contracts. Completed tasks are either aborted before payment or paid out successfully. Because selfish and lazy customers never rate, tasks 5 and 6 cannot be successfully completed by either party. Compared to the lazy customer, the selfish one does upload a proof but never rates, being the reason why the end states for task 5 and 6 are different. Task 15 is the typical escalation case (marked red), where a dissatisfied target and a rational mitigator would argue about the truthfulness of the proof of service. In this case, no payment is made and end state is “rejected”.

```

1 for i in {0..29}; do
2   end_state='curl -sH "Accept:application/json" \
3     http://localhost:3000/mitigation_tasks/$i/fetch | \
4     python -c "import sys, json; print json.load(sys.stdin)[6]"`;
5
6   echo $i $end_state;
7 done

```

Listing 5.1: Shell snippet to get the mitigation contract state from the /mitigation_tasks/:id/fetch endpoint of the reputation API

To verify the reputation updates triggered in test mode, we perform a second HTTP GET request to the /customers/:customer_addr/reputation endpoint. Since this endpoint needs the customer address as first and only parameter, this address needs to be obtained from the /customers endpoint beforehand. These requests can be executed similarly to the example in Listing 5.2.

```

1 customers='curl -sH "Accept:application/json" \
2   http://localhost:3000/customers`;
3
4 for c in $customers; do
5   rep_summary='curl -sH "Accept:application/json" \
6     http://localhost:3000/customers/$c/reputation`;
7   total=$((rep_summary.positive \
8     + rep_summary.neutral \
9     + rep_summary.negative))
10  echo $c \
11    $rep_summary.positive $rep_summary.neutral $rep_summary.negative \
12    $total;

```

13 **done**

Listing 5.2: Shell snippet to obtain customer reputation from the `/customers` endpoints of the reputation API (JSON parsing omitted for better readability).

The evaluation results presented in Table 5.3 and 5.4 have been collected with such shell snippets, after the simulator script successfully completed in test mode. The results in these two tables originate from the same test run. More precisely, the requests were executed in a newly configured environment with dedicated smart contract addresses (*i.e.* new identity, reputation and mitigation smart contracts), such that the environment contained only the 16 tasks and 8 customers created by the simulator test run. The simulator runs exactly 16 tasks in test mode, which results in 2 feedbacks per task ($16 \cdot 2 = 32$ interactions). Additionally to the reputation metrics obtained from running Listing 5.2, the logs from this particular simulator instance reveal the customer strategy for each account address. This is presented in Table 5.4. The customer addresses in this table stem from this specific test run and cannot be replicated by a repeated test run. However, each simulator test creates customers with their particular strategy in exactly the same deterministic order. The reputation results after another test run in a new environment should ideally only differ in customer addresses.

For the purpose of this evaluation, the customers were either exclusively mitigators or attack targets. In production environments though, the reputation data obtained from the `/customers/:customer_addr/reputation` endpoint needs to be interpreted with special care, because it is computed using reputation claimed as mitigator as well as reputation claimed as attack target. This obfuscates the reputation metric. It is advisable to only use the `/targets/:target_addr/reputation` and `/mitigators/:mitigator_addr/reputation` when requesting a reputation summary, in order to preserve the expressiveness of the reputation metric (see Section 3.4).

Customer Address	Customer Strategy	Number of Feedbacks			Number of Interactions
		Positive	Neutral	Negative	
0x999...	Satisfied Target	1	3	0	4
0xd2d...	Selfish Mitigator	1	2	1	4
0xaae...	Rational Mitigator	1	2	1	4
0xc01...	Uncooperative Target	0	4	0	4
0xba6...	Uncooperative Mitigator	0	4	0	4
0x7d1...	Selfish Target	0	3	1	4
0x514...	Dissatisfied Target	0	3	1	4
0x362...	Lazy Mitigator	0	2	2	4
Total:		3	23	6	32

Table 5.4: Simulator in test mode: Reputation summaries and total number of interactions. The satisfied target receives one positive rating from the rational mitigator. The selfish and rational mitigator receive one negative feedback from the dissatisfied target. Since they upload proofs, they also receive one positive rating from the satisfied target. The selfish and dissatisfied target receive one negative feedback from the rational mitigator. The lazy mitigator is rated negatively twice by the satisfied and dissatisfied targets, because no proof was uploaded.

5.5 Customer Reputation in Acceptance Mode

The tests with the simulator in acceptance mode were performed using different compositions of customer strategies. The three experiments only differ in measurement duration (from one up to 10 days) and the amount of customers.

Strategy	Number of Customers for Acceptance Test Run		
	First (1 day)	Second (5 days)	Third (10 days)
Rational Mitigator	10	20	50
Satisfied Target			
Uncooperative Target	10	20	50
Selfish Target			
Dissatisfied Target			
Uncooperative Mitigator			
Lazy Mitigator			
Selfish Mitigator			
Total Number of Customers:	80	160	400

Table 5.5: Simulator in acceptance mode: Distribution of customer strategies for all three test runs, with the rational and satisfied customers as desired strategies

The Figures on the remaining pages of this section present the measured average Beta reputation by time and customer strategy for all three test runs in acceptance mode. To the end of the second test run, the experiment suffered a network outage, which explains the constant reputation values from 2 March 12AM onward in Figures 5.3, 5.4, B.3 and B.4. The machine which hosted the non-mining `geth` instance and the simulator lost connection to the miner. This made it impossible to create any more mitigation tasks. Hence, reputation stagnated.

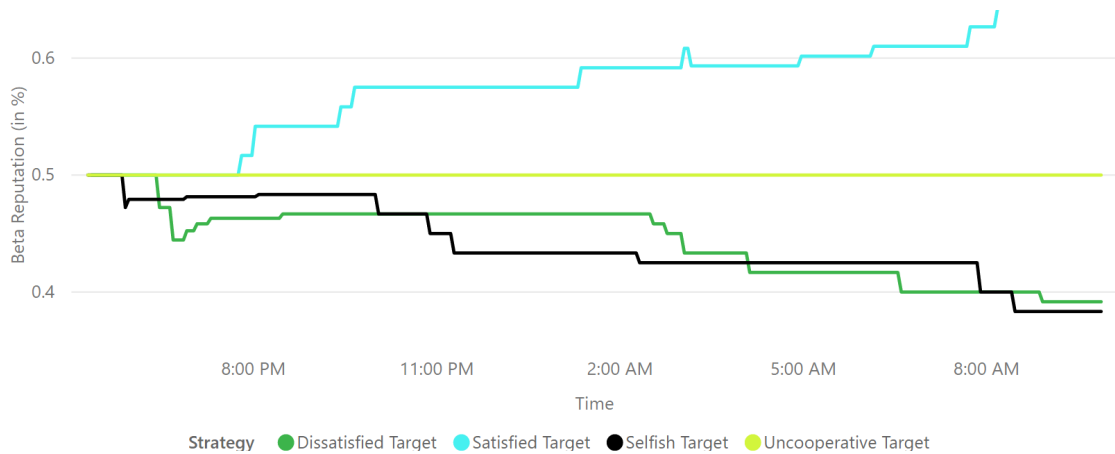


Figure 5.1: Average target Beta reputation for first test in acceptance mode, with 10 customer strategies each

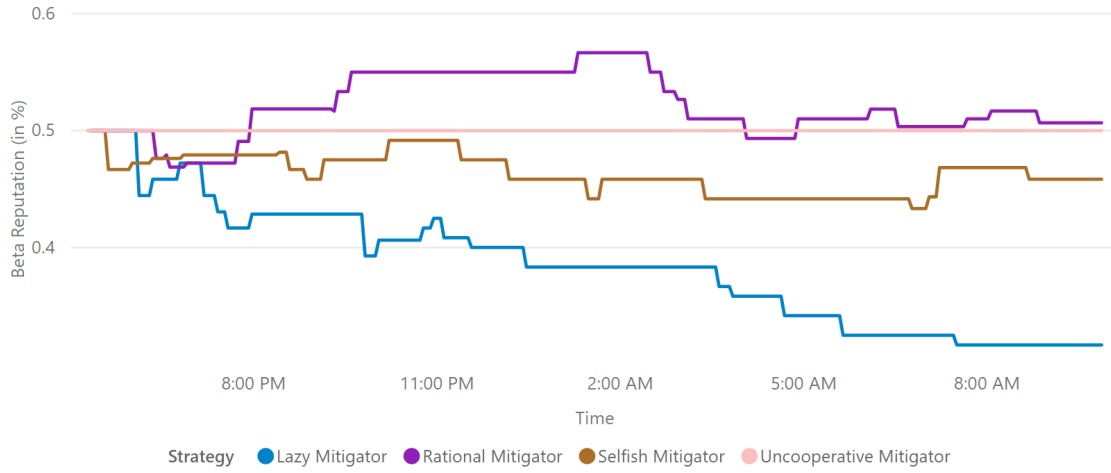


Figure 5.2: Average mitigator Beta reputation for first test in acceptance mode, with 10 customer strategies each

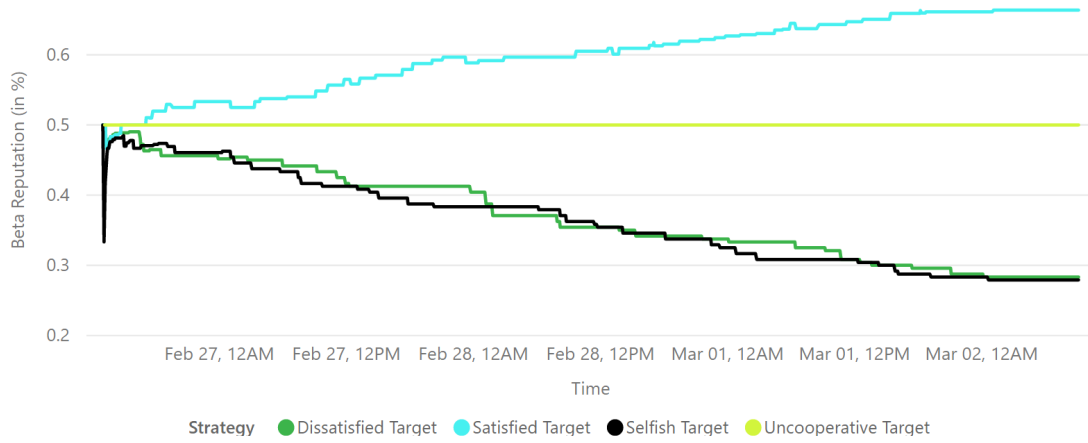


Figure 5.3: Average target Beta reputation for second test in acceptance mode, with 20 customer strategies each

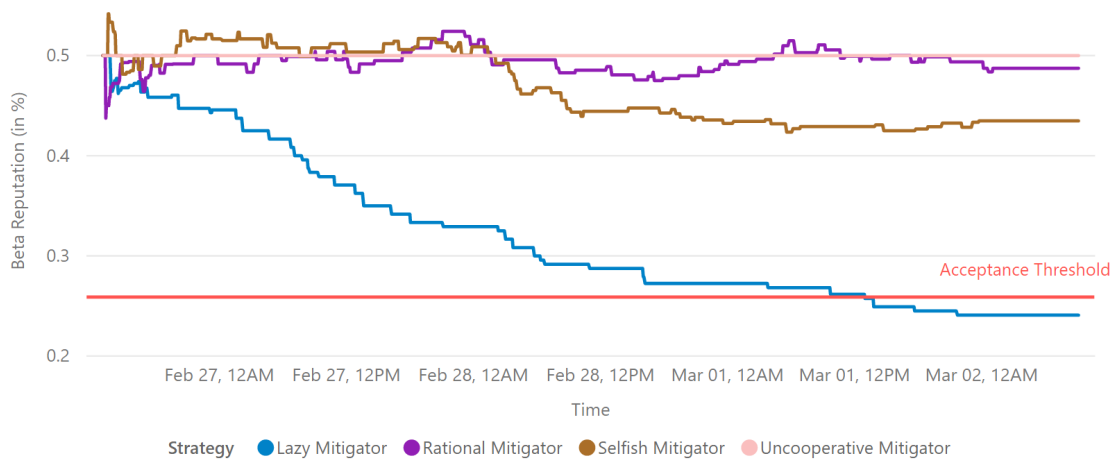


Figure 5.4: Average mitigator Beta reputation for second test in acceptance mode, with 20 customer strategies each. Customers below the acceptance threshold are disqualified from future mitigation contracts.

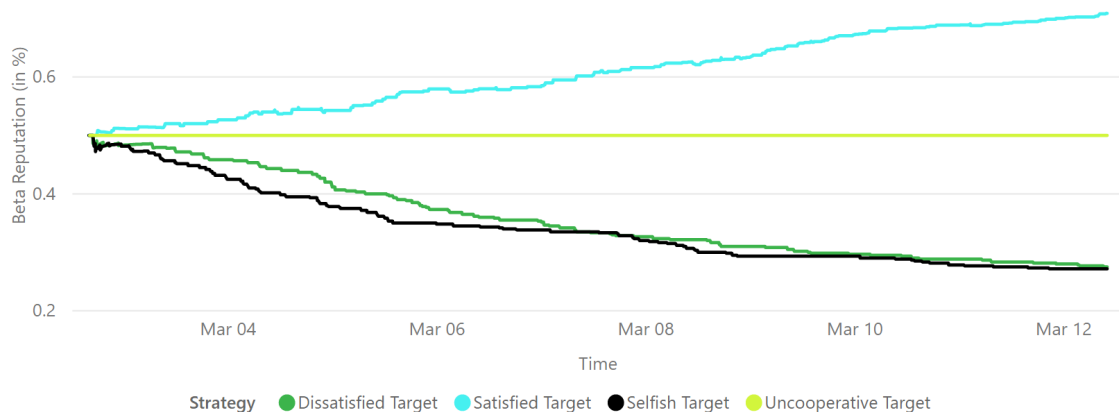


Figure 5.5: Average target Beta reputation for third test in acceptance mode, with 50 customer strategies each

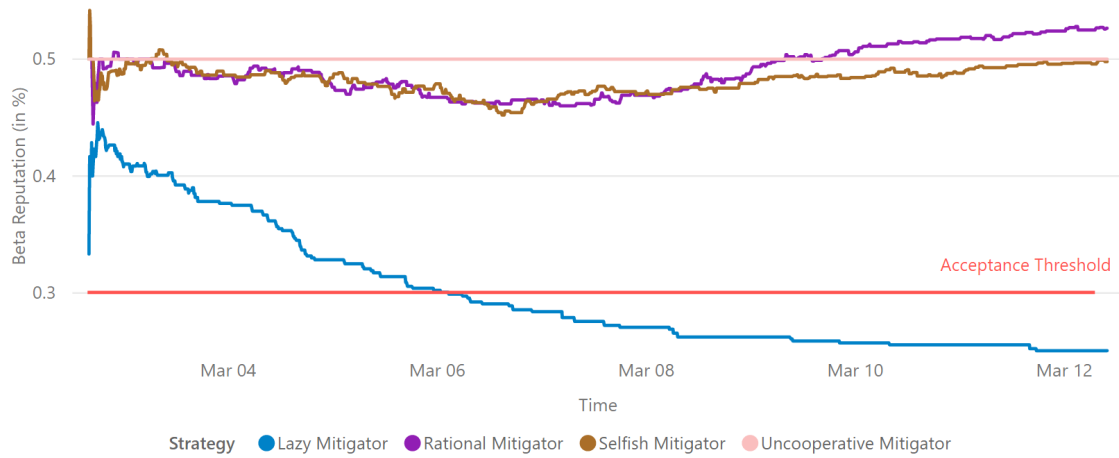


Figure 5.6: Average mitigator Beta reputation for third test in acceptance mode, with 50 customer strategies each. Customers below the acceptance threshold are disqualified from future mitigation contracts.

Chapter 6

Discussion

This chapter first discusses the evaluation results. Afterwards it investigates, how much the implemented reputation system can adhere to the desired system properties identified in the literature (see Section 2.4) and claimed during the design phase of the system (see Section 3.2). In the end of this chapter, several limitations of the implemented solution and the chosen evaluation methodology (evaluation by simulation) are mentioned.

6.1 Evaluation Results

At the time of writing, there seem to exist no concrete (or remotely comparable) implementations of the BloSS that produce real, historical transaction data. Therefore, the evaluation of the reputation and reward scheme using simulated data is limited in meaningfulness. Still, the simulated interactions between the different types of customers allow for a discussion about how reputation could evolve in the proposed system and how a customer with a particular strategy would perform.

Overall, Beta reputation scores for attack targets (T 's) allow to identify a satisfied T , because in comparison with the undesired T strategies it develops the highest average reputation value (see Figure 5.1). From Figure 5.3 it becomes evident, that dissatisfied and selfish T 's are constantly downgraded and will eventually no longer receive help from mitigators (M 's), due to their bad rating. It is not desirable in all circumstances, that satisfied T 's crowd out dissatisfied T 's, since this will incentivize T 's to accept also poorly (or even badly) delivered mitigation services. Because in the current design, no third party checks that a rating indeed matches the quality of the delivered service, this problem currently remains unsolved.

After the first few mitigation contracts, the M reputation values in Figure 5.2 also allow to clearly distinguish a constant rational M from the lazy M . However, unlike for the T 's, it is difficult to distinguish the rational M from the selfish and uncooperative M (see Figure 5.4). The uncooperative M usually shows very low positive and negative reputation (see Figure B.4), because it aborts mitigation tasks early. Hence, chances of picking an uncooperative M for a transaction is diminished the most by considering raw

reputation values (especially the amount of positive and negative ratings). Because both, the selfish and rational M upload proofs, the only difference between the two strategies is, that the selfish M never rates. Selfish M behavior is an irrational strategy, since the selfish M (deliberately or not) deprives itself of reward payments. If we assume, that M rarely forgets to rate services in the end (which is the definition of selfish behavior), we can also assume that there exist more rational than selfish M 's. This leads us to the conclusion, that chances picking a rational M compared to a selfish M with the same Beta reputation score are higher, due to the reward incentive.

Modifying the acceptance threshold of the simulator (fixed to 0.3 during the course of the experiments) might further influence the long term reputation values, because customers are excluded sooner from transactions when the acceptance threshold is higher.

6.2 Robustness against Rating Fraud

6.2.1 Sybil- and Collusion Attacks

The reputation system prototype excludes the possibility, that a customer can boost its reputation by creating mitigation contracts with itself. This prevents customers from maintaining multiple account pseudonyms on the blockchain and transacting between them to inflate reputation (Sybil attack), as long as the system is not deployed on a public ledger. Hence, an authority which is controlling access to the customer accounts in the system is required. Since trustworthy authorities give or revoke writing permissions for a consortium blockchain, these authorities regulate and manage customer data. They can be held responsible for matching real-world entities with customer accounts in order to prevent Sybil situations. The BloSS already relies on certificates to assert domain ownership [9]. A similar approach can be adopted to manage the customer accounts and pseudonyms for the reputation module. The impact of an authority on whitewashing (*i.e.* re-entry) attacks was not considered in this thesis. Unfortunately, the need for regulating authorities weakens the self-regulatory nature of the implemented reputation system.

6.2.2 Ballot Stuffing and Bad-mouthing

This system is not immune against ballot stuffing. Besides the transactions recorded on the blockchain, customers always have the possibility to agree on discounts and benefits over alternative communication channels [11]. In this situation, what might seem like altruistic behavior in the context of the presented system, was in fact arranged in a different conversation.

Contrary to ballot stuffing, the blockchain-based reputation system design impedes bad-mouthing [11]. A customer can only leave feedback for completed transactions. This elevates the cost barrier of bad-mouthing the competition, because a transaction needs to be committed for every fraudulent reputation statement [11].

6.3 Verifiability and Anonymity

As mentioned in Section 3.2 and demonstrated with examples in Section 4.7, the architecture of the prototype allows each consumer to independently verify the reputation metrics. In the current design of the systems, verifiability of the reputation metric leads to reduced anonymity. This only provides limited security for the rater. Fully anonymous reputation systems like the one developed for the decentralized autonomous marketplace Beaver [46] make use of advanced cryptographic techniques, such as ring-signatures and zero-knowledge proofs, to uphold stringent security and anonymity requirements. These techniques allow raters to stay anonymous by veiling the source of the reputation claim, but without compromising the validity of the rating (see Section 2.4).

6.4 Limitations

The baseline scenario presented in Chapter 3, where exactly two customers work together under exactly one mitigation contract and where they act in accordance to this protocol, might be too simplistic for real world scenarios. In some instances, a contract between multiple parties with additional steps during attack mitigation could be required. Still, a $1 : n$ defense situation with one attack target and n mitigators can be solved with n “ $1 : 1$ mitigation contracts”.

The blockchain used for prototyping the system relies on proof-of-work (PoW) consensus to confirm the transactions. This has the disadvantage of varying block times, which among other determinants also depends on the current difficulty of the math problem. Since the parties of a mitigation contract arrange fixed deadlines in blocks, the absolute time span in seconds varies. Employing another consensus algorithm, for example proof-of-authority (PoA), can solve this problem. No mining (*i.e.* solving complex math problems) is required in PoA systems, since blocks are appended to the chain by authorized peers only [38]. This comes with the benefit of predictable block times and thus steady time intervals (*i.e.* predictable deadlines) [38].

The evaluation with the target profiles described in Table 5.1 assumes, that there are no lazy targets T that never start a task they initiated. This would be irrational behavior, since a rational T is motivated to get the issue resolved as soon as possible. Nonetheless, there could be a T that forgets about an approved task. Such behavior is undesired, since mitigator M has no feedback until the task is either started or aborted.

The simulator assumes that a customer participates in all contracts as either M or T , but not as both. In a real world scenario, today's M can be tomorrow's T and vice-versa. The simulator was programmed in this simplified version in order to better attribute reputation scores to one customer address. For the same reason, the simulator does not contain a balanced T strategy or a disturbing [44] (camouflage [11]) customer strategy. The balanced T strategy would simulate a random draw between a satisfied or dissatisfied evaluation action. A colluding strategy which only cooperates with other colluding customers is not part of the simulator as well. Further, the assumptions about the composition of the

customer landscape and the distribution of the strategies can deviate from the situation in a proper, cooperative DDoS defense. The simulator in this work always assumes to have the same amount of customer for every strategy (see Table 5.5).

The reputation engine of the prototype works with binary feedback. Therefore, the Beta reputation is a suitable metric, because it predicts the probability of future reputable behavior based on past, binary reputation data. According to Jøsang and Ismail (2002), this metric builds “a sound mathematical basis” [5]. The metric deployed in a cooperative DDoS system might change, depending on customer needs. For this reason, the reputation engine of the prototype returns the reputation source for every reputation query. That enables customers to independently verify and recompute reputation metrics.

Chapter 7

Final Considerations

This master thesis achieved its general objective, by presenting the design, implementation and evaluation of a reputation and reward system for cooperative DDoS defense. In the **problem analysis and planning** stage, an in-depth literature review served to understand the problems, current concepts and mechanisms in reputation management. In the **requirements and system design** stage, the important requirements for the reputation and reward system were gathered and explained. The result of this stage was a detailed mitigation protocol including the processes related to reputation and reward payments. The protocol has been further refined during the implementation phase. Afterwards, the reputation and reward scheme was implemented in software by applying the relevant knowledge obtained through the preceding literature review. The **working prototype** covered the relevant requirements captured during the design phase. Finally, the evaluation results were discussed in light of the problems and challenges identified in state-of-the-art reputation systems.

Under the assumptions, that a mitigation service is delivered in a process similar to the one outlined during the thesis, the reputation system in this work allows customers to identify constant lazy, selfish and honest behavior, by looking at historical binary reputation values. The system provides these raw reputation values, such that more complicated and user-defined metrics can be computed on top. The Beta reputation used during system evaluation is one example of such a metric. Further, the raw values allow any consumer to trace back each reputation statement to its source interaction/transaction.

The system does not allow to differentiate selfish and rational mitigators by reputation itself, because their behavior in terms of service delivery is similar (see Section 6.1). However, a selfish customers is easily identified by looking at the past interactions of this customer. Added to that, the reward payment provides a suitable incentive structure which disciplines selfish customers. Mitigators will remember to apply the final service rating, because otherwise they would deprive themselves of payment.

In sum, the reputation and reward scheme of this thesis can prevent free-riding (attack targets) and false-reporting (mitigators) in the long run by incentivizing rational behavior. However, it remains to show the effectiveness of the approach in practice, since this could only be evaluated by simulation so far. The deployment of the reputation in a real-world

scenario might lead to usability problems, one being predictable deadlines, as described in Section 6.4. Also, in the current design the advantage of a transparent metric is offset by decreased anonymity (of the rater).

The development of the simulator script was accompanied by several obstacles. First, the JavaScript event-driven programming paradigm lead to several timing issues. It was necessary to wait for events and transaction receipts, in order to continue with consecutive actions. The simulator script was developed in several iterations. Some changes led to unexpected issues further down the road, but could be fixed in later iterations. Sometimes, the simulator crashed due to accidentally stopped (or not yet started) services and peers running out of funds. Eventually, all timing issues could be fixed and the simulator script was running reliably.

7.1 Future Work

The price of the mitigation contracts was fixed to one Ether for all experiments. Intuitively, a rating for a more valuable transaction might be of variable importance in the real world (depending on the mitigator). Because the price of a mitigation task remained fixed during the course of the evaluation, this thesis did not exercise any profit analysis or optimizations thereof.

Only the attack target can measure the quality or effectiveness of a mitigation service. If the attack target can be assured, that the network configuration in the domain of the mitigator was adjusted to its liking, the target is responsible to provided the correct configuration file(s). In this case, the mitigation system is only required to guarantee the integrity of the applied configuration file(s). Further investigations are needed, to design a secure deployment process, that reliably applies the network configuration in the domain of the attack origin.

The system built during this thesis trades reputation verifiability against customer anonymity, since for every transaction the reputation claim and owner are published in plain text on IPFS. A verifiable but untraceable reputation system, as leveraged in the Beaver marketplace (see Section 3.2), can guarantee validity of each rating or review without explicitly linking the rating and the transaction. Ideally, it should not be possible to infer the owner of the reputation statement (*i.e.* the rater) from the reputon for a particular mitigation service. vice-versa, knowing the owner of the reputation statement, the content of a particular rating should remain unknown. Nevertheless, the reputation system should only store valid ratings.

Finally, as mentioned in Section 5.3, the system still lacks an escalation process to dispute ratings and rewards.

Bibliography

- [1] Akamai. *How to Protect Against DDoS Attacks - Stop Denial of Service*. 2016. URL: <https://www.akamai.com/us/en/resources/protect-against-ddos-attacks.jsp> (visited on Dec. 4, 2017).
- [2] Akamai. *Q3 2017 State of the Internet Security Report*. Tech. rep. 2017.
- [3] Alex Beregszaszi and Thomas Bertani. *Iudex*. URL: <https://github.com/iudex> (visited on Nov. 3, 2017).
- [4] Alex Rea, Aron Fischer, and Jack du Rose. *Colony Technical White Paper*. Tech. rep. 2017.
- [5] Audun Josang and Roslan Ismail. “The Beta Reputation System”. In: *Proceedings of the 15th Bled Electronic Commerce Conference*. Vol. 5. 2002, pp. 2502–2511.
- [6] *Backfeed: A Public Objective Non-Economic Reputation System*. Nov. 2017. URL: <https://github.com/Backfeed/backfeed>.
- [7] *Bitrated Developer API*. URL: <https://www.bitrated.com/docs> (visited on Nov. 5, 2017).
- [8] T. Bocek et al. “Game Theoretical Analysis of Incentives for Large-Scale, Fully Decentralized Collaboration Networks”. In: *2008 IEEE International Symposium on Parallel and Distributed Processing*. Apr. 2008, pp. 1–8.
- [9] Bruno Rodrigues et al. “A Blockchain-Based Architecture for Collaborative DDoS Mitigation with Smart Contracts”. In: *Security of Networks and Services in an All-Connected World*. Vol. 10356. 2017, pp. 16–29.
- [10] Matthew Buechler et al. *Decentralized Reputation System for Transaction Networks*. Tech. rep. Dept. of CIS - Senior Design, University of Pennsylvania, Philadelphia, PA, 2015.
- [11] Yuanfeng Cai and Dan Zhu. “Fraud Detections for Online Businesses: A Perspective from Blockchain Technology”. In: *Financial Innovation* 2.1 (Dec. 2016), p. 20.
- [12] Davide Carboni. “Feedback Based Reputation on Top of the Bitcoin Blockchain”. In: abs/1502.01504 (Feb. 2015).
- [13] Christopher Georgen. *Topl, Empowering Growth by Enabling Investment*. 2017. URL: <https://github.com/Topl/whitepaper/blob/master/Whitepaper.pdf> (visited on Nov. 27, 2017).
- [14] CloudFlare. *CloudFlare Advanced DDoS Protection*. 2016. URL: <https://www.cloudflare.com/static/media/pdf/cloudflare-whitepaper-ddos.pdf> (visited on Dec. 4, 2017).
- [15] Dd B. DeFigueiredo and Earl T. Barr. “Trustdavis: A Non-Exploitable Online Reputation System”. In: *E-Commerce Technology, 2005. CEC 2005. Seventh IEEE International Conference On*. 2005, pp. 274–283.

- [16] Sergi Delgado-Segura, Cristian Tanas, and Jordi Herrera-Joancomartí. “Reputation and Reward: Two Sides of the Same Bitcoin”. In: *Sensors* 16.6 (May 2016), p. 776.
- [17] Mieso K. Denko. “Detection and Prevention of Denial of Service (DoS) Attacks in Mobile Ad Hoc Networks Using Reputation-Based Incentive Scheme”. In: *Journal of Systemics, Cybernetics and Informatics* 3.4 (2005), pp. 1–9.
- [18] R. Dennis and G. Owen. “Rep on the Block: A next Generation Reputation System Based on the Blockchain”. In: *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*. Dec. 2015, pp. 131–138.
- [19] Dr. Christian Lundkvist et al. *Uport: A Platform for Self-Sovereign Identity*. Tech. rep. 2017.
- [20] F. Randall Farmer and Bryce Glass. *Building Web Reputation Systems*. 1st ed. 2010.
- [21] Giorgos Zacharia. “Collaborative Reputation Mechanisms for Online Communities”. PhD thesis. Massachusetts Institute of Technology, 1999.
- [22] K. Giotis, M. Apostolaki, and V. Maglaris. “A Reputation-Based Collaborative Schema for the Mitigation of Distributed Attacks in SDN Domains”. In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. Apr. 2016, pp. 495–501.
- [23] T. Dong Huynh, Nicholas R. Jennings, and Nigel R. Shadbolt. “FIRE: An Integrated Trust and Reputation Model for Open Multi-Agent Systems”. In: *Proceedings of the 16th European Conference on Artificial Intelligence*. ECAI’04. 2004, pp. 23–27.
- [24] IETF. *Distributed Denial of Service (DDoS) Open Threat Signaling Requirements*. 2017. URL: <https://datatracker.ietf.org/wg/dots> (visited on Nov. 23, 2017).
- [25] Audun Jøsang, Roslan Ismail, and Colin Boyd. “A Survey of Trust and Reputation Systems for Online Service Provision”. In: *Decision Support Systems*. Emerging Issues in Collaborative Commerce 43.2 (Mar. 2007), pp. 618–644.
- [26] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. “The Eigen-trust Algorithm for Reputation Management in P2p Networks”. In: *Proceedings of the 12th International Conference on World Wide Web*. 2003, pp. 640–651.
- [27] Rizwan Khan and Avimanyou Vatsa. *Detection and Control of DDOS Attacks over Reputation and Score Based MANET*. Nov. 2017.
- [28] Matt Kindy. *Divine: A Blockchain Reputation System For Determining Good Market Actors*. June 2017. URL: <https://medium.com/top1-blog/divine-a-blockchain-reputation-system-for-determining-good-market-actors-7c47a0308ae8> (visited on Nov. 22, 2017).
- [29] H. Kopp et al. “Design of a Privacy-Preserving Decentralized File Storage with Financial Incentives”. In: *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*. Apr. 2017, pp. 14–22.
- [30] Murray Kucherawy and Nathaniel Borenstein. *A Media Type for Reputation Interchange*. URL: <https://tools.ietf.org/html/rfc7071> (visited on Nov. 3, 2017).
- [31] Orfeas Stefanos Thyfronitis Litos and Dionysis Zindros. *Trust Is Risk: A Decentralized Financial Trust Platform*. Tech. rep. 156. 2017.
- [32] Jian Liu et al. “Towards Fairness of Cryptocurrency Payments”. In: *arXiv:1609.07256 [cs]* (Sept. 2016).

- [33] Yuan Liu et al. *An Identity Management System Based on Blockchain*. URL: <https://www.ucalgary.ca/pst2017/files/pst2017/paper-8.pdf> (visited on Oct. 8, 2017).
- [34] mdelias. *The Precedent Protocol Whitepaper*. Oct. 2017. URL: <https://github.com/mdelias/precedent> (visited on Dec. 21, 2017).
- [35] Axel Moinet, Benoît Darties, and Jean-Luc Baril. “Blockchain Based Trust & Authentication for Decentralized Sensor Networks”. In: *arXiv:1706.01730 [cs]* (June 2017).
- [36] Andrew Mortensen, Tirumaleswar Reddy, and Robert Moskowitz. *Distributed Denial of Service (DDoS) Open Threat Signaling Requirements*. URL: <https://tools.ietf.org/html/draft-ietf-dots-requirements-07> (visited on Dec. 4, 2017).
- [37] *NEM: Technical Reference*. Tech. rep. 2015.
- [38] *Parity, Proof of Authority Chains*. URL: <https://paritytech.github.io/wiki/Proof-of-Authority-Chains.html> (visited on Feb. 12, 2018).
- [39] Paul Sztorc. *Truthcoin Whitepaper*. 2015. URL: <http://www.truthcoin.info/papers/truthcoin-whitepaper.pdf> (visited on July 2, 2018).
- [40] Jack Peterson and Joseph Krug. “Augur: A Decentralized, Open-Source Platform for Prediction Markets”. In: *arXiv preprint arXiv:1501.01042* (2015).
- [41] Bruno Rodrigues, Thomas Bocek, and Burkhard Stiller. “Enabling a Cooperative, Multi-Domain DDoS Defense by a Blockchain Signaling System (BloSS)”. In: *Semantic Scholar* (2017).
- [42] Jordi Sabater and Carles Sierra. “REGRET: Reputation in Gregarious Societies”. In: *Proceedings of the Fifth International Conference on Autonomous Agents*. AGENTS ’01. 2001, pp. 194–195.
- [43] Alexander Schaub et al. “A Trustless Privacy-Preserving Reputation System”. In: *ICT Systems Security and Privacy Protection*. IFIP Advances in Information and Communication Technology. May 2016, pp. 398–411.
- [44] Andreas Schlosser, Marco Voss, and Lars Brückner. “Comparing and Evaluating Metrics for Reputation Systems by Simulation”. In: *Proceedings of the IEEE Workshop on Reputation in Agent Societies*. 2004.
- [45] Mike Sharples and John Domingue. “The Blockchain and Kudos: A Distributed System for Educational Record, Reputation and Reward”. In: *Adaptive and Adaptable Learning*. Lecture Notes in Computer Science. Sept. 2016, pp. 490–496.
- [46] Kyle Soska et al. “Beaver: A Decentralized Anonymous Marketplace with Secure Reputation.” In: *IACR Cryptology ePrint Archive 2016* (2016), p. 464.
- [47] Melanie Swan. *Blockchain: Blueprint for a New Economy*. Jan. 2015.
- [48] Y. Wang and J. Vassileva. “Trust and Reputation Model in Peer-to-Peer Networks”. In: *Proceedings Third International Conference on Peer-to-Peer Computing (P2P2003)*. Sept. 2003, pp. 150–157.
- [49] *Work.Nation: Decentralized Skill Attestations Using uPort, Ethereum and IPFS*. Oct. 2017. URL: <https://github.com/worknation/work.nation>.
- [50] Li Xiong and Ling Liu. “Building Trust in Decentralized Peer-to-Peer Electronic Communities”. In: *International Conference on Electronic Commerce Research (ICECR-5)* (Feb. 2003).

- [51] Harlan Yu, J. Rexford, and E. W. Felten. “A Distributed Reputation Approach to Cooperative Internet Routing Protection”. In: *1st IEEE ICNP Workshop on Secure Network Protocols, 2005. (NPSec)*. Nov. 2005, pp. 73–78.
- [52] T. Yu et al. “INDAPSON: An Incentive Data Plan Sharing System Based on Self-Organizing Network”. In: *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. Apr. 2014, pp. 1545–1553.
- [53] Yu Zhang and Mihaela van der Schaar. “Reputation-Based Incentive Protocols in Crowdsourcing Applications”. In: *arXiv:1108.2096 [physics]* (Aug. 2011).
- [54] Zibin Zheng et al. “Blockchain Challenges and Opportunities: A Survey”. In: *Work Pap* (2016).

Abbreviations

BloSS	Blockchain Signaling System
API	Application Programming Interface
CLI	Command line interface
CLNY	Colony token
CPU	Core processing unit
DAM	Decentralized Anonymous Marketplace
GiB	Gibibyte, 1024^3 bytes
GHz	Gigahertz
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IPFS	InterPlanetary File System
IP	Internet protocol
ISP	Internet Service Provider
JSON	JavaScript Object Notation
MAD	Mutual assured destruction
MANET	Mobile ad hoc Network
OFE	Optimistic fair exchange
PoA	Proof-of-authority
PoW	Proof-of-work
P2P	Peer-to-Peer
RAM	Random-access memory
RoR	Ruby on Rails
SDN	Software-defined network
TTP	Trusted third party
VTC	VoteCoin

Glossary

BloSS The Blockchain Signaling System (BloSS) proposed by Rodrigues *et al.* (2017) in [41] is “a novel approach deploying hardware to simplify the signaling of DDoS attacks in a cooperative network defense system” (p. 1).

DDoS As defined in [36] by Mortensen *et al.* (2017), DDoS is “a distributed denial-of-service attack, in which traffic originating from multiple sources are directed at a target on a network. DDoS attacks are intended to cause a negative impact on the availability of servers, services, applications, and/or other functionality of an attack target” (p. 3).

DDoS attack target According to Mortensen *et al.* (2017) in [36], the DDoS attack target is “a network connected entity with a finite set of resources, such as network bandwidth, memory or CPU, that is the focus of a DDoS attack” (p. 3).

DDoS attack telemetry Mortensen *et al.* (2017) define DDoS attack telemetry in [36] as “collected measurements and behavioral characteristics defining the nature of a DDoS attack” (p. 4).

Fairness Property of an exchange protocol. According to Liu *et al.* (2016) in [32], strong fairness is when “at the time of protocol termination, either both players get what they want, or neither of them does” (p. 4) and weak fairness is “where strong fairness cannot be achieved, but an honest player can prove to an (external) arbiter that the other player has received (or can still receive) the item the latter expects” (p. 4).

Incentive (mechanism) The online oxford dictionary¹ defines an incentive as “a thing that motivates or encourages someone to do something”, or as “a payment or concession to stimulate greater output or investment”.

Mitigation In [36], Mortensen *et al.* (2017) define mitigation as “a set of countermeasures enforced against traffic destined for the target or targets of a detected or reported DDoS attack, where countermeasure enforcement is managed by an entity in the network path between attack sources and the attack target” (p. 4).

Mitigator “An entity, typically a network element, capable of performing mitigation of a detected or reported DDoS attack”, as defined in [36] by Mortensen *et al.* (2017, p. 4).

¹<https://en.oxforddictionaries.com>

Mitigation contract A contract which specifies the service agreement between DDoS attack target and a mitigator.

Mitigation payout A proportional share of the mitigation price which serves as monetary reward for the mitigator.

Mitigation price The price for the mitigation service as defined in the mitigation contract.

Mitigation request A request for a service contract agreement between the DDoS attack target and a mitigator.

Mitigation service The sum of all counteractions taken by the mitigator to fulfill his duties as specified in the mitigation contract.

Proof of service/blocking An artifact which proves, that a mitigation service has been provided.

Reputation In Schlosser *et al.* (2004), reputation is defined as “the collected and processed information about one entity’s former behavior as experienced by others” (p. 1). Jøsang *et al.* (2007) use in [25] the Oxford dictionary’s definition in which “reputation is what is generally said or believed about a person’s or thing’s character or standing” (p. 620).

Timeliness Property of an exchange protocol. Liu *et al.* (2016) define a protocol in [32] timely if “honest players can be certain that the protocol will be completed at a certain finite point in time. At completion of the protocol, the state of the exchange is final from that player’s perspective” (p. 4).

Trust Schlosser *et al.* (2004) define trust in [44] as “a measure of willingness to proceed with an action (decision) which places parties (entities) at risk of harm and is based on an assessment of the risks, rewards and reputations associated with all the parties involved in a given situation” (p. 1).

List of Figures

1.1	Social dilemma of false-reporting and free-riding	2
2.1	Threshold reputation model for crowdsourcing applications	10
2.2	MAD transaction	11
2.3	Optimistic fair exchange with TTP	12
2.4	Anonymous feedback with blind signatures	13
2.5	Cosigned voucher as incentive to vote	14
2.6	Rating fraud	19
2.7	Storing reputons on- and off-chain	21
3.1	DDoS mitigation service process with payment and reputation rating . . .	24
3.2	Payout process	26
3.3	System flowchart	28
4.1	Mitigation Protocol	34
5.1	Average target Beta reputation for first test in acceptance mode	47
5.2	Average mitigator Beta reputation for first test in acceptance mode	48
5.3	Average target Beta reputation for second test in acceptance mode	48
5.4	Average mitigator Beta reputation for second test in acceptance mode . . .	48
5.5	Average target Beta reputation for third test in acceptance mode	49
5.6	Average mitigator Beta reputation for third test in acceptance mode	49
A.1	Reputation API	67

B.1	Average target reputation for first test in acceptance mode	70
B.2	Average mitigator reputation for first test in acceptance mode	71
B.3	Average target reputation for second test in acceptance mode	72
B.4	Average mitigator reputation for second test in acceptance mode	73
B.5	Average target reputation for third test in acceptance mode	74
B.6	Average mitigator reputation for third test in acceptance mode	75

List of Tables

2.1	Literature research overview	7
2.2	Comparison of flow engines	16
2.3	Prisoners dilemma payoff matrix	22
2.4	DDoS attack mitigation scenarios	22
4.1	Mitigation contract states	33
4.2	Docker containers used to develop and test the prototype	41
5.1	Attack target (T) strategies	44
5.2	Mitigator (M) strategies	44
5.3	Simulator: Task configurations and end states in test mode	45
5.4	Simulator: Reputation results in test mode	46
5.5	Distribution of customer strategies for simulations in acceptance mode	47
A.1	Snapshot of Solidity test coverage	68

Appendix A

Implementation

A.1 Documentation of the Reputation API

GET /customers	Show all targets and mitigators
GET /customers/:customer_addr	Show customer
GET /customers/:customer_addr/fetch	Fetch customer from blockchain
GET /customers/:customer_addr/reputons	Show reputation claims about customer
GET /customers/:customer_addr/reputation	Show customer reputation
GET /infos	Show server infos
GET /mitigation_tasks	Show all mitigation contracts
GET /mitigation_tasks/:id	Show mitigation contract
GET /mitigation_tasks/:id/fetch	Fetch mitigation contract from blockchain
GET /mitigators	Show all mitigators
GET /mitigators/:mitigator_addr	Show tasks of mitigator
GET /mitigators/:mitigator_addr/fetch	Fetch mitigator from blockchain
GET /mitigators/:mitigator_addr/reputons	Show reputation claims about mitigator
GET /mitigators/:mitigator_addr/reputation	Show reputation for mitigator
GET /targets	Show all attack targets
GET /targets/:target_addr	Show tasks of attack target
GET /targets/:target_addr/fetch	Fetch target from blockchain
GET /targets/:target_addr/reputons	Show reputation claims about attack target
GET /targets/:target_addr/reputation	Show reputation for attack target

Figure A.1: Excerpt from the API documentation

A.2 Test Coverage and Continuous Integration

To ensure the correct behavior of the prototype at run time, the smart contracts were extensively tested. An overview about the unit test coverage results is given in Table A.1.

File	Statements		Branches		Functions		Lines	
Identity.sol	100%	5/5	100%	2/2	100%	4/4	100%	7/7
Mitigation.sol	88.57%	62/70	74.19%	46/62	88.89%	24/27	95.45%	63/66
Reputation.sol	96.55%	28/29	71.88%	23/32	100%	6/6	96.77%	30/31
Overall:	91.35%	95/104	73.96%	71/96	91.89%	34/37	96.15%	100/104

Table A.1: Solidity test coverage snapshot from 22 February 2018

Appendix B

Evaluation Results

This section contains additional evaluation results. These are the positive, negative and neutral reputation values by time and customer strategy for all experiments of the evaluation in Chapter 5.

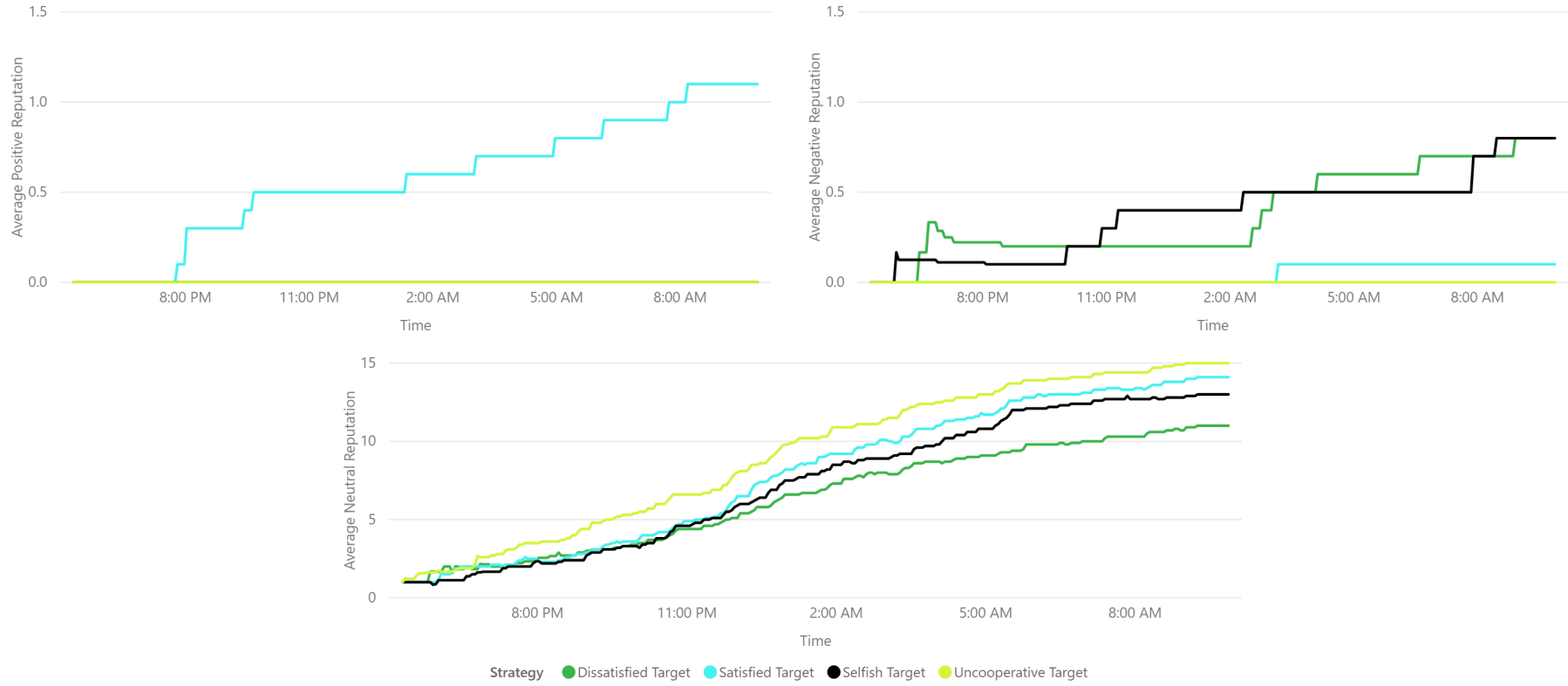


Figure B.1: Average positive, negative and neutral reputation by time and target strategy in the first acceptance mode test

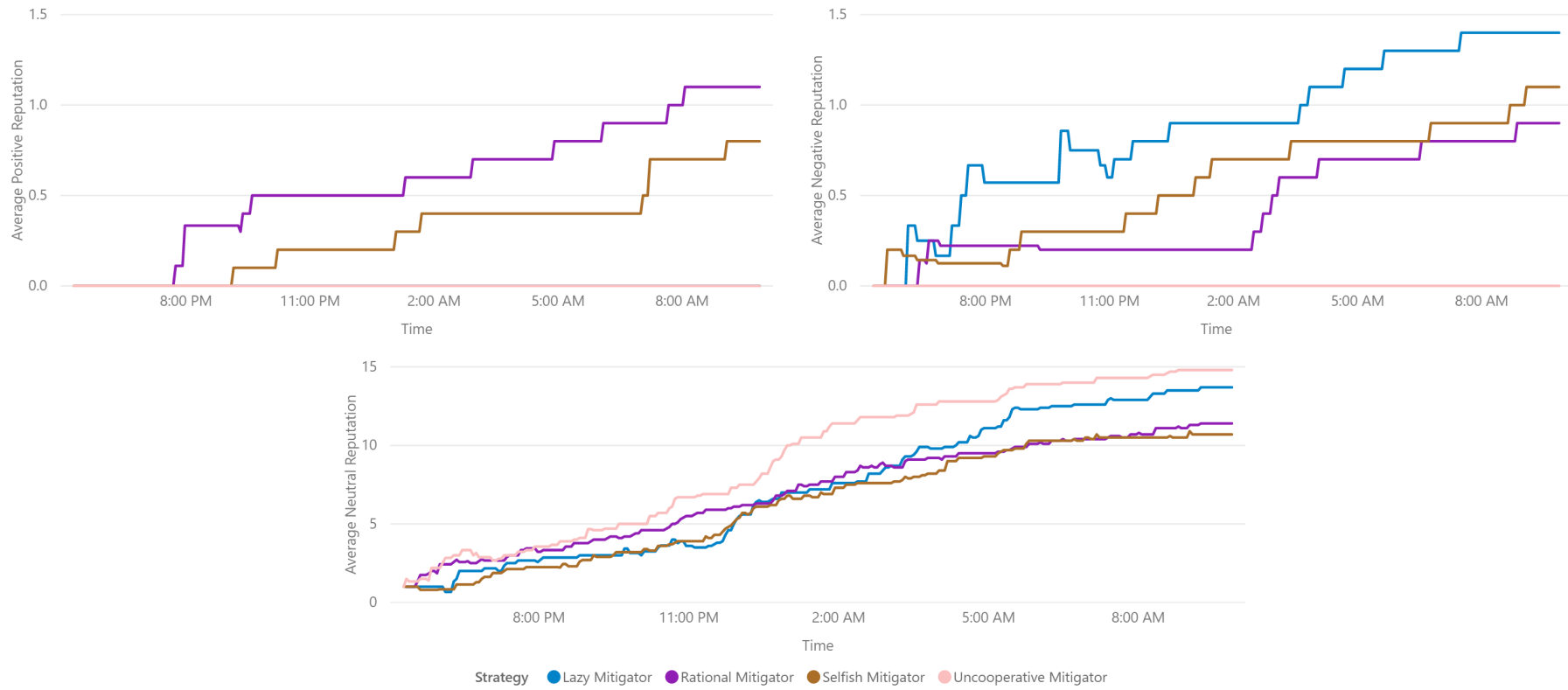


Figure B.2: Average positive, negative and neutral reputation by time and mitigator strategy in the first acceptance mode test

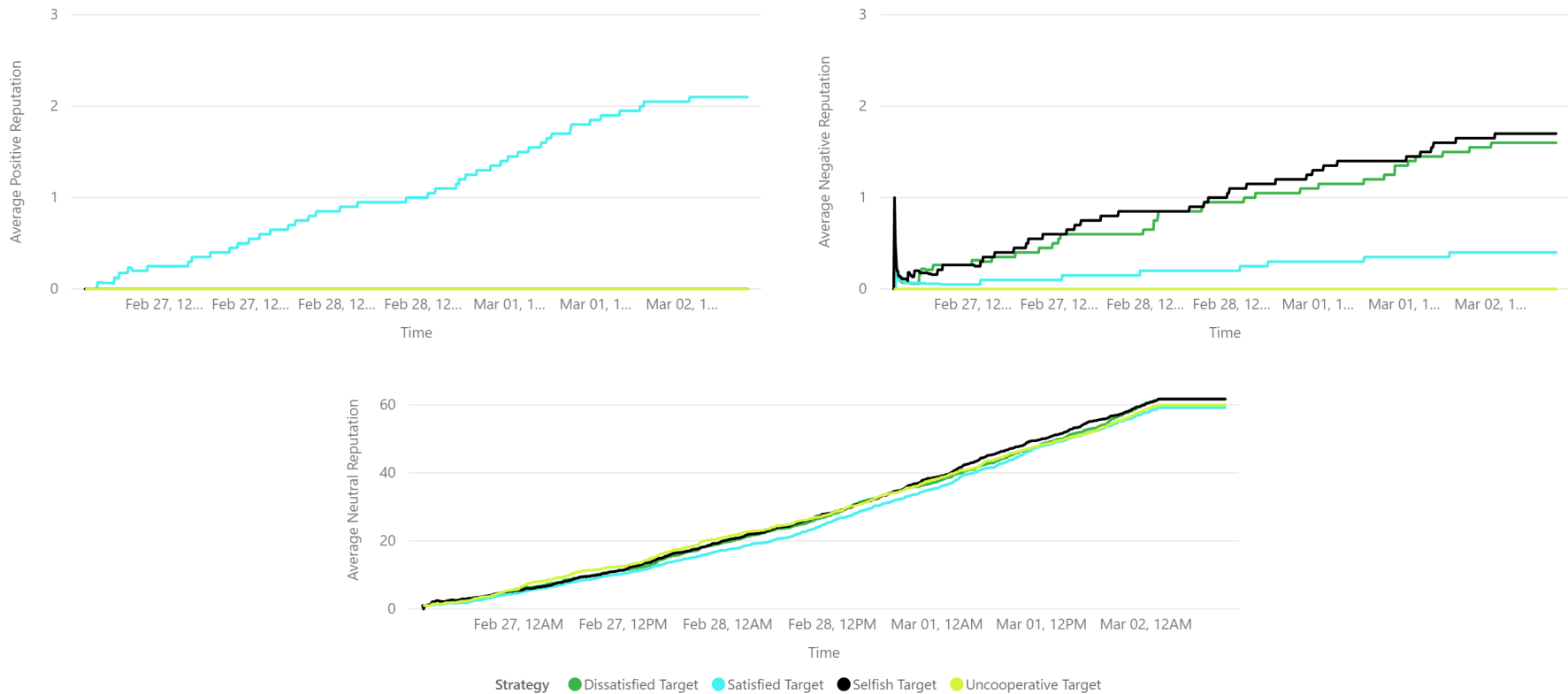


Figure B.3: Average positive, negative and neutral reputation by time and target strategy in the second acceptance mode test

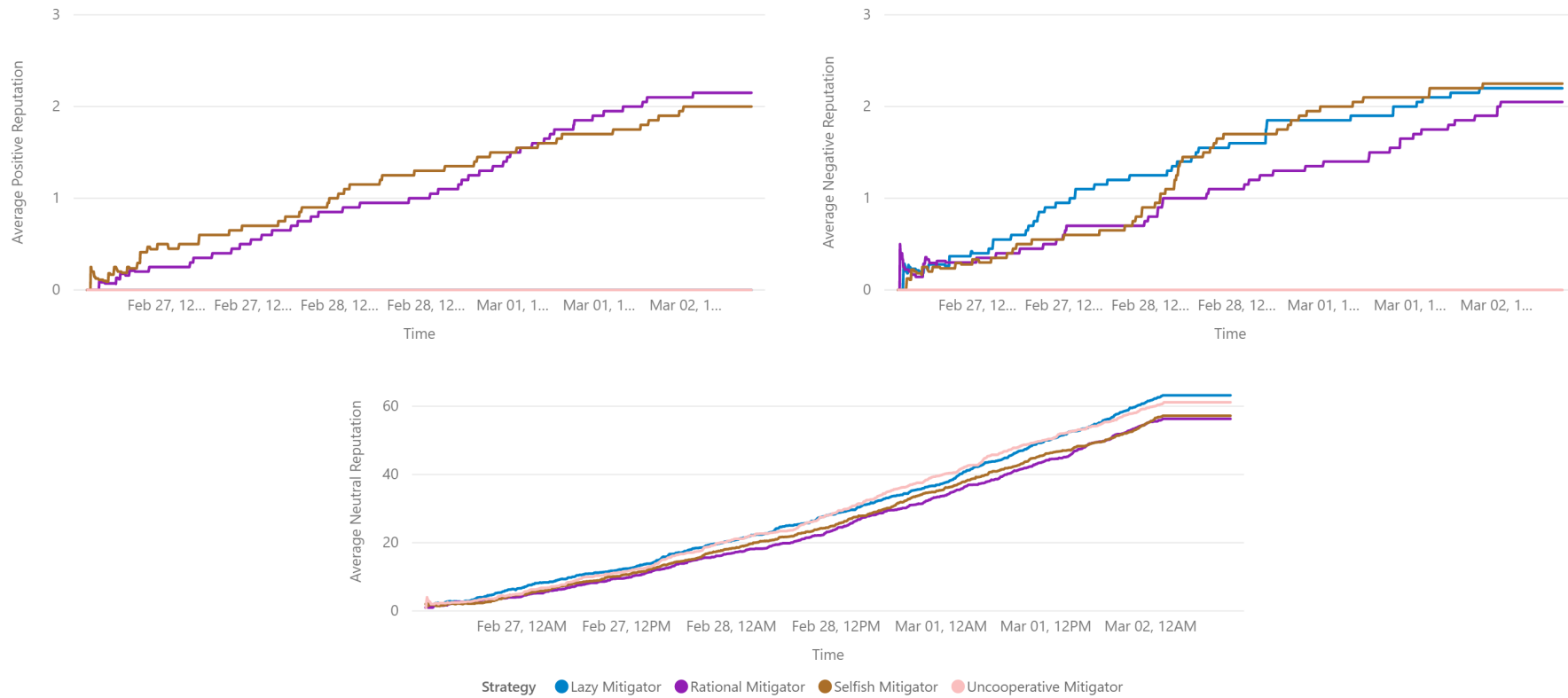


Figure B.4: Average positive, negative and neutral reputation by time and mitigator strategy in the second acceptance mode test

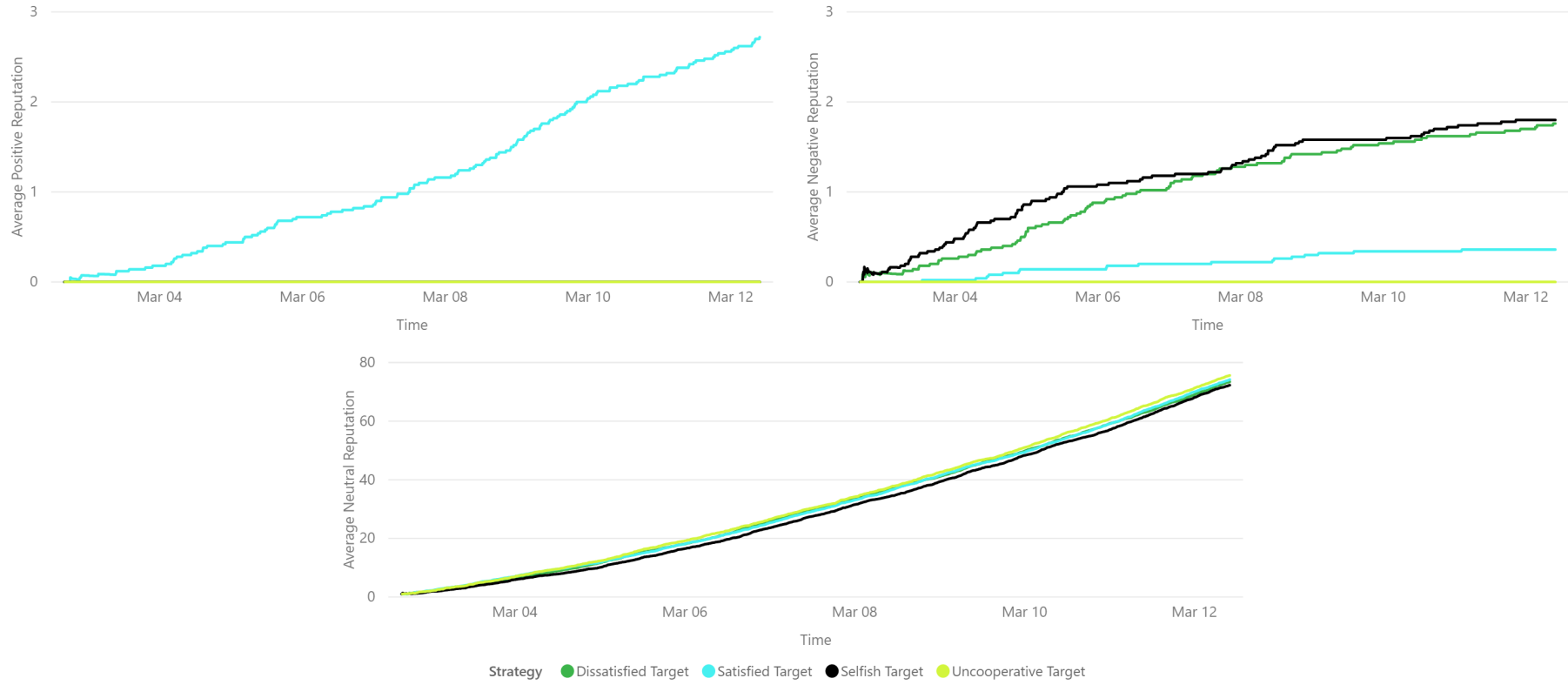


Figure B.5: Average positive, negative and neutral reputation by time and target strategy in the third acceptance mode test

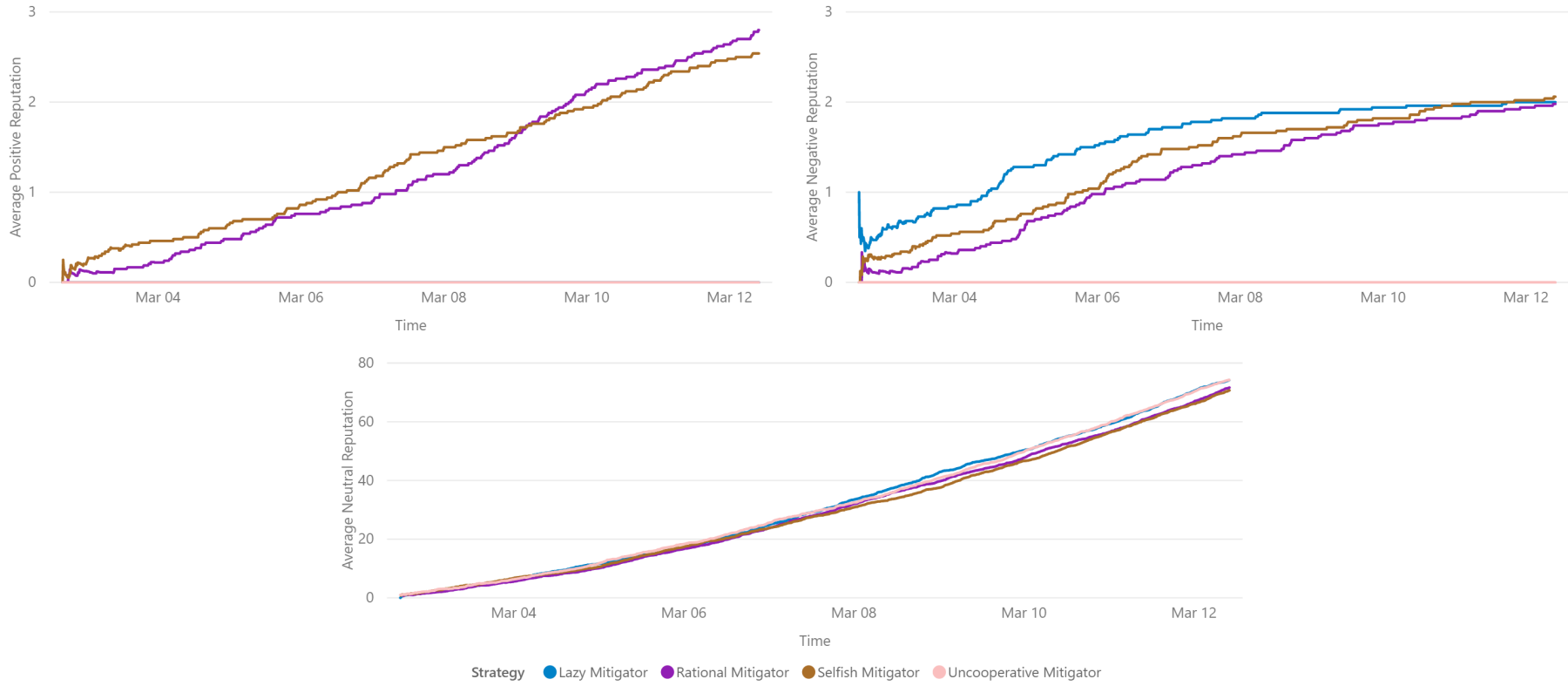


Figure B.6: Average positive, negative and neutral reputation by time and mitigator strategy in the third acceptance mode test

Appendix C

Contents of the CD

The content of the enclosed CD is structured as follows:

- The **API-Doc** folder holds the documentation of the Reputation API endpoints (see also Figure A.1).
- The **Background** folder contains mindmaps used for the literature review.
- The **Code** folder contains all code including the final prototype (see Table 4.2), a test coverage snapshot (see Table A.1) and an early-stage simulator that simulates different reputation engines using IPython notebooks. The **README.html** file contains instructions on how to recreate and use the development environment.
- The evaluation results and logs of the final simulator (final prototype) are enclosed in the **Evaluation** folder.
- The **Thesis.TEX** folder holds an electronic copy of this thesis including all figure source files (svg).
- Additionally, the root folder contains the thesis proposal, the planning sheet (time-line), the mid-term presentation and spreadsheet artifacts from the design stage.

```
|-- API-Doc_20180205
|-- Background
|-- Code
|   |-- Final_Prototype
|   |   '-- README.html
|   |-- Python_Simulator
|   '-- Solidity_Coverage_20180222
|-- Evaluation_Results_20180306
|-- MA-A-Gruhler-v4.pdf
|-- MasterThesis_MidTermPresentation.pdf
|-- MsThesisPlanning_20170927.pdf
|-- reputation_reward_design_20170929.ods
|-- reputation_reward_design_20171012.ods
'-- Thesis_TEX
```

Listing C.1: Overview about the contents of the enclosed CD