Organizational Modeling and Simulation of Blockchain Systems

A case study on blockchain vulnerabilities using Multi-Agent Reinforcement Learning

Hector ROUSSILLE

PhD Thesis

Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier

Supervisors:
Dr. Fabien MICHEL ..... Maître de conférences HDR, Université de Montpellier
Dr. Önder GÜRCAN ....Chercheur Senior, NORCE (ex CEA LIST)

Université de Montpellier
Department of Computer Science
2024

Declaration of Authorship

I, Hector ROUSSILLE, declare that this thesis titled, "Organizational Modeling and Simulation of Blockchain Systems" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

UNIVERSITÉ DE MONTPELLIER

Abstract

CEA List - Paris Saclay University
LIRMM - Montpellier University

Doctor of Computer Science

Organizational Modeling and Simulation of Blockchain Systems

by Hector ROUSSILLE

Blockchain technology has gained substantial traction in recent years, revolutionizing industries through its decentralized and trustless nature. However, the security of blockchain systems and, specifically, their underlying incentive mechanisms remains a critical concern, potentially leading to catastrophic losses if those economic incentives shaping the behavior of rational participants are not aligned with the expected behavior. This thesis addresses this challenge by proposing a comprehensive framework that ultimately leverages MARL to enhance the security of blockchain incentives while not being strictly limited to it. We propose a generic blockchain model that encapsulates the core components of blockchain systems, making it flexible and easily adaptable to diverse blockchain designs. This model serves as a foundational framework for enhancing blockchain security. Based on this model, we define a taxonomy of incentive vulnerabilities in blockchain systems. This classification categorizes, ranks and prioritizes vulnerabilities based on their feasibility and network impact. The taxonomy aids in identifying critical areas of interest where automatic and potentially exploratory work might be required to assess the resilience of the system. To complete the framework, we introduce a blockchain simulator that is, by construction, as close as possible to the generic blockchain model but, it is also exact with respect to specific protocols, and, compatible with reinforcement learning, allowing us to replicate real-world scenarios using honest or byzantine agents with arbitrary objectives. The same simulator can then be used to assess the effectiveness of a given countermeasure. Finally, we show a concrete study of protocol vulnerability of Ethereum 2.0 using all of the above contributions, serving as a practical example of the usage of MARL in the context of the proposed framework. By integrating these contributions, this thesis contributes to the evolving field of blockchain security and provides the means for developers and researchers to analyze, identify and address incentive vulnerabilities in a standardized and systematic manner. Using MARL as a security enhancement tool offers promising results, paving the way for more robust and secure blockchain systems in the future.

Contents

Declaration of Authorship iii

Abstract vii

Résumé 1

1 Introduction 9
1.1 Context 9
1.2 AI in blockchain systems 9
1.3 Thesis Overview 10
1.3.1 Contributions 10
1.3.2 Document Outline 11

2 Background 13
2.1 Distributed Systems 13
2.2 Consensus 14
2.2.1 Consensus Algorithms 14
2.2.2 Challenges and Trade-Offs 14
2.3 Blockchain Systems 15
2.3.1 Blockchain Data Structure 15
2.3.2 Decentralized Applications 17
2.3.3 Oracle in Blockchain Systems 17
2.3.4 Permission Models in Blockchain 17
2.3.5 Consensus in Blockchain Systems 18
2.3.6 Incentives 20
2.4 Reinforcement Learning (RL) 21
2.4.1 Markov Decision Process (MDP) 21
2.4.2 Estimating the expected rewards with a Value Function 22
2.4.3 The Need For A Model 22
2.4.4 Model Free Algorithms 23
2.4.5 Exploration vs. Exploitation 23
2.4.6 Continuous State Space 23
2.5 Multi-Agent Reinforcement Learning (MARL) 24
2.5.1 Challenges in MARL 24
2.5.2 Approaches to MARL 24
2.5.3 Applications of MARL 24

3 AGR4BS: Agent, Group, Roles for Blockchain Systems 27
3.1 Related Work 27
3.1.1 Process-Oriented Paradigm 27
3.1.2 Graph-Theoretic Paradigm 28
3.1.3 Object-Oriented Paradigm 29
3.1.4 Agent-Oriented Paradigm 29

3.1.5 Discussion . . . 30
3.2 Organization-centric Modeling for Blockchain Systems . . . 30
3.2.1 Motivations behind Organization-centric Modeling . . . 30
3.2.2 The Agent/Group/Role (AGR) Approach . . . 31
3.2.3 The Methodology for AGR . . . 32
3.3 AGR4BS: A Generic Organizational Model for Blockchain Systems . . . 32
3.3.1 Role Types . . . 32
3.3.2 Group Types . . . 33
3.3.3 Management of the Groups . . . 36
3.3.4 Roles in Detail . . . 37
3.3.5 Interactions . . . 39
3.3.6 Agent Types . . . 40
3.4 Case Studies . . . 40
3.4.1 Bitcoin . . . 41
3.4.2 Ethereum 2.0 . . . 44
3.4.3 Tendermint / Cosmos . . . 46
3.4.4 Hyperledger Fabric . . . 49
3.5 Modeling Attacks . . . 52
3.5.1 Front Running . . . 53
3.5.2 Eclipse Attack . . . 54
3.5.3 Wormhole Attack . . . 55
3.6 Discussion . . . 55
3.6.1 Expressivity of AGR4BS . . . 56
3.6.2 Organizational Differences of Blockchain Systems . . . 56
3.6.3 Robustness and Resilience . . . 58
3.6.4 Reliability . . . 58
3.7 Conclusion . . . 59

4 A role based taxonomy of incentive vulnerabilities 61
4.1 Related Work . . . 61
4.2 Taxonomy Characteristics . . . 62
4.3 Role-based Taxonomy . . . 63
4.3.1 Block Proposer . . . 63
4.3.2 Block Endorser . . . 65
4.3.3 Transaction Endorser . . . 65
4.3.4 Transaction Proposer . . . 66
4.3.5 Blockchain Maintainer . . . 66
4.3.6 Oracle . . . 67
4.3.7 Investee . . . 68
4.4 Conclusion . . . 68

5 A generic blockchain simulator 69
5.1 Related Work . . . 69
5.2 Design rationale . . . 70
5.2.1 Modularity . . . 70
5.2.2 Reproducibility . . . 70
5.2.3 Reinforcement Learning Compatibility . . . 70
5.3 Generic components . . . 70
5.3.1 Messages . . . 70
5.3.2 Network . . . 71
5.3.3 Factory . . . 71

5.3.4 Agents . . . 72
5.3.5 ExternalAgents . . . 73
5.3.6 Groups . . . 74
5.3.7 Roles . . . 74
5.3.8 Environment . . . 76
5.3.9 Scheduler . . . 77
5.3.10 Overview . . . 77
5.4 Blockchain Specific Components . . . 78
5.4.1 Transaction . . . 78
5.4.2 Block . . . 79
5.4.3 Blockchain . . . 80
5.4.4 State . . . 80
5.4.5 Virtual Machine . . . 85
5.4.6 Factory . . . 85
5.4.7 Roles . . . 86

6 MARL Analysis of Incentive Vulnerability in Ethereum 2.0 87
6.1 Ethereum 2.0 . . . 87
6.1.1 Epochs and Slots . . . 87
6.1.2 Checkpoints . . . 88
6.1.3 Validator Duties . . . 88
6.1.4 The Fork Choice Rule . . . 90
6.1.5 Rewards and Penalties . . . 91
6.2 Selfish Block Creation in Ethereum 2.0 . . . 92
6.2.1 Attack description . . . 92
6.2.2 Attack motivation . . . 94
6.3 Simulation Settings . . . 95
6.3.1 Environment . . . 95
6.3.2 Agents . . . 95
6.3.3 Models . . . 96
6.3.4 Actions . . . 97
6.3.5 Observations . . . 97
6.3.6 Reward Functions . . . 98
6.4 Results . . . 98
6.4.1 Discussion . . . 101
6.5 Conclusion . . . 101

7 Conclusion 103
7.1 Summary . . . 103
7.2 Limits . . . 104
7.3 Perspectives . . . 104

Bibliography 107

List of Figures

2.1 The blockchain data structure starts with a genesis block b_0 and cryptographically links successive blocks in reverse order of their block numbers. 15
2.2 Representation of a blockchain system. 16

3.1 Agent/Group/Role representations as a conceptual model (a) and as a cheeseboard diagram (b). 31
3.2 Proposed methodology for defining our generic organizational model. 32
3.3 The initial generic blockchain role type model. 33
3.4 The organizational structure and behavior of the Transaction Management group. 34
3.5 The organizational structure and behaviors of the Block Management group. 35
3.6 The organizational structure and behavior of the Pool group. 36
3.7 The organizational structure and behavior of a *user-defined* Decentralized Application (DApp) group. 37
3.8 The Group Manager role. 37
3.9 The roles and their corresponding attributes and behaviors for blockchain systems. 38
3.10 The interactions type for blockchain systems 39
3.11 The agents for blockchain systems and their interaction models. 40
3.12 An organizational model of Bitcoin-like systems. 42
3.13 43
3.14 Cheeseboard diagram for an organizational view of a Bitcoin system . 44
3.15 An organizational model of Ethereum 2.0 blockchain system. 45
3.16 Sequence diagram of *invest*, *redistribute* and *withdraw* behaviors (Figure 3.6) for an Ethereum staking pool. 46
3.17 Cheeseboard diagram for an organizational view of an Ethereum 2.0 system. 46
3.18 The organizational model of Tendermint-like systems. 48
3.19 Sequence diagram of the *propose block* behavior (Figure 3.5) in Tendermint. 48
3.20 Cheeseboard diagram for an organizational view of a Tendermint system 49
3.21 An organizational model of Hyperledger-like systems. 50
3.22 Sequence diagram of the *propose transaction* behavior (Figure 3.4) in Hyperledger Fabric. 51
3.23 Cheeseboard diagram for an organizational view of a Hyperledger Fabric system 52
3.24 The organizational structure of a transaction based front-running attack. 53
3.25 The organizational structure of an eclipse attack. 54
3.26 The cheeseboard transition diagram of an eclipse attack 54

3.27 The organizational structure of a wormhole attack 55
3.28 The cheeseboard transition diagram of a wormhole attack. 56

5.1 UML Diagram of the Message class 71
5.2 UML Diagram of the Network class 72
5.3 UML Diagram of the Factory class 72
5.4 UML Diagram of the Agent class 73
5.5 UML Diagram of the Context class 73
5.6 UML Diagram of the Agent class 74
5.7 UML Diagram of the Role class 75
5.8 UML Diagram of the Environment class 76
5.9 UML Diagram of the Scheduler class 77
5.10 Overview of the simulator's architecture 77
5.11 UML Diagram of the Transaction class 79
5.12 UML Diagram of the Block class 79
5.13 UML Diagram of the Blockchain class 80
5.14 UML Diagram of the Account class 81
5.15 UML Diagram of the StateChange class 82
5.16 UML Diagram of the CreateAccount and DeleteAccount classes 82
5.17 UML Diagram of the IncrementAccountNonce and DecrementAccount-Nonce classes 83
5.18 UML Diagram of the AddBalance and RemoveBalance classes 83
5.19 UML Diagram of the UpdateAccountStorage classe 84
5.20 UML Diagram of the Receipt class 84
5.21 UML Diagram of the State class 85
5.22 UML VM of the State class 85
5.23 UML Diagram of the blockchain specific Factory class 86

6.1 An Ethereum 2.0 Epoch composed of 32 Slots 88
6.2 Ethereum 2.0 finalization process where a source, justified checkpoint (blue) is linked by a supermajority link (red) to a target, proposed checkpoint (white), leading to the justification of the target and the finalization (green) of the source. 88
6.3 An attestation produced at slot N can be included in a block at slot N + 1 at the earliest. 89
6.4 High level Ethereum 2.0 validator duties 89
6.5 Ethereum 2.0 fork choice rule: LMD-GHOST: A fork between the block of slots $N + 2$ and $N + 3$ is resolved by applying the LMD-GHOST algorithm based on the validators attestations (blues). We assume that validators stakes are uniform for simplicity. Attestations are displayed where they are created. 90
6.6 A malicious block proposal that purposely does not extend the head of the canonical chain. 92
6.7 In this scenario with 32 validators, each having equal stakes the validators favors the block at slot $N + 2$ because the attestation issued at that same slot were not yet aware of block $N + 3$. The proposer boost $(0.4 * \frac{32}{32})$ is not enough to counteract the not yet included attestation (dashed line) and the fork choice rule does not select the malicious fork. 93

6.8 The malicious attester at slot $N+2$ does not respect for fork choice rule and votes for the block at slot $N+1$, therefore the malicious block at slot $N+3$ is favored by the fork choice rule because of the proposer boost, therefore, honest validators attest to it too. 94
6.9 The malicious attester at slot $N+3$ immediately votes for the malicious block, disregarding the honest fork choice rule and favoring the malicious block. 94
6.10 The malicious proposer includes the attestation(s) from the forked block in its own, hoping to steal the proposer reward. 95
6.11 The 32 agents are deterministically arranged during an Ethereum 2.0 epoch with 8 malicious block proposers (red square), 24 honest block proposers(green square), 10 malicious block endorsers (red circle) and 22 honest block endorsers (green circle). 96
6.12 Average protocol reward of both honest and malicious blockchain agents in simulations using the average malicious reward criterion. . . 99
6.13 Number of successful and failed forks in simulations using the average malicious reward criterion. 99
6.14 Average protocol reward of both honest and malicious blockchain agents in simulations using the number of successful and failed forks reward criterion. 100
6.15 Number of successful and failed forks in a simulation in simulations using the number of successful and failed forks reward criterion. . . . 100

List of Tables

3.1 Group differences of blockchain systems 57

4.1 Comparison of studies with a focus on taxonomies concerning the security of blockchains. 62

4.2 The taxonomy of role-based incentive vulnerabilities. Very Low: ●○○○○ , Low: ●●○○○ , Medium : ●●●○○ , High: ●●●●○ , Very High : ●●●●● . . 64

6.1 Weights of the different validator duties. 91

List of Abbreviations

AI	**A**rtificial **I**ntelligence,
AGR4BS	**A**gent, **G**roup, **R**ole for **B**lockchain **S**ystems
BFT	**B**yzantine **F**ault **T**olerent
CAP	**C**onsistency, **A**vailability, **P**artition tolerance
DAG	**D**irected **A**cyclic **G**graph
DQN	**D**eep **Q** **N**etwork
EOA	**E**xternally **O**wned **A**ccount
GHOST	**G**reedy **H**eaviest **O**bserved **S**ub **T**ree
SPoF	**S**ingle **P**oint of **F**ailure
P2P	**P**eer to **P**eer
PBFT	**P**ractical **B**yzantine **F**ault **T**olerent
PoW	**P**roof of **W**ork
PoS	**P**roof of **S**take
DPoS	**D**elegated **P**roof of **S**take
PoA	**P**roof of **A**uthority
MARL	**M**ulti **A**gent **R**einforcement **L**earning
MEV	**M**aximum **E**xtractable **V**alue
RL	**R**einforcement **L**earning

Dedicated to Laura…

Résumé

Cette thèse vise à explorer et à normaliser l'utilisation de l'intelligence artificielle (IA) pour évaluer et améliorer la sécurité des systèmes de blockchain. Plus précisément, nous cherchons à utiliser l'apprentissage par renforcement (RL) et plus particulièrement, l'apprentissage par renforcement multi-agents (MARL) pour former des agents rationnels (*i.e.,* basés sur l'utilité, la recherche de profit) afin de valider que les incitations économiques du système sont alignées sur le comportement rationnel. Il est également possible d'utiliser le MARL comme mécanisme de détection, permettant d'avertir les développeurs si une exploitation donnée du protocole est détectée, ou bien, d'optimiser la capacité de traitement d'un système blockchain en définissant ses hyperparamètres tels que la taille des blocs et l'intervalle entre les blocs en fonction de la charge du réseau en temps réel. Cette approche est destinée à faciliter la conception et le développement de nouveaux systèmes de blockchain publics ainsi qu'à sécuriser les systèmes existants soumis à des mises à jour de leurs mécanismes d'incitation.

À cette fin, nous avons identifié trois besoins majeurs : Premièrement, il n'existe pas de modélisation des systèmes de blockchain qui soit à la fois générique et suffisamment expressive pour s'appliquer à des systèmes de blockchain très différents, et encore moins pour représenter les attaques à la fois au niveau comportemental (*i.e.,* byzantins) et au niveau du système (*i.e.,* attaques du réseau et de la réputation). Deuxièmement, nous n'avons pas connaissance de travaux catégorisant les vulnérabilités d'incitations dans les couches de consensus et d'exécution qui permettraient aux chercheurs et aux développeurs d'orienter leurs efforts vers les composants d'un système de blockchain qui sont les plus susceptibles d'être exploités. Troisièmement, les études RL et MARL nécessitent un environnement, bien qu'il existe de nombreux simulateurs de blockchain, ils sont soit spécifiques à une blockchain en particulier, soit orientés vers un sujet spécifique tel que la production ou la propagation de blocs, il existe un besoin pour un simulateur de blockchain générique, compatible RL et consensus agnostique, permettant une déviation comportementale à n'importe quel niveau du système.

IA dans les systèmes de blockchain

Il existe relativement peu de travaux axés sur l'automatisation de la recherche de vulnérabilités d'incitations dans les systèmes de blockchain avec l'apprentissage par renforcement, mais le sujet a gagné en popularité pendant la période où cette thèse a été menée. La plupart des travaux existants se concentrent sur le consensus Proof of Work (PoW), en particulier sur l'attaque dite Selfish Mining après sa découverte dans (Eyal and Sirer, 2014), qui a été affinée dans (Nayak et al., 2016) et (Sapirshtein et al., 2016) sous les noms de stubborn mining et optimal selfish mining, respectivement.

(Wang et al., 2021a) a proposé une version modifiée du Q-Learning tabulaire maximisant les gain de minage relatif d'un agent dans une configuration à 2 agents, de sorte que les auteurs sont en mesure de faire correspondre la stratégie de minage

égoïste optimale sans les connaissance générale du système requise dans (Sapirshtein et al., 2016). Ce résultat a montré des perspectives prometteuses pour le RL, en particulier dans le contexte des blockchains utilisant un consensus de preuve de travail. (Wang et al., 2021b; Yang et al., 2020) appliquent tous deux le RL à une variante du minage égoïste appelée minage égoïste par corruption, qui consiste essentiellement à fusionner la stratégie initiale de minage égoïste avec des attaques par corruption telles que décrites dans (Gao et al., 2019), en étendant l'idée initiale de (Bonneau, 2016). (Yang et al., 2020) applique un modèle d'attaquant rationnel conforme à notre ligne de travail, garantissant qu'une attaque est une menace valide dans un cadre multi-agents avec une motivation à long terme prise en compte par les attaquants. Le RL est utilisé ici pour que l'attaquant choisisse entre un ensemble de stratégies déjà définies dans le but de maximiser son profit, l'expressivité des modèles de RL sont donc limités et contraints à ce qui est déjà connu.

En réponse à ces nouvelles stratégies d'exploitation basées sur l'IA, plusieurs systèmes de détection eux même basés sur l'IA ont été conçus. Ces systèmes sont présentés dans (Peterson et al., 2022; Wang et al., 2021c). Ces systèmes n'utilisent pas spécifiquement le RL, mais s'appuient sur l'IA pour fournir une alerte précoce en cas de détection d'une attaque de minage égoïste en utilisant l'apprentissage automatique et l'apprentissage profond basés sur les données passées et actuelles de la blockchain.

(Hou et al., 2021) est le plus proche de notre travail : Les auteurs proposent un cadre pour automatiser l'analyse des attaques dans les systèmes de blockchain, et discutent des avantages de l'utilisation de l'apprentissage par renforcement basé sur l'apprentissage profond dans le contexte des systèmes de blockchain. Ils fournissent des résultats inédits à la fois sur Bitcoin et sur une première version d'Ethereum 2.0. Ils proposent de suivre une approche en trois étapes, dont la première consiste à créer un environnement de simulation du protocole en question. La deuxième étape consiste à choisir un modèle d'attaque qui inclut la définition du nombre et du type d'agents. La troisième et dernière étape consiste à sélectionner un algorithme RL pour mener l'étude. Ils ne proposent pas de composants déjà utilisables pour modéliser, classifier ou simuler un système blockchain. Leurs résultats montrent qu'à mesure que des agents plus rationnels sont ajoutés au système, les stratégies de minage égoïstes deviennent moins rentables, ce qui démontre l'importance de simulations à plus grande échelle pour reproduire fidèlement la dynamique des systèmes de blockchain publics.

(Zhang et al., 2020) propose une approche intéressante dans laquelle les agents RL sont formés pour optimiser les hyperparamètres d'une blockchain fragmentée tels que l'intervalle de re-fragmentation, le nombre de fragments et la taille des blocs, ce qui permet au système de s'adapter dynamiquement à la charge de travail. Les auteurs ont placé l'IA au centre du protocole de la blockchain, en lui donnant un contrôle total sur certains paramètres critiques de la blockchain. Cela peut ouvrir le système à de nouveaux vecteurs d'attaque tels que l'exploitation de modèles pour nuire à la blockchain en utilisant le système même qui est censé aider à la sécuriser.

Il convient également de noter que des groupes de recherche spécifiques à la blockchain explorent la RL dans le contexte de leur propre blockchain, comme le Robust Incentive Group (RIG[1]) de la Fondation Ethereum.

Contributions

Nos contributions sont :

[1] https://ethereum.github.io/rig/ dernier accès le 15/10/2023

- **AGR4BS** : Un modèle blockchain organisationnel générique basé rôles qui peut représenter des systèmes de nature différentes.
- Taxonomie des vulnérabilités d'incitations de la blockchain fondée sur le modèle AGR4BS.
- **pyAGRBS** : Un simulateur de blockchain générique open source également aligné sur AGR4BS, conçu pour la modularité et la compatibilité avec l'apprentissage par renforcement.

AGR4BS

Le Chapitre 3 décrit notre première contribution et définit notre modèle blockchain organisationnel et générique nommée AGR4BS (Agent, Group, Roles for Blockchain Systems) inspiré du modèle AGR (Ferber et al., 2004). Il permet de représenter la composante sociale, multi-agent et organisationnelle des systèmes blockchain par le biais de trois abstractions de haut niveau, a savoir, *Agent*, *Groupe* et *Rôle*, de ce fait, il se différencie des outils de modélisation existants pour les systèmes blockchain.

Les *Agents* sont des entités actives et communiquantes pouvant endosser un ou plusieurs *rôle* au sein de *groupes*. Un agent se doit d'avoir au moins un rôle, et peut appartenir a plusieurs groupes simultanément.

Les *Rôles* sont des représentations abstraites de fonctions qu'ont les agents au seins de groupes. Un rôle décrit les responsabilités et contraintes qui sont associées aux agents, il se compose de routines concrètes appelées *comportements*.

Les *Groupes* sont des structures organisationnelles composées d'agents ayant des activités et / ou des buts communs. Les agents membres d'un même groupe peuvent communiquer entre eux.

Dans le cadre des systèmes blockchain, nous avons identifié deux types de groupes:

- Groupe structurel
- Groupe d'intérêt

Un groupe structurel est un groupe nécessaire au bon fonctionnement d'un système blockchain donné, (*ex:,* Les Mineurs de Bitcoin). Nous définissons deux groupes structurels nécessaires : Le groupe de gestion des transactions ainsi que le groupe de gestion des blocs, toute représentation d'un système blockchain se doit de posséder ces deux groupes. Un groupe d'intérêt est un groupe superflu, donc non nécessaire au bon fonctionnement du protocole. Les agents peuvent créer et rejoindre des groupes d'intérêts afin de fournir de nouvelles fonctionnalités au système blockchain, celles-ci peuvent être potentiellement rentable pour les membres du groupe (*ex:,* Une mining pool).

Les Rôles nécessaires aux systèmes blockchain sont :

- Transaction proposer
- Transaction endorser
- Block Proposer
- Block Endorser
- Blockchain Maintainer

- Investor
- Investee
- Contractor
- Oracle

Transaction proposer permet de créer et proposer une transaction (*i.e.,* une transition d'état du système). *Transaction endorser* permet de voter pour ou contre l'inclusion dans un bloc d'une transaction proposée par un *Transaction proposer*. *Block proposer* permet de créer et proposer de nouveaux blocs au système. *Block endorser* permet de voter pour ou contre l'inclusion dans la chaîne d'un bloc propose par un *Block Proposer*. *Blockchain Maintainer* maintient une copie locale de la blockchain et s'assure de son intégrité. *Investor* permet d'investir des ressources financières ou computationelles au sein du système blockchain. *Investee* permet de recevoir des investissements dans le but d'accomplir des taches spécifiques pour ses investisseurs, moyennant une commission. *Contractor* fournit des services interne a la blockchain aux autre participants sur une base contractuelle. Ce rôle permet par exemple de représenter et d'implémenter des contrat intelligents ayant des fonctionnalités arbitraires. Enfin, le rôle *Oracle* permet de fournir des services extérieurs, comme des flux de données a la blockchain et donc a ses participants.

Un agent utilisant le rôle de *Contractor* est donc un contrat intelligent, tandis qu'un agent qui ne l'utilise pas est considéré comme un noeud, c'est a dire, un participant actif du système.

En utilisant ce modèle nous montrons que nous sommes en mesure de représenter des systèmes blockchain de nature différentes comme Bitcoin (Nakamoto, 2008) , Tendermint (Kwon, 2014), Ethereum 2.0 (Buterin et al., 2020) ou encore Hyperledger Fabric (Rocha and Ducasse, 2018). En effet, ces modèles ne partagent pas le même mécanisme de consensus et donc n'ont pas la même structure, ils ne supportent pas tous les contrat intelligents et peuvent avoir des mécanismes de permissions différents, mais il peuvent tous être modélisés avec un seul et unique modèle : AGR4BS.

Enfin, il est possible de se concentrer sur des déviations de rôle ou de comportements afin de modéliser des attaques du système ou de participants par d'autres participants. Certaines des attaques que nous modélisons sont le délit d'initié, l'attaque éclipse ou encore l'attaque de trous de ver.

Le modèle AGR4BS, de part son expressivité et sa modularité sert de fondation a la taxonomie ainsi qu'a notre simulateur blockchain.

Taxonomie

Notre seconde contribution est détaillée dans le Chapitre 4, il s'agit d'une taxonomie des vulnérabilités d'incitations dans les systèmes blockchains basée sur la notion de rôle définie par AGR4BS.

Pour classer, catégoriser et mesurer les vulnérabilités, nous utilisons principalement les concepts suivants : famille d'impact, gravité, risque, échelle, et score de priorité.

Famille d'impact se rapporte au type d'impact attendu de l'exploitation de la vulnérabilité. Trois possibilités sont envisagées : L'équité, l'économie et la sécurité. Un impact sur l'équité se produit chaque fois qu'une discrimination entre les agents se produit pour une raison quelconque qui ne fait pas partie du protocole. De même,

tout déséquilibre entre la proportionnalité des ressources investies et la récompense reçue est inclus dans cette famille d'impact. Un impact économique se produit lorsque l'économie du système est perturbée, par exemple par une augmentation artificielle des frais de transaction. Un impact sur la sécurité se produit lorsqu'une propriété essentielle de la blockchain est compromise, par exemple l'impossibilité de finaliser les blocs nouvellement créés ou la perte d'intégrité de la blockchain en raison de l'inclusion de données invalides.

Sévérité définit le niveau d'impact d'une attaque réussie et prend une valeur très élevée, élevée, moyenne, faible et très faible. Ces niveaux ne sont pas basés sur une notion objectivement quantifiable de gravité, mais sont utilisés pour classer les vulnérabilités de manière informelle et aider ainsi à calculer leurs scores de priorité respectives. Très faible » signifie qu'un agent ou un groupe d'agents est légèrement touché mais qu'il fonctionne toujours, sans impact quantifiable sur les groupes ou le système. 'Faible' implique également qu'un agent, ou un sous-groupe d'agents, est impacté de manière plus significative, voire non fonctionnel, alors que le groupe et le système de blockchain dont il fait partie sont toujours fonctionnels. Un niveau de gravité « moyen » a un impact sur les agents et les groupes d'une manière qui ne met pas en péril le système, mais qui a des conséquences sur au moins l'une de ses propriétés essentielles, telles que l'équité, la sécurité ou l'économie. Un niveau de gravité « élevé » implique un impact non négligeable sur le système. Enfin, le niveau « Très élevé » correspond à une menace immédiate, telle qu'un problème général d'équité ou l'arrêt pur et simple du système.

Risque fait référence à la faisabilité d'une attaque en termes de ressources nécessaires pour la mener a bien. Les niveaux de risque sont similaires à ceux définis pour la gravité : « Très élevé », « Élevé », « Moyen », « Faible » et « Très faible ». Très élevé » signifie que la vulnérabilité associée est relativement facile à mettre en place car elle ne nécessite que quelques ressources. Le niveau « élevé » correspond à une attaque qui nécessite un certain nombre de ressources, mais qui reste réalisable par la plupart des participants. Moyen » signifie que l'attaque nécessite une quantité non négligeable de ressources. Les termes « faible » et « très faible » sont utilisés pour décrire les attaques nécessitant des ressources excessivement importantes.

Score de priorité est une valeur numérique calculée en fonction de la sévérité et du risque.

Nous listons ainsi différents vecteurs d'attaque connus dans des systèmes blockchain de différentes nature, tel que Bitcoin ou Ethereum, et les associons avec un ou plusieurs rôles AGR4BS, puis, nous calculons leurs scores de priorités. Ce travail nous permet de mettre en avant les rôles et comportements les plus sujets aux déviations ayant un fort impact potentiel sur le système tel que *Blockchain Maintainer* et *Block proposer*. La taxonomie permet aussi de catégoriser de futures attaques ainsi que de quantifier leur impact. Une vulnérabilité d'incitation touchant le protocole Ethereum 2.0 et pouvant mener a la créations volontaire de branches adverses dans la blockchain a attirée notre attention d'une part en raison de son score de priorité élevé mais aussi car elle touche un protocole relativement jeune. Cette vulnérabilité, sélectionnée grâce a notre taxonomie basée sur AGR4BS fera l'objet d'une étude pratique et donc d'une mise en application du cadre de travail que nous proposons

Simulateur

Dans le Chapitre 5 nous présentons notre troisième et dernière contribution : un simulateur blockchain, basé rôles et compatible avec le MARL. Il existe plusieurs

simulateurs de blockchain dits génériques dans la littérature. La plupart d'entre eux sont mentionnés dans (Albshri et al., 2022). Ils adoptent pour la plupart un mécanisme événementiel (Babulak and Wang, 2008), aligné sur la nature des systèmes de blockchain. Cependant, ils n'adoptent pas toujours une approche basée sur les agents, et certains d'entre eux ne considèrent que le réseau lui-même, en se concentrant sur les transactions et les temps de diffusion des blocs en fonction d'une topologie de réseau spécifique, et non sur l'analyse des incitations, parfois même en ne permettant pas de telles études en raison de choix de conception.

Les simulateurs les plus proches de nos besoins sont BlockSim (Alharby and Moorsel, 2019), MAX (Gürcan, 2024) et DAGSim (Zander et al., 2019). Ces simulateurs ne sont pas basés sur le modèle AGR a l'exception de MAX, mais ce dernier manque de performance pour permettre l'entraînement de modèles d'intelligence artificielle pendant une simulation, en particulier a cause de son modèle de temps non événementiel. Nous avons donc pris la décision de développer notre propre simulateur, dont l'architecture se base sur AGR4BS. Écrit en Python, ce simulateur est déterministe, événementiel et modulaire. Ainsi, les agents peuvent jouer un nombre arbitraire de rôles qui comportent eux même un certain nombre de comportements permettant la réalisation de fonctions de bas niveau. Des modèles blockchain comme Bitcoin ou Ethereum 2.0 avec un support complet des contrat intelligents sont déjà disponibles (https://github.com/hroussille/agr4bs), les chercheurs ou développeurs peuvent librement modifier une implémentions existante ou bien développer les rôles et comportements nécessaires pour implémenter un nouveau modèle.

Simulation

Nos trois contributions se rejoignent dans le Chapitre 6 ou nous présentons une étude d'une vulnérabilité potentielle du système de récompense des contributeurs du protocole Ethereum 2. Elle permettrait a des agents rationnels des déviations des rôles *Block Proposer* et *Block Endorser* afin de créer volontairement des branches adverses dans la blockchain afin de voler les récompenses des agents honnêtes. Cette vulnérabilité fut choisie en fonction de son score de priorité élevé dans notre taxonomie et du fait qu'elle impacte un protocole récent qui se trouve être la seconde blockchain en terme de capitalisation. Cette étude permet aussi de démontrer la mise en pratique du MARL dans notre simulateur, et sert donc d'exemple concret du cadre de travail que nous proposons dans cette thèse.

Nous considérons donc 32 agents blockchain, dont 8 agents utilisant un rôle déviant de *Block Proposer* et 10 utilisant un rôle déviant de *Block Endorser*, les rôles déviants contiennent des comportements représentés par des modèles d'apprentissage profond entraînés avec l'algorithme Deep Q Learning (DQN). Les agents évoluent dans un protocole Ethereum 2.0 correct. Nous considérons deux fonctions objectifs pour les agent, l'une étant un critère économique collectif visant a déterminer si les agents malicieux créent des branches pour optimiser leur récompenses. La seconde étant un critère base sur le nombre de branches adverses crées visant a déterminer si les agents malicieux optimisent leur récompenses en créant des branches. La simulation utilisant la première fonction objectif ne montre aucun signe de convergence des agents malicieux au cours de l'apprentissage, tandis qu'en utilisant dans la seconde fonction objectif, les agents malicieux créent de plus en plus de branches et obtiennent une récompense moyenne supérieur a celle des agents honnêtes. Ces résultats montrent que l'utilisation du MARL est possible, mais met en lumière des difficultés inhérentes a son utilisation dans les systèmes blockchain, la nature asynchrone de

ces environnent et les multiple interactions entre les agents violent certaines hypothèses de base de l'apprentissage par renforcement, comme la stationnarité de la fonction objectif et de la fonction de transition d'état, ce qui a tendance a rendre l'apprentissage instable. Nous recommandons donc d'utiliser le MARL quand cela est possible, mais aussi de de considérer d'autres approches automatiques qui ne reposent pas sur ces mêmes hypothèses comme les algorithmes génétiques.

Chapter 1

Introduction

1.1 Context

This thesis aims at exploring and standardizing the usage of Artificial Intelligence (AI) to assess and improve security of blockchain systems. Specifically, we aim at using Multi-Agent Reinforcement Learning (MARL) to train rational (*i.e.*, utility based, profit seeking) and irrational agents in order to validate that the system's incentives are aligned with the rational behavior. It is also possible to use MARL as a smart detection mechanism, warning the system if a given exploit is detected, or to optimize the workflow of a blockchain system by defining its hyperparameters such as block size and block rate according to real time network load. This approach is intended to help the design and development of new public blockchain systems as well as secure existing ones subject to updates of their incentive mechanisms.

To this end, we identified three major needs: First, there is no modeling of blockchain systems that is both generic and expressive enough to be applicable to widely different blockchain systems, let alone be able to represent attack both at a behavioral level (*i.e.*, faulty / byzantine implementation) and at a system level (*i.e.*, network and reputation attacks). Second, we did not find any work categorizing incentive vulnerabilities in both the consensus and execution layers that would allow researchers and developers to direct their efforts towards the components of a blockchain system that are most likely to be exploited. Third, RL and MARL studies require an environment, while there exist many blockchain simulators, they are either blockchain specific or opinionated towards a specific topic such as block production or block propagation, there is a need for a generic, RL compatible, consensus agnostic, blockchain simulator allowing behavioral deviation at any level of the system.

1.2 AI in blockchain systems

There are relatively few existing works focused on automating incentive vulnerability search in blockchain systems with Reinforcement Learning, but the topic did gain in popularity during the time this thesis was conducted. Most of the existing ones focus on Proof of Work (PoW) consensus, specifically the so-called Selfish Mining attack following its discovery in (Eyal and Sirer, 2014) which was further refined in (Nayak et al., 2016) and (Sapirshtein et al., 2016) under the names of stubborn mining and optimal selfish mining respectively.

(Wang et al., 2021a) proposed a modified version of tabular Q-Learning maximizing an agent relative mining gain in a 2-agent setup, so that the authors are able to match the Optimal Selfish Mining strategy without the general system knowledge required in (Sapirshtein et al., 2016). This result showed promising perspectives for RL, especially in the context of PoW. (Wang et al., 2021b; Yang et al., 2020) both apply

RL to a variant of selfish mining called bribery selfish mining, which is essentially merging the initial selfish mining strategy with bribery attacks as described in (Gao et al., 2019) itself, extending the initial idea of (Bonneau, 2016). (Yang et al., 2020) enforces a rational attacker model in line with our line of work, ensuring that an attack is a valid threat in a multi-agent setting with long term incentive taken into account by the attackers. RL is used here so that the attacker chooses between a set of already defined strategies with the aim of maximizing profit, the expressivity of the RL policy is therefore limited and constraint to what is already known.

In response to those new AI powered mining strategies, several AI based detection systems were designed. Such systems are presented in (Peterson et al., 2022; Wang et al., 2021c). Those systems do not specifically use RL, but rely on AI to provide early warning to the blockchain system in case a selfish mining attack is detected using machine learning and deep learning for inference based on past and current blockchain data.

(Hou et al., 2021) is the closest to our work: The authors propose a framework to automate attack analysis in blockchain systems, and discuss the benefits of using deep learning based reinforcement learning in the context of blockchain systems. They provide novel results on both Bitcoin and an early version of Ethereum 2.0. They propose to follow a three stage approach, where the first step is to build a simulation environment of the protocol of interest. The second step is to choose an attack model which includes the definition of the number, and type of agents. The third and final step is to select a RL algorithm to conduct the study. They do not propose already usable components to model, classify nor simulate a blockchain system. Their result show that as more rational agents are added to the system, selfish mining strategies become less profitable, which demonstrates the importance of larger scale simulations to closely match the dynamics of public blockchain systems.

(Zhang et al., 2020) proposes an interesting approach where RL agents are trained to optimize sharded blockchain hyperparameters such as the re-sharding interval, the shard number and block size, effectively allowing the system to scale dynamically under load by carefully handling the tradeoff between security and throughput. The authors put AI in the center of the blockchain protocol, giving it full control over some critical blockchain parameters. This may open the system to new attack vectors such as models exploitation to harm the blockchain using the very system that is supposed to help secure it.

It must be noted also that blockchain specific research groups are exploring RL in the context of their own blockchain, such as the Robust Incentive Group (RIG[1]) of the Ethereum Foundation.

1.3 Thesis Overview

1.3.1 Contributions

Our contributions are threefold :

- **AGR4BS** : A generic organizational, role based blockchain model that can represents widely different systems.
- A taxonomy of blockchain incentive vulnerabilities grounded in the AGR4BS model.

[1] `https://ethereum.github.io/rig/` last accessed on 15/10/2023

- **pyAGRBS** : An open source, generic blockchain simulator also aligned with AGR4BS, designed for modularity and reinforcement learning compatibility.

1.3.2 Document Outline

This document is structured as follows :

- Chapter 2 - **Background** - Introduces the necessary concepts to grasp blockchain systems and Reinforcement Learning.
- Chapter 3 - **AGR4BS** - Presents AGR4BS, a generic, role base blockchain model that can represent widely different blockchain systems and vulnerabilities.
- Chapter 4 - **A role based taxonomy of incentive vulnerabilities** - Describes a novel taxonomy of blockchain incentive vulnerabilities grounded in AGR4BS.
- Chapter 5 - **A generic blockchain simulator** - provides insight on the design choices made for the architecture of our role based blockchain simulator.
- Chapter 6 - **Towards RL experiments in blockchain systems** - describes one practical study of an Ethereum 2.0 vulnerability in a coherent framework based on the previous contributions.

Chapter 2

Background

2.1 Distributed Systems

Distributed systems have become a fundamental paradigm in computer science and have a significant impact on various aspects of our modern world, the most prominent example being the Internet itself. In this section, we provide an overview of distributed systems, their importance, and the key concepts that underlie their design and implementation as they will be of upmost importance to understand blockchain systems.

A distributed system is a collection of interconnected, autonomous computers that work together as a unified system to provide a set of services or perform tasks. Unlike traditional centralized systems, where all processing occurs on a single machine, distributed systems distribute the computation and data across multiple nodes. This distribution offers several advantages, including fault tolerance, scalability, and improved performance. The motivation for using distributed systems arises from the need to solve complex problems that cannot be efficiently or safely addressed by a single, monolithic machine. Such systems can scale horizontally by simply adding more machines, allowing them to handle increasing workloads and user demands. Additionally, through data and services replication across multiple nodes and potentially diverse geographical locations, distributed systems can continue to operate even in the presence of hardware failures or network disruptions since they never expose a Single Point of Failure (SPoF). This replication can also be used to distribute tasks across multiple machines, leading to significant performance improvement and parallelize tasks.

To understand distributed systems, it is essential to grasp some fundamental concepts such as: Concurrency, Consistency and Fault Tolerance. Nodes in a distributed system need to communicate with each other to exchange information and coordinate activities. This involves designing efficient communication protocols and handling potential network delays and failures. Concurrency in a distributed system involves multiple tasks running concurrently on different nodes. It requires managing issues and caveats regarding synchronization and communication in what is essentially an asynchronous system. Consistency is a challenging problem: ensuring a consistent view of the data or state across the distributed nodes. This can be achieved though distributed transactions and data replication. Fault Tolerance is mandatory as a distributed system must be designed to handle arbitrary nodes failures, up to a certain threshold, and stay functional. Fault tolerance is achieved through clever algorithms, redundancy, replication and fault detection when possible.

2.2 Consensus

Consensus is a fundamental problem in distributed systems, playing a pivotal role in ensuring that a distributed system of nodes can agree on a single, consistent value or decision, even in the presence of failures or unreliable components. In distributed systems, nodes often need to agree on critical decisions, such as electing a leader, committing a distributed transaction, or reaching a consensus on a specific value or order of events. Achieving consensus is challenging because neither the participants nor the network can be trusted. Nodes can crash, experience network failures, or exhibit Byzantine faults, making it difficult to ensure that a majority of nodes are available to reach a consistent decision. Here lies most of the challenge of reaching consensus in distributed systems: in an asynchronous network it is impossible to differentiate between a genuine crash, a Byzantine fault or an arbitrary long network delay when waiting for a response from another node.

2.2.1 Consensus Algorithms

Several consensus algorithms have been developed to address the challenges posed by distributed systems. Some of the most well-known consensus algorithms include:

- Practical Byzantine Fault Tolerance (PBFT) (Castro and Liskov, 1999): PBFT is designed to tolerate Byzantine faults and is commonly used in permissioned blockchain networks like Hyperledger Fabric.
- Paxos (Lamport, 2001): Proposed by Lamport, Paxos is a widely used consensus algorithm known for its elegance and rigor. It forms the basis for many distributed systems, including Apache ZooKeeper and Google's Chubby.
- Raft (Ongaro and Ousterhout, 2014): Raft is another consensus algorithm designed for simplicity and understandability. It focuses on ease of implementation and is often used in distributed databases like etcd and CockroachDB as well as blockchain systems.
- HoneyBadgerBFT (Miller et al., 2016): HoneyBadgerBFT combines asynchronous Byzantine fault tolerance with cryptographic techniques to achieve consensus. It is currently used in some blockchain and cryptocurrency systems.

2.2.2 Challenges and Trade-Offs

While consensus algorithms address the need for agreement in distributed systems, they also introduce challenges and trade-offs. The key considerations revolve around complexity, scalability and latency. Achieving consensus may introduce additional latency into the system operations, particularly in algorithms based on the notion of committee, which rely on multiple rounds of communication. The complexity of such algorithms must also be taken into account, as lowering latency through an efficient implementation may favor the introduction of unwanted bugs at the core of the system. Most consensus algorithms do not scale linearly with the number of nodes, since they have a limited scalability and the decentralization can only be extended up to a certain point imposed by technical and mathematical limitations. The famous CAP theorem (Brewer, 2000) states that between the core properties of *consistency*, *availability* and *security*, any distributed system may only ensure two of

them, sacrificing the third one. All the algorithms mentioned in 2.2.1 sacrifice availability. In other words, the system will halt if it is unable to ensure a consistent and secure workflow.

2.3 Blockchain Systems

Blockchain technology is a prime example of a distributed system that has gained immense popularity in recent years. In this section, we explore how blockchain systems operate as distributed systems, starting with an overview of their basic components and concepts.

2.3.1 Blockchain Data Structure

At the core of every blockchain system are two fundamental data structures: *blocks* and *transactions*. A Transaction is a cryptographically signed action or operation recorded on the blockchain leading to a transition in the overall state of the system. Most blockchain systems bundle transactions into blocks. Blocks can be seen as containers for transactions, cryptographically linked to a parent block to form the chain. Each block typically includes a reference to the previous block called its parent block and is uniquely identifiable though its hash. Blockchain systems maintain a distributed ledger, also referred to as *state*, that records these ordered transactions in a way that is secure, transparent, and tamper-resistant.

The data structure of a blockchain maintained by a participant can be modeled as a dynamic append-only tree, where each block b_i contains a cryptographic reference to its previous block b_{i-1} (Figure 2.1). b_0 is the root block known as the *genesis block* and b_h is the furthest block from the genesis block which is referred to as the *blockchain head*.

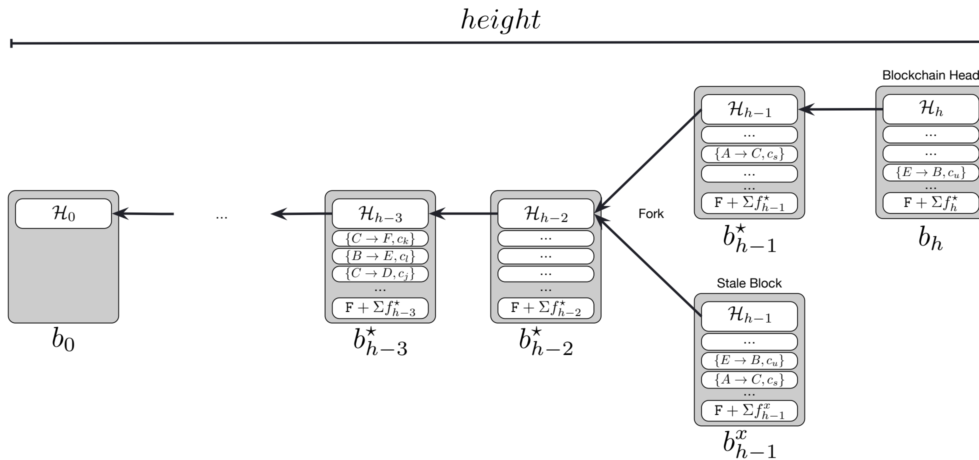


FIGURE 2.1: The blockchain data structure starts with a genesis block b_0 and cryptographically links successive blocks in reverse order of their block numbers.

A block b_{i-1} can have multiple children blocks, which causes a situation called a *fork*. One of the chains is then selected as the *main chain* according to the blockchain protocol used. All chains other than the main chain are called *side chains*. If, at any time, there exists more than one main chain candidate (i.e., there are multiple heads), the blockchain is said to be *inconsistent*. This situation disappears when a new block

extends one of these side chains. The blocks on the other branches are discarded and referred to as *stale blocks*.

Technically speaking, all participants store unconfirmed transactions in their own *memory pools* and confirmed transactions in their *local blockchains* (Figure 2.2).

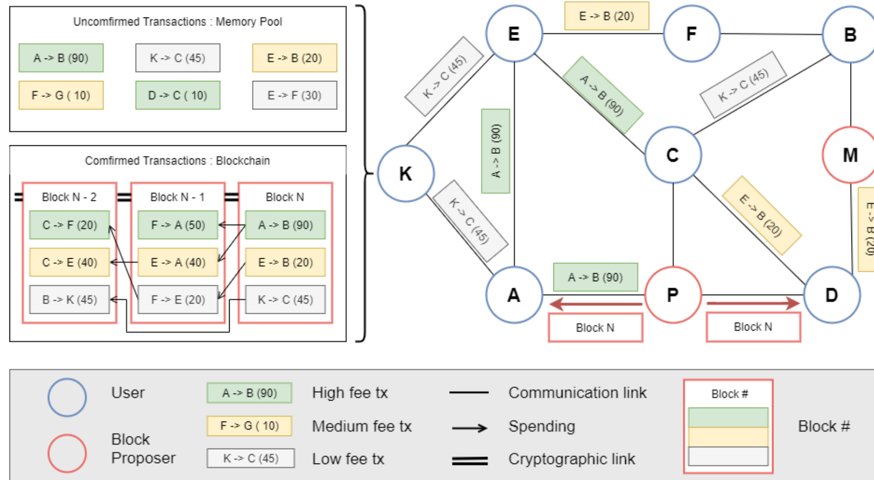


FIGURE 2.2: Representation of a blockchain system.

Basically, there are two main types of participants for all types of blockchain systems: *users* and *block proposers*. Users create transactions with a fee and then *propose* them by diffusing across the blockchain network to be confirmed (i.e., totally ordered and cryptographically linked to the blockchain). Each participant, receiving the proposed transaction, *validates* and diffuses it to its own neighbors. After receiving a certain number of transactions, block proposers *select transactions* to confirm and order them by *creating a dedicated block* through a blockchain consensus mechanism, e.g., Proof-of-Work (PoW), Proof-of-Stake (PoS), Delegated Proof-of-Stake (DPoS), Byzantine Fault-Tolerance (BFT); for a review see (Bano et al., 2019; Wang et al., 2019). Depending on the mechanism used and the blockchain technology, block proposers are referred to as miners (Nakamoto, 2008), validators (Wood, 2014), bakers (Goodman, 2014), orderers (Androulaki et al., 2018), committee members (Buchman et al., 2018) etc. respectively. The successful block proposer *proposes* its block by diffusing it to the network to be appended to the local blockchains. Each participant receiving the proposed block *validates* it against its local blockchain and diffuses it to its own neighbors. Upon inclusion of its block by all participants, the corresponding successful Block Proposer is rewarded by $R + F_i$ where R is a static block reward and F_i is the total amount of fees of transactions included in the block i. This way, user and block proposer participants altogether maintain a shared data structure referred to as the blockchain.

Although blockchains initially only provided cryptocurrency related operations, the support of Turing-complete Smart Contracts (SC) that encode arbitrary data processing logic has been introduced in 2014 (Wood, 2014). With this advancement, blockchains evolved from merely cryptocurrency platforms to distributed transactional and logical systems. In smart contract enabled blockchains, transactions consisting of smart contract invocations are executed by all participants willing to continuously maintain the blockchain state. Today, such blockchain systems can be considered as world-scale decentralized computers, Ethereum (Buterin, 2014) being the most well-known example.

2.3.2 Decentralized Applications

Smart contracts allow anyone to create user defined secure applications, called Decentralized Applications (DApp), that exist and run on an underlying blockchain system. Many DApps such as exchanges, money loans, games, or payment terminals are already being used today (Cai et al., 2018).

Recently, DApps are being increasingly utilized in performing financial functions (e.g., lending or borrowing funds, going long or short on a range of assets, trading coins) on blockchain systems, called Decentralized Finance (DeFi) applications (Werner et al., 2021). A popular application area of DeFi is Decentralized Exchange, (DEX) (Foundation and Development, 2019) where participants trade assets. A DEX application relies on a smart contract called a Liquidity Pool, which is responsible for locking funds and providing *currency availability* (aka: liquidity) to its participants. This way, the participants can invest their money and contribute to the DEX (or any other system relying on that pool) in exchange for interests over time. Another example of DeFi is the Borrow/Lend application which uses also Liquidity Pools to allow participants to borrow in the currency of their choice if available, without requiring a central institution such as a bank.

Besides individual usage, real entities like companies can use DApps to represent and regulate themselves securely and autonomously. The collection of such decentralized applications is called a Decentralized Autonomous Organization (DAO) (El Faqir et al., 2020). Thanks to the smart contracts, a company represented as a DAO can work with external partners and execute commands based on them without any human intervention. An example of a DAO is Pie DAO (PieDAO, `https://www.piedao.org/`, accessed on 23 June 2021) which is a decentralized asset allocation system aimed at automating wealth creation. Users can create, join or leave allocations (i.e., investment diversification plans). Participants will vote for or against the allocations of their choice. Pie DAO is effectively bringing crowd wisdom to the investment world.

2.3.3 Oracle in Blockchain Systems

In a blockchain system, by design, there is no proper way to add external information in a trusted manner. Either the provider or the data itself is trusted. Therefore, any interaction between the blockchain and the outside world contradicts the blockchain trustless philosophy. However, to leverage the power of the blockchain technology and, more specifically, smart contracts, such interactions are often necessary and desired.

The current solution is to use oracles (Blockchain Oracles, `https://blockchainhub.net/blockchain-oracles/`, accessed on 2 July 2021): participants bridging the blockchain system with the outside world (i.e., Web Services, sensor data stream, etc.). The issue of having such trusted entities and the related vulnerabilities in public blockchain systems have already been discussed as the *Oracle problem* (Caldarelli, 2020; Lo et al., 2020).

2.3.4 Permission Models in Blockchain

Blockchain networks can be categorized into different permission models based on who is allowed to participate in the network and validate transactions. The three primary permission models are:

Public Blockchains

Public blockchains are open and permissionless networks, allowing anyone to participate as a node in the Peer-to-Peer (P2P) network. They are highly decentralized as no single entity controls the network; nodes validate transactions through the consensus mechanism. Due to this decentralization, they exhibit high censorship resistance, as one would need to control an overwhelming proportion of the network to effectively censor anyone. Since they are public, they provide the maximum level of transparency: any transaction or data is public and therefore visible to anyone. Transparency is a key element for both trust and monitoring in public blockchains. Bitcoin (Nakamoto, 2008) and Ethereum (Buterin, 2014) are the most well known examples of public blockchains. In this thesis, we will mostly focus on public blockchains.

Consortium Blockchains

Consortium blockchains are semi-private networks where a predefined group of participants, often businesses or organizations, collectively validate transactions. Such blockchains are said to be semi-decentralized, since they are more centralized than public blockchains but less than private ones. A selected group of distinct trusted entities may validate transactions and create blocks. Access to a consortium blockchain is often conditioned on receiving permission by one or several members of the consortium. Comparatively to public blockchains, the low number of entities taking part in block creation allows higher throughput. This is effectively a trade-off where decentralization and distribution is exchanged in favor of efficiency, making them suitable for enterprise applications. Several consortium blockchains already exist today, such as Hyperledger Fabric (Androulaki et al., 2018).

Private Blockchains

Private blockchains are fully controlled and operated by a single entity or organization and are by definition centralized. They can provide a high level of privacy since they can be used for, and within, a closed system, therefore providing privacy to both the owning organization and the participants. Such blockchain permission schemes can be used in supply chain management or record-keeping.

2.3.5 Consensus in Blockchain Systems

Here, we discuss the most well known consensus mechanisms in blockchain systems, namely Proof of Work and Proof of Stake, thus providing a high level overview of what the goal of the consensus is in a public blockchain system. In a blockchain system, the goal of the consensus is to allow the participants of the system to come to an agreement about the block considered as the head of the chain, which in turn defines the canonical chain.

Proof of Work (PoW)

Proof of Work (PoW) is a consensus mechanism or algorithm used in blockchain networks to validate and secure transactions and add new blocks to the blockchain. It was first introduced as a fundamental component of Bitcoin (Nakamoto, 2008) by its pseudonymous creator, Satoshi Nakamoto. PoW plays a crucial role in preventing

various security threats, ensuring network integrity, and regulating the creation of new blocks in a decentralized manner.

The revolution of Bitcoin lies in PoW in the sense that the nodes come to a consensual view of the state of the system in a probabilistic way.

In a PoW-based blockchain network, participants, known as "miners," compete to solve complex mathematical puzzles. These puzzles are computationally intensive and require substantial computational power, and thus energy consumption. Miners collect a set of unconfirmed transactions with the highest fee from the network users and bundle them into a candidate block, which they aim to add to the blockchain. These unconfirmed transactions represent a collection of user-generated transactions waiting to be processed. To add a new block to the blockchain, miners must find a specific value, called a "nonce", that, when combined with the candidate block's data and hashed, produces a hash value that meets specific criteria. This criteria typically involves generating a hash that starts with a certain number of leading zeros. Achieving this requires miners to repeatedly guess and compute the hash value until they find one that meets the criteria.

Miners compete against each other to find this nonce, as the first miner to solve the puzzle gets the opportunity to add the new block to the blockchain. This process is known as "mining." It is important to note that the chances of finding the correct nonce are purely based on computational power, making it a probabilistic process. Miners with more computational resources have a higher probability of finding the correct nonce. Once a miner finds a valid nonce and successfully hashes the candidate block to meet the required criteria, they broadcast the solution to the network. Other nodes in the network can easily verify the correctness of the solution by recomputing the hash. If the solution is valid, the new block is added to the blockchain, and the miner is rewarded with cryptocurrency tokens (e.g., Bitcoin) and any transaction fees included in the block. There is no committee in PoW, the chain may happen to have several blocks at the same height at some point, which is referred to as a fork. With a non-zero probability, one chain will eventually outpace the other and the honest nodes will only mine and extent on the longest chain, as per what is known as the fork choice rule.

PoW is known for its robust security and simplicity of implementation. For an attacker to alter a block's contents or create fraudulent transactions, they would need to control a majority of the network's computational power, which is extremely difficult and costly. PoW is also the only self-healing (Badertscher et al., 2020) consensus mechanism known to date for blockchain systems. The byzantine security threshold for PoW usually lies at 51% of the total computational power. If a participant or group of participants goes beyond this threshold, they can decide which chain will become the longest one by simply dedicating all their computational power to mine on it. Reaching this threshold, while not impossible, is near impossible for a well established chain such as Bitcoin.

PoW encourages decentralization because anyone with computational resources can participate in the mining process, making it challenging for a single entity to monopolize the network. PoW rewards miners for their computational effort, and the process of finding the nonce is theoretically a fair competition. Miners are incentivized to play because they are assumed to be rational and have invested resources in mining equipment and electricity. Implementation details may open vulnerabilities that rational miners will naturally exploit as they are looking to increase their profitability. The most well known example of such an issue is the Selfish Mining (Eyal and Sirer, 2014) strategy.

However, PoW also has some notable disadvantages, including high energy consumption due to the intensive computational work involved, which has led to environmental concerns. As a response to these concerns, alternative consensus mechanisms, like Proof of Stake (PoS), have been developed to reduce energy consumption while maintaining network security.

Proof of Stake (PoS)

Proof of Stake (PoS) is a consensus mechanism used in blockchain networks such as Ethereum 2.0 (Buterin et al., 2020) and Tendermint (Kwon, 2014), as an alternative to Proof of Work (PoW). While PoW relies on computational power and energy-intensive mining, PoS is designed to be more energy-efficient and environmentally friendly.

In PoS, validators (often referred to as "stakers") are chosen to create and validate new blocks based on the amount of cryptocurrency tokens they hold and are willing to "stake" as collateral. So, validators are selected to create new blocks and validate transactions based on their stake in the network. The more tokens a participant is willing to "lock up" as collateral, the higher the chance they have of being chosen as a validator.

PoS is designed to be secure against various types of attacks, including the infamous "51% attack" that can affect PoW-based networks. In a PoS system, an attacker would need to acquire a majority of the network's total staked tokens, which can be prohibitively expensive and challenging.

PoS is often considered more environmentally friendly than PoW, as it does not require the massive computational power and energy consumption associated with mining.

PoS encourages decentralization by allowing anyone with a stake to participate in block creation and validation. It does not favor those with expensive mining equipment, but is arguably less accessible than PoW, since users willing to participate must buy the collateral.

PoS provides robust security against certain types of attacks, making the network resistant to control by a single malicious entity. Validators in PoS systems are economically incentivized to act honestly and in the best interest of the network, as they have a financial interest in the system's long term stability. Failing to do so may lead to some of their collateral to be confiscated.

While PoS offers several advantages, it is not without its challenges and considerations. These may include issues related to initial token distribution, potential centralization of wealth, and the need for mechanisms to prevent "nothing at stake" problem (Li et al., 2017). PoS is a significant departure from PoW, and its effectiveness depends on the specific design and rules of the blockchain network implementing it. It is worth noting that major blockchain networks, such as Ethereum, are transitioning from PoW to PoS to reduce their environmental impact and improve scalability.

2.3.6 Incentives

Public blockchains must provide incentives for their participant to stay active and respect the agreed upon rules of the system. Those incentives mostly consist in the block reward, and the transaction fees.

The block reward, also referred to as coinbase transaction, is a reward given to the producer of a given block for its work to extend the blockchain. This block reward is defined in the algorithm, so that its modification is often a significant event, since this directly impacts the economic incentive of the participants maintaining and extending the blockchain. In the Bitcoin network, such an event is known as a halving (Meynkhard, 2019). To improve their block creation capabilities, block proposers may *invest* either in hardware, capital or other agents depending on the consensus mechanism used. For example, in a PoW blockchain they can invest in new hardware since the more computational power they have, the easier they can create blocks. In a PoS blockchain, they can invest in the stakes, since the more locked stakes they have, the more chance to enter the committee. In addition to PoS, in DPoS blockchains, they can also invest in other block proposers by delegating their stakes.

On the other hand, transaction fees depend on the network activity. So, users who want their transactions to be processed within a defined period of time will have to provide a greater fee under high network load. This competition between non-contributing users gives rise to what is known as the "fee market".

Since block producers are rational, they ought maximize their profit by both increasing their block creation chances and to select the pending transactions with the highest fees.

This thesis heavily focuses on the notion of incentive within a system of rational participants. Any flaw in the incentive design may encourage unwanted behaviors that can ultimately have devastating consequences for the system.

2.4 Reinforcement Learning (RL)

Reinforcement Learning (RL) (Sutton and Barto, 2018) is a type of machine learning, not requiring prior / expert knowledge, where an agent learns to make sequences of decisions by interacting with an environment. It is primarily used in scenarios where the agent's actions influence its future rewards.

RL is highly versatile and can be used in various use-cases from Chess and Go (Silver et al., 2016), self-driving vehicles (Liang et al., 2018) or even supply chain management (Giannoccaro and Pontrandolfo, 2002).

The growth in popularity of RL in the last 15 years can indubitably be attributed to innovations leading to the birth of deep reinforcement learning (Mnih et al., 2013).

2.4.1 Markov Decision Process (MDP)

At the core of RL is the Markov Decision Process, which consists of the following components:

- **State Space ($\mathcal{S}$)**: The set of all possible states in the environment.
- **Action Space ($\mathcal{A}$)**: The set of all possible actions the agent can take.
- **Transition Probability Function (P)**: Describes the probability of transitioning from one state to another after taking a particular action.
- **Reward Function (R)**: Provides a numeric reward for each state-action pair.

The agent interacts with the environment by taking actions, and its goal is to find a policy that maximizes its reward.

A policy (π) is a strategy that maps states to actions. It can be deterministic or stochastic. The goal of RL is to find the optimal policy, denoted as π, which maximizes the expected cumulative reward G over a potentially infinite horizon of interactions with the MDP.

$$G = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(a_t, s_t) P(s_{t+1}|s_t, a)] \tag{2.1}$$

Knowing that a_t is given by the current policy π, the previous equation can be rewritten to:

$$G = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(\pi(s_t), s_t) P(s_{t+1}|s_t, a)] \tag{2.2}$$

Where $\gamma \in [0, 1]$ is called the discount factor: it controls the horizon at which rewards are taken into account. A γ value of 1 means that all rewards are considered regardless of how far they are. A lower γ value restrict the rewards being considered to a finite horizon.

2.4.2 Estimating the expected rewards with a Value Function

There exist two value functions in RL:

State-Value Function

The state-value function $V_\pi(s)$ represents the expected cumulative reward when following policy π starting from state s. It can be computed using the Bellman equation:

$$V^\pi(s) = \sum_a \pi(a|s) \left(R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^\pi(s') \right) \tag{2.3}$$

Action-Value Function (Q^π)

The action-value function $Q^\pi(s, a)$ represents the expected cumulative reward when starting from state s, taking action a, and then following policy π. It can also be computed using the Bellman equation:

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \sum_{a'} \pi(a'|s') Q^\pi(s', a') \tag{2.4}$$

2.4.3 The Need For A Model

Several algorithms were created using only the definitions above. The most well known ones are arguably value iteration (Bellman, 1957) and policy iteration (Howard, 1960). Still, they do have a strong constraint for real world applications: the underlying MDP must be known. They are therefore classified as model-based approaches.

In many potential applications, the model is either too complex to be modeled as an MDP or simply unknown.

A second kind of approach, called model-free, aims at learning the policy π that maximizes G without relying on a model of the environment. One of the first successful model-free algorithm is Q-Learning (Watkins and Dayan, 1992).

2.4.4 Model Free Algorithms

Q-Learning (Watkins and Dayan, 1992) is a popular tabular RL algorithm for finding an optimal deterministic policy without relying on a model of the environment. It only uses the Q value function and updates it repeatedly as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right) \tag{2.5}$$

Where α is the learning rate, which defines how much the current estimate $Q(s,a)$ is to be updated according to the error $R(s,a) - Q(s,a)$ and the expected reward of following the current policy $\pi(s) = \arg\max_a Q(s,a)$.

This algorithm is said to be **off-policy**, because the learning step samples the next action from a different policy, $\max_{a'} Q(s',a')$, than the one currently being learned.

The SARSA (Rummery and Niranjan, 1994) algorithm is very similar with the key difference that it is **on-policy** as it does not sample the next action greedily, but strictly follows the current policy with $Q(s',a')$.

Both Q-Learning and SARSA have been proven to converge (Watkins and Dayan, 1992) as long as all (s,a) tuples are eventually sampled, which is equivalent to say that it converges as long as the algorithm is given a chance to fully **explore** the environment it is trying to learn.

The practical solution is to follow an ϵ greedy policy: a random action is sampled with probability ϵ, and the greedy policy is followed with probability $1-\epsilon$, leading to each state-action pair being visited.

This poses the crucial question of when to stop **exploring** and start **exploiting** a policy in an unknown environment, also known as the Exploration vs. Exploitation dilemma.

2.4.5 Exploration vs. Exploitation

Balancing exploration (*i.e.,* trying new actions) and exploitation (*i.e.,* choosing actions with known high rewards) is a fundamental challenge in RL (Ishii et al., 2002). Given an infinite amount of time, it is equivalent to knowing when an individual or algorithm has nothing more to learn from its environment. In a more realistic and constrained time setup, this is a trade-off that must be carefully managed to maximize performance while minimizing the time spent learning from the environment.

2.4.6 Continuous State Space

The aforementioned algorithms are all tabular, they maintain a table of N states per M actions and update the estimates $Q(s,a)$ during the learning process. This approach is impractical for large spaces and simply impossible for continuous ones. The rise of deep neural networks allowed a first innovation, which was to represent the Q function by a neural network under the name of Deep Q Learning (DQN) (Mnih et al., 2013).

The neural network is repeatedly optimized to minimize a sequence of loss functions, each representing an expected cumulative reward error for a particular step.

Given that the Q function can now be a differentiable deep neural network, it is thus possible it use continuous actions with the DDPG (Lillicrap et al., 2016) algorithm.

This algorithm combines ideas from both policy-based and value-based approaches in what is known as the Actor-Critic method (Grondman et al., 2012).

The implicit policy $\pi(s)$ is replaced by a neural network (i.e., the Actor) that is trained through experiences so that it minimizes the bellman error from the Q function (i.e, the Critic).

2.5 Multi-Agent Reinforcement Learning (MARL)

Multi-Agent Reinforcement Learning (MARL) extends RL to scenarios with multiple interacting agents, each with their own goals and actions. In MARL, agents learn to collaborate or compete in a shared environment. This gives rise to new difficulties and instabilities, given that the actions of one agent can influence the performance of all others (Busoniu et al., 2008).

2.5.1 Challenges in MARL

The introduction of one or several additional agents, cooperating or competing, has a significant impact on the learning process. Since any individual agent is no longer the only one to influence the environment, its performance can be degraded without any changes to its policy, because of others' actions. This is known as non-stationarity (Papoudakis et al., 2019).

Non-stationarity can lead to difficult learning processes or even oscillating behaviors, sometimes preventing convergence because of the overly complex multi-agent interactions (Matignon et al., 2012; Zhang et al., 2021).

2.5.2 Approaches to MARL

MARL can be approached in several ways depending on the problem at hand.

The learning can be centralized (*i.e.*, agents share a single critic and / or policy, akin to a hive mind) or decentralized (*i.e.*, agents are independent entities with their own policies) (Lyu et al., 2021).

The environment may require cooperation, competition or a mix of both, involving some form of communication and coordination between policies that are being optimized in parallel.

The algorithms can be tailored for specific settings such as MADDPG (Lowe et al., 2017), which is a centralized, synchronous, cooperative, multi-agent version of DDPG that aims to alleviate non-linearity by providing each agent knowledge about the current Critic of all others.

In the scope of this thesis, the synchronicity assumption does not hold. Decentralized systems such as blockchains are asynchronous by nature, and we will focus either on decentralized MARL with no common knowledge assumptions, also known as Independent Q-Learning, or on hive mind approaches when different agents share a single policy.

2.5.3 Applications of MARL

MARL has a wide range of applications in various fields such as :

Robotics : where MARL is used extensively to coordinate groups of robots. A prime example if this would be swarm robotics where multiple robots collaborate on a specific task consisting in exploration, mapping or search-and-rescue (Blais and Akhloufi, 2023). Autonomous and semi autonomous vehicle: make use of MARL for complex decision making in multi-agent traffic scenarios (Zhou et al., 2021).

The gaming and simulation industry : benefits from MARL, which has been used to develop agents that can compete or cooperate in complex multi-player games such as Dota 2 (OpenAI et al., 2019).

Resource management: where MARL can be used to optimize both resources allocation and usage. This includes applications in power grid management (Chen et al., 2021), where agents manage energy production and distribution. and is easily transposable to the digital world with network traffic management (Foerster et al., 2016), where agents control the flow of data through a network, maximizing throughput and minimizing latency.

Economics and finance: also apply MARL to model and simulate interactions between different market participants (Liu et al., 2022).

Smart cities and Internet of Things(IoT) : It is used to manage urban infrastructure, such as traffic light control (Damadam et al., 2022) or public transportation systems (Chen et al., 2016), where multiple independent agents represented by sensors or vehicles need to make real-time decisions.

Chapter 3

AGR4BS: Agent, Group, Roles for Blockchain Systems

In this chapter, we discuss the design choices and definition of AGR4BS (Roussille et al., 2022), a generic blockchain model intended to solve the modeling complexity when faced with blockchain systems as numerous as they are diverse. AGR4BS will be used as the groundwork for the subsequent contributions of this thesis. Understanding this modeling is paramount in order to fully grasp the taxonomy and the design of the simulator that it inspired, both of which will be defined in the following chapters.

3.1 Related Work

A *model* is an abstraction of some aspects of an existing or planned system. Models serve particular purposes, that is, for example, to present a human-understandable description of some aspect of a system or information in a form that can be efficiently analyzed. In the blockchain context, there are very few studies that directly target the modeling issue at a high level of abstraction (Gürcan, 2020; Gürcan, 2019).

This does not mean that there is no model for blockchain systems. Based on the existing studies in the blockchain literature, we identified the following modeling paradigms: *process*-oriented, *object*-oriented, *graph-theoretic* and *agent*-oriented paradigms. In the following, we describe each paradigm by showing how they model participants (*i.e.,* users and Block Proposers), interactions, behaviors, and data structures (*e.g.,* blockchains, transactions) using the abstractions they provide.

3.1.1 Process-Oriented Paradigm

In process-oriented paradigm (aka distributed programming paradigm), a system encompasses *multiple distributed processes*[1] connected with *communication links* (aka channels) that *cooperate* on some *common* task (*e.g., shared memory* or *consensus*) (Cachin et al., 2011):

- *Processes* execute the distributed algorithm assigned to them through a set of components implementing the algorithm within these processes.
- *Channels* allow processes to broadcast messages by triggering events.
- A *shared memory* allows local direct access to a resource from possibly many processes.

[1] A *process* abstraction may represent a physical or virtual computer, a processor within a computer, or a specific thread of execution in a concurrent system.

- *Consensus* mechanisms aim at providing a way for processes to agree on a common decision / outcome under a decentralized framework.

This paradigm aims at building and/or analyzing systems which are *dependable* (offering reliability and security) and have *predictable behavior* even under negative influence from the environment (offering tolerance to faults).

Many studies use this paradigm to analyze[2] and/or build[3] blockchains using the following related abstractions: participants (*e.g.,* users and Block Proposers) are modeled as *processes*, interactions as *channels*, the blockchain as a *shared memory* and deciding on a common block is represented by using *consensus* abstractions.

For instance, (Eyal and Sirer, 2014) shows that some *Block Proposers* in a Bitcoin blockchain (aka miners) can *deviate* from their nominal behaviors, thus acquiring an unfair advantage which consequently decreases the *dependability* of the system.

(Neuder et al., 2020) builds and improves on both (Eyal and Sirer, 2014) and (Sapirshtein et al., 2016) by providing a hybrid strategy deviating at both the block creation and the networking levels, effectively achieving the unfair advantage for miners while leading other agents to work for the *deviating* miners.

3.1.2 Graph-Theoretic Paradigm

The Graph-Theoretic Paradigm focuses on topology and therefore on connective properties of algebraic / mathematical objects. In the context of distributed computing, these objects are generalizations of graphs, and their connectivity properties related to the computability of distributed algorithms.

Exploiting certain topological properties of higher dimensional geometric objects to prove results of distributed algorithms is referred to as the topological approach to distributed computing. Techniques from combinatorial and algebraic topology have advanced characterization of synchronous and asynchronous distributed algorithms, as well as their solvability (Alpern and Schneider, 1985; Herlihy and Rajsbaum, 1995; Nowak, 2010; Saks and Zaharoglou, 1993). In graph theory, a vertex is a point in a graph. Vertices are linked together by edges that represent a relation between two vertices.

In this paradigm, the participants are modeled by using *vertices*, the transactions using *edges*, the interactions using *simplex* and/or *face* abstractions and frauds as *spatio-temporal pattern* abstractions.

The ability of sheaf-theoretic frameworks to decipher global information from local information has led to a diversity of applications, such as those that have been further proposed to model concurrent processes in distributed systems (Malcolm, 2009), semantics for object-oriented programming languages (Wolfram and Goguen, 1991) and representations of information systems (Sagar and Kishore, 2019). In (Meldman-Floch, 2018), the author explores topological models of distributed computing for scalability focused Blockchain technologies. To do so, the author models a block as a sheaf and develops a theory for distributed consensus protocols.

[2](Anceaume et al., 2019; Decker et al., 2016; Eyal and Sirer, 2014; Garay et al., 2015; Neuder et al., 2020; Sapirshtein et al., 2016)

[3](Androulaki et al., 2018; Gilad et al., 2017; Herlihy, 2018; Kwon, 2014; Nakamoto, 2008; Wood, 2014)

3.1.3 Object-Oriented Paradigm

In object-oriented paradigm, a system is composed of *multiple objects* (instances of classes) interacting with local or remote *method invocations* (message passing) (Larman, 2004).

Classes have specific *responsibilities* and encapsulate *data* (in the form of *attribute* abstractions) and *code* (in the form of *method* abstractions) that manipulates these data, and are related to each other using *association* abstractions.

This paradigm aims at building and/or analyzing systems which are *evolvable* (*i.e.*, easy to extend) and *maintainable* (*i.e.*, easy to fix).

Few studies explicitly use this paradigm in the blockchain literature. Examples are (Alharby and Moorsel, 2020; Marchesi et al., 2020; Rocha and Ducasse, 2018)[4]. In these studies, the participants are modeled as *class*, interactions as *associations*, the blockchain as a *class* and deciding on a common block as *method* abstractions.

For instance, (Alharby and Moorsel, 2020) proposes a generic PoW blockchain model with the aim of building an extensible blockchain simulator. Currently, they are able to model and simulate both Bitcoin and Ethereum 1.0 and validate their results against historical data. They are compatible with PoS blockchains, given a few modifications. As another example, (Marchesi et al., 2020) proposes a notation based on UML (Filho and Braga, 2017) for supporting smart contract design. Concretely, they add stereotypes for UML Class and Sequence diagrams to better express the entities and the interactions between them.

3.1.4 Agent-Oriented Paradigm

In an agent-oriented perspective, a system is composed of *multiple autonomous agents* that are able to perceive their *environment*, reason independently (either reactively or proactively) and act upon their environments (Ferber, 1999; Wooldridge, 2009), thus forming a so-called Multi-Agent System (MAS). This paradigm especially aims at building and/or analyzing systems which have some degree of *openness*, *autonomy*, *intelligence* and *complexity*.

An agent is an entity of the system that is relying on some degree of autonomy in order to pursue its goal or fulfill its functionality, either passively or actively. Coordination is the mean by which agents exchange information or resources in the pursuit of an objective.

MAS modeling can be considered according to two main perspectives: *agent-centric* and *organization-centric*. While the former focuses on the agent's internal architecture, the latter points on the structure of the system, and firstly considers agents as empty shell, in order to thus focus on the MAS organizational aspects.

Many studies use this paradigm in an agent-centric sense to analyze blockchains (Amoussou-Guenou et al., 2020; Ciatto et al., 2020a,b; Hou et al., 2021; Toroghi Haghighat and Shajari, 2019; Wang et al., 2021a; Zhang et al., 2020; Zhang et al., 2019). In these studies, the participants are modeled as *agents*, interactions as *coordination* abstractions, the blockchain is modeled as a *shared knowledge* and deciding on a common block is modeled by using *goal*, *strategy* or *game* abstractions.

As an example, (Hou et al., 2021) takes a generic Reinforcement Learning (RL) approach to detect attacks on different blockchain systems through RL based simulations and strategy search, while being fairly constrained to the block creation

[4] In fact, (Rocha and Ducasse, 2018) uses the Entity Relationship (ER) model, but since ER is a *mental model* which is similar to object-orientation, we grouped them in the same category.

processes in PoW blockchains as well. The aim is similar to (Eyal and Sirer, 2014), that is, discovering attack vector and weaknesses in the blockchain.

While this search is experimental in contrast to analytical, it is able to grasp some of the multi-agent interactions and limitations arising from many heterogeneous agents each pursuing a specific goal.

As another example, (Zhang et al., 2020) focuses on meta-agents, that is agents acting on several blockchain hyper parameters to balance a security/throughput trade-off as a way to optimize the blockchain performance while preserving the security and dependability of the system through RL.

3.1.5 Discussion

Models provide abstractions used to represent and communicate what is important, without unnecessary detail, and help to cope with the complexity of the problem studied or the solution developed. Consequently, it is crucial to use an *adequate* modeling approach. Considering the previous literature, one can see that all existing blockchain modeling approaches are elaborated having in mind some specific aspects and /or problems of blockchain systems (*e.g.,* network topology targeted studies tend to use the graph-theoretic paradigm), which in turn makes them specific so that they cannot be applied to others easily.

3.2 Organization-centric Modeling for Blockchain Systems

In this section, we first describe the motivations behind using an organization-centric modeling for blockchain systems (Section 3.2.1), then present the chosen organizational model, namely Agent/Group/Role (AGR), for defining our generic organizational model for blockchain systems (Section 3.2.2) and finally describe the methodology we used for applying AGR (Section 3.2.3).

3.2.1 Motivations behind Organization-centric Modeling

Since blockchain systems are social systems (as shown in Chapter 2), organizational modeling provides relevant abstractions with respect to what blockchain systems actually are.

Indeed, organization-centric modeling *abstracts away* the internal details (*i.e.*, the cognitive capabilities) of agents, and thus allows *focusing* on the structural, organizational and social dimensions of blockchain systems, that is on what relates the structure of an organization to the externally observable agent behaviors.

Representing blockchain systems using the *organization* abstraction allows agents to cooperate with each other by defining common cooperation schemes like responsibilities, groups, protocol and global tasks. For example, deciding on a common block on a blockchain system is an institutional action only possible because the blockchain system defines the rules that must be followed to do so.

Additionally, *norms* can be used to constraint the behaviors of independent agents towards the global goal of the organization. In other words, when an agent adopts a role, it adopts a set of behavioral constraints supporting the global purpose of the organization. It is then up to the agent to obey or disobey these constraints. For instance, in a blockchain system, when an agent adopts the *user* role, it adopts the behavioral constraints of preparing proper transactions and validating all the data before relaying.

Having a *specification* of the organization allows agents to reason about it. That is to say, the agents can decide whether to join or leave organizations during their lifetime, can change/adapt their current organizations, and can decide to obey/disobey the norms of the organization.

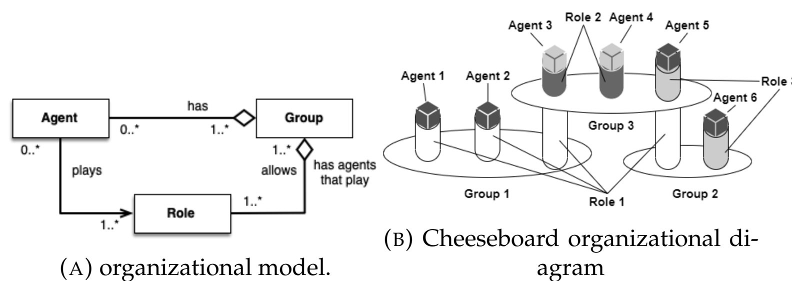
Moreover, such an organizational specification may also enable the organization to reason about itself and about the agents to ensure the achievement of its global purpose. That is to say, organizations can decide to let agents join/leave during execution, they can let agents change/adapt their current organizations, and they can govern the agents' behaviors (*i.e.,* monitor, enforce, regiment).

3.2.2 The Agent/Group/Role (AGR) Approach

Among the organization-centric multi-agent-oriented approaches proposed in the literature[5], the Agent/Group/Role (AGR) approach proposed in (Ferber et al., 2004) is a good fit for our motivations and purpose.

Especially, AGR describes what is a MAS organization at a high level of abstraction and is thus very flexible and open to various interaction schemes and organizational designs.

The AGR model (Figure 3.1) is based on three first-class abstractions: *agent*, *group* and *role* (Figure 3.1a). Those abstractions are composable and interact with each other (Figure 3.1b).



(A) organizational model.

(B) Cheeseboard organizational diagram

FIGURE 3.1: Agent/Group/Role representations as a conceptual model (a) and as a cheeseboard diagram (b).

Roles are abstract representations of functional positions of agents in a group. A role describes the responsibilities associated to it, the constraints that agents need to satisfy in order to obtain that role, and the benefits that agents would obtain by playing that role.

Groups identify contexts for patterns of activities (*i.e.,* roles) that can be shared by sets of agents (*i.e.,* they group together agents working together). Agents may communicate, if and only if, they belong to the same group. Groups are *organizational structures* where the *interactions* make an aggregate of agents a functionally coherent whole. Moreover, groups may establish boundaries as well. Agents that do not belong to a group may not know its structure.

Agents are active, communicating entities playing *roles* within *groups*. Agents play at least one role in a group, but may hold multiple roles and be a member of multiple groups as well. However, *no constraints* are placed upon the architectures, the cognitive abilities and/or the mental issues of agents.

[5](Abbas, 2015; Criado et al., 2013; Dignum et al., 2005; Ferber et al., 2004; Giorgini et al., 2006; Hübner et al., 2002).

3.2.3 The Methodology for AGR

In this subsection, we briefly describe the process we use in Section 3.3 to design the generic organization model based on the AGR approach.

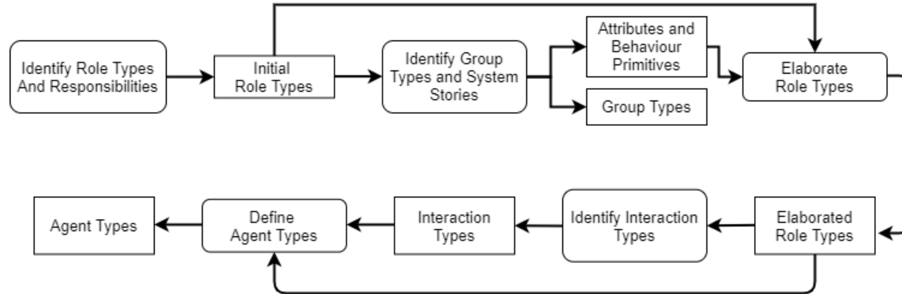


FIGURE 3.2: Proposed methodology for defining our generic organizational model.

Figure 3.2 shows the workflow we use to define our organizational model of blockchain systems. Our approach is similar to what is presented in (Rodriguez et al., 2021), with a system point of view on functionalities through *System Stories* which are inspired by the AGILE development cycle and help define and refine system requirements. However, unlike (Rodriguez et al., 2021) we do not restrict ourselves to a particular role or agent.

We first define the roles which are part of the system, *i.e.*, what high-level functionalities must be present in the system. From those role definitions, we infer the different groups and how the roles are grouping together to achieve their goals. When those two steps are done, we elaborate on the roles and define their behaviors, *i.e.*, what low-level functionality must be present to fulfill the high-level ones. Next, we define how the roles interact with one another inside a group, *i.e.*, what needs to be communicated and how it is done. Finally, we define the agent types, which interaction types they can have and the roles they can play in the system.

3.3 AGR4BS: A Generic Organizational Model for Blockchain Systems

Using the AGR approach, we hereby propose a generic organizational model for blockchain systems that acts as a basis for the definition of several concrete blockchain systems, namely AGR4BS. This way, it is possible to build and/or analyze concrete blockchain systems which reside at the agent level, *i.e.*, where agents with different cognitive abilities may interact. This allows for a clear division of the different building blocks of blockchain systems, while leaving the possibility to explore behavioral divergence in a well-defined framework.

To this end, we identify all possible roles and their corresponding nominal (honest) behaviors applicable to all types of blockchain systems. The agents participating in blockchain systems may play one or several generic roles listed below, in possibly more than one blockchain system at the same time.

3.3.1 Role Types

With respect to existing blockchain systems, we identified nine generic role types (Figure 3.3). In the following, we carefully assign responsibilities to these role types.

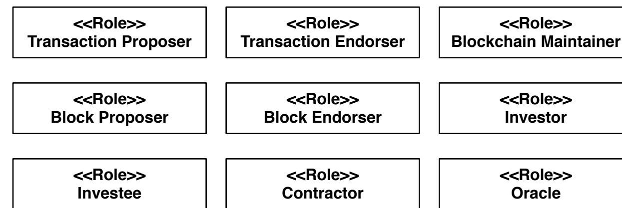


FIGURE 3.3: The initial generic blockchain role type model.

Here are the responsibilities for each role :

- Transaction proposer : proposes transactions
- Transaction endorser : endorse endorse the proposed transactions
- Block Proposer : creates and proposes blocks
- Blockchain Maintainer : maintains a local copy of the blockchain and ensure it's integrity.
- Investor : invests financial or computational value in the system.
- Investee : receives investments and perform a task on behalf of its investor(s) in exchange for a commissions.
- Contractor : provides internal services to other participants on a contractual basis (*i.e.*, Smart Contracts)
- Oracle : provides external services and data feeds to the other participants.

3.3.2 Group Types

In the blockchain systems context, we identified two categories of generic types of blockchain groups applicable to any kind of blockchain system: Structural Groups and Interest Groups.

Structural Groups fulfill essential functions of the blockchain system, and all agents are aware of the existence of these groups. We identified two types of structural groups: *Transaction Management* and *Block Management*.

Interest groups are composed of agents increasing the quality of one or several properties of the blockchain such as scalability, throughput, security, or reward variance. Interest groups are therefore not structural (*i.e.*, non-essential) for the overall blockchain, and their existence is not necessarily known by all participants.

In the following, we give the specification of each group in terms of roles and their related behavioral primitives.

Structural Group: Transaction Management

This group is responsible for the way transactions in a blockchain network are processed. It is composed of four roles: *Transaction Proposer*, *Transaction Endorser*, *Blockchain Maintainer* and *Contractor*. Figure 3.4 represents the organizational structure and behaviors of this group by visualizing and relating roles and behaviors[6]. There are

[6]Note that, the interaction protocols can be represented by any sort of interaction diagram (such as UML sequence, Petri nets, finite state automaton and so on) in a concrete organization (*i.e.*, blockchain system) level.

three high-level meaningful behaviors: *Propose transaction*, *Validate transaction* and *Execute transaction*.

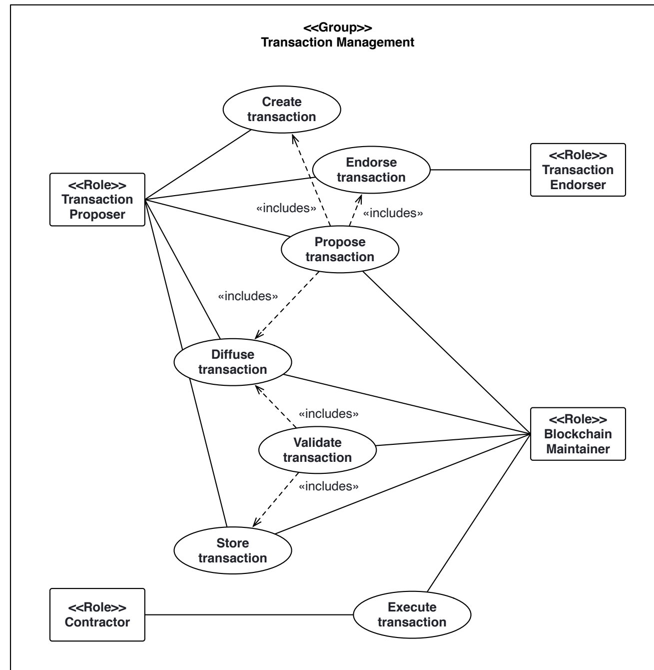


FIGURE 3.4: The organizational structure and behavior of the Transaction Management group.

The high-level nominal scenario of *Propose transaction* is as follows:

1. *Transaction Proposer* aims to transfer a value and thus creates a *transaction* by carefully choosing inputs, outputs, and a fee.
2. *Transaction Proposer* asks *Transaction Endorser(s)* to validate the transaction.
3. *Transaction Endorser(s)* decide(s) to endorse the transaction using a *transaction endorsement policy* and send(s) the endorsement result(s) to the *Transaction Proposer*.
4. *Transaction Proposer* proposes the transaction by diffusing it to *Blockchain Maintainers*.

The high-level nominal scenario of *Validate transaction* is as follows:

1. *Blockchain maintainer* validates the transaction against the local copy of the *blockchain*.
2. *Blockchain maintainer* stores the transaction in its *memory pool* if it is valid.
3. *Blockchain maintainer* diffuses the transaction by sending it to the neighboring *Blockchain maintainers*.

Execute transaction executes a transaction to invoke a *Contractor* behavior.

Structural Group: Block Management

This group is responsible for the way blocks in a blockchain network are processed. Figure 3.5 represents the organization structure of this group by visualizing and relating roles and behaviors. It is composed of three roles: *Block Proposer*, *Block Endorser* and *Blockchain Maintainer*. The interactions shown in this figure covers the

principal aspects such as transaction selection, block creation, block endorsement, block proposal, block diffusion, block validation and block appending that relate agents through their roles.

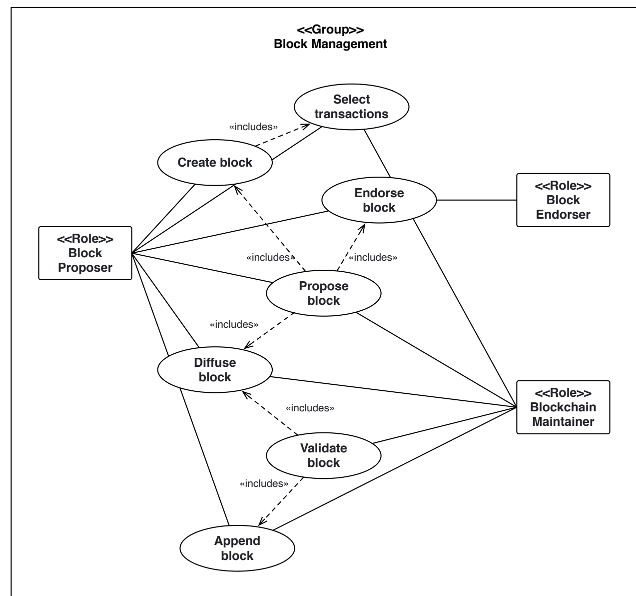


FIGURE 3.5: The organizational structure and behaviors of the Block Management group.

The high-level nominal scenario of *Propose block* is as follows:

1. *Block Proposer* selects transactions from *Blockchain Maintainer* using a selection strategy.
2. *Block Proposer* tries to create a block using the selected transactions.
3. *Block endorser(s)* decide(s) to endorse a confirmed block (*i.e.*, a block that is already in the blockchain) as the parent block of the new block using a *block endorsement policy*.
4. *Block Proposer* proposes the block by diffusing it to *Blockchain maintainers*.

The high-level nominal scenario of *Validate block* is as follows:

1. *Blockchain maintainer* validates the block against its local copy of the *blockchain*.
2. If the block is valid, *Blockchain maintainer* either
   (a) appends the block to its blockchain if its parent is also in the blockchain
   (b) or (if it is an orphan) stores the block in its *memory pool*.
3. *Blockchain maintainer* diffuses the block by sending it to the neighboring *Blockchain maintainers*.

Interest Group: Pool

This group is responsible for bringing together investors and investees on a blockchain system. It is composed of two roles: *Investor* and *Investee*. Figure 3.6 represents the

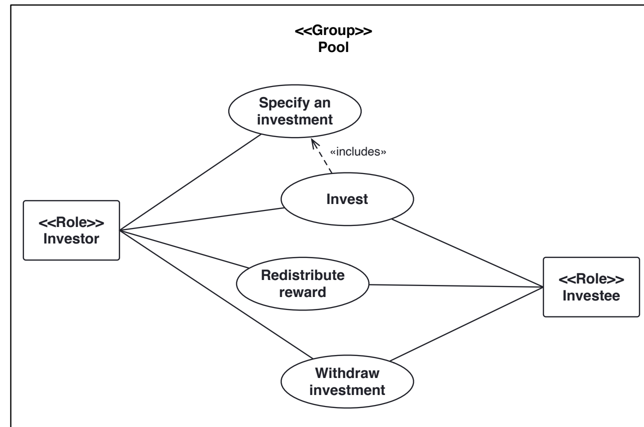


FIGURE 3.6: The organizational structure and behavior of the Pool group.

organizational structure of this group by visualizing and relating roles and behaviors. There are three high-level meaningful behaviors: Invest, Withdraw and Redistribute.

The high-level main success scenario of *Invest* is as follows:

1. *Investor* specifies an investment (*i.e.*, an investee and an amount of investment) based on its *incentives*.
2. *Investor* makes its investment.
3. *Investor* regularly receives the redistributed rewards.
4. At any time, *Investor* can decide to withdraw either a part of or all of its investment.

Withdraw sends the withdrawal request to the *Investee*, this request might be a transaction or an asynchronous message.

The *Redistribute* behavior sends transactions to the relevant investors rewarding them proportionally to their contribution.

Interest Group: Decentralized Application (DApp)

This group is responsible for any kind of *user-defined* transactional decentralized application (DApp) realized on the blockchain system. It is composed of *user-defined* roles, where at least one role should be *Contractor*.

In a Decentralized Application group, at least one role should be *Contractor*. A DApp may or may not interact with one or more *Oracle*.

3.3.3 Management of the Groups

As we will see in the Section 3.4, depending on the considered blockchain system, the membership of the different groups can be managed in two main ways:

- (1) explicit groups where the agents need to satisfy some *explicitly* well-defined criteria to enter, and
- (2) implicit groups where agents can enter without any check.

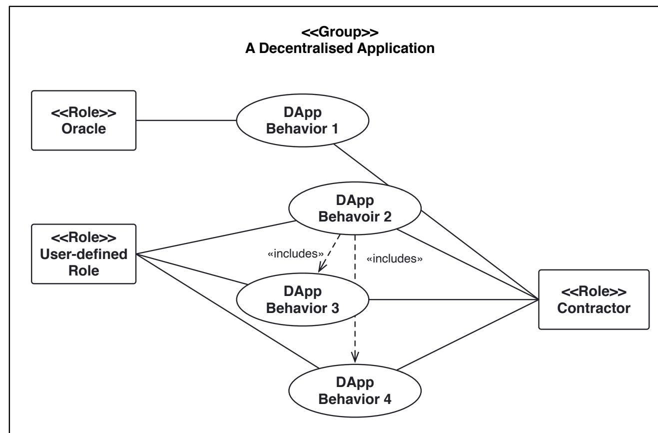


FIGURE 3.7: The organizational structure and behavior of a *user-defined* Decentralized Application (DApp) group.

Explicit groups may be in two forms: either they have (1) an agent playing the Group Manager role (see Figure 3.8) who is responsible for checking the conformity of agents to the specification of the structure and roles of the group and, authorizes or denies them to enter into the group (Ferber et al., 2004), or (2) the specification of the structure and roles of the group are immutably defined on the blockchain (*i.e.*, shared securely with everyone) and the agents can infer whether they can enter into the group or not. Implicit groups' specifications are not explicitly defined, and consequently agents form such groups in an emergent manner. In other words, the *specifications* of those groups are implicitly implemented inside each agent, consequently anyone can join or leave in the way they see fit.

FIGURE 3.8: The Group Manager role.

The agents are aware of the explicit groups (see Section 3.3.2). This means that the agents are aware of the other members in these groups and thus can *cooperate directly* with each other. This also means that, there are clear *specifications* about how to behave, join and/or leave these groups (*i.e.*, the agents can reason about these groups). However, it is not the case for implicit groups. In these groups, the agents are not necessarily aware of the other members and thus can by default only *cooperate indirectly* with each other.

3.3.4 Roles in Detail

Using the linguistic analysis technique[7] (Abbott, 1983) on the group descriptions, in this section we elaborate the role types by identifying their attributes and behavioral primitives for blockchain systems (Figure 3.9). Here, it should be noted that, there can be several underlying possible strategies associated with each behavioral primitive.

[7]In this technique, the nouns, and noun phrases in textual descriptions of a domain are identified and considered as candidate conceptual classes and/or attributes.

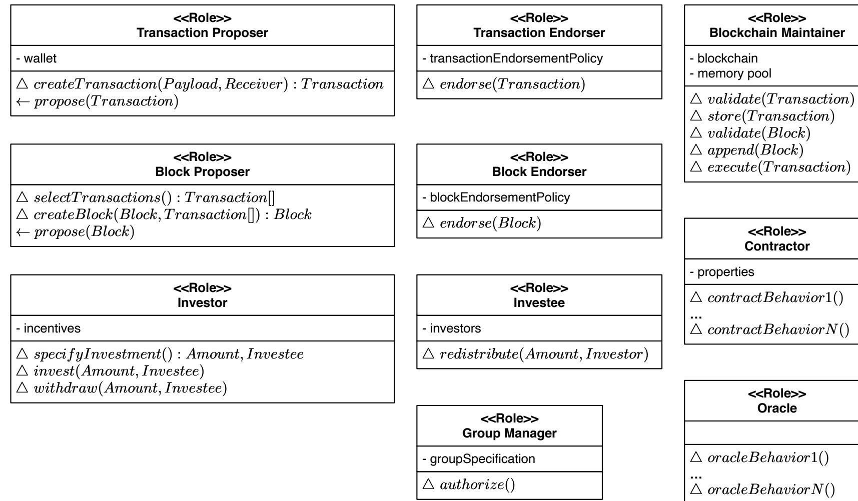


FIGURE 3.9: The roles and their corresponding attributes and behaviors for blockchain systems.

Transaction Proposer. When an agent aims to transfer a message, whether it is a financial amount, a new smart contract or simply data, it uses *createTransaction(Payload, Receiver)* to create a transaction by carefully choosing the payload, output, and a fee, and then broadcast it using *propose(Transaction)*. Typically, a simple transaction references previous transaction outputs as new transaction inputs and dedicates all input values to new outputs. Validating the correctness of these inputs and outputs against the blockchain falls into the responsibilities of the agent before diffusing its transaction.

Transaction Endorser. When an agent receives a transaction proposal, that is a transaction not already validated, it uses *endorse(Transaction)* in order to vouch for the transaction.

Blockchain Maintainer. When an agent aims to maintain a blockchain, upon receiving a block, it is responsible for carefully validating it, as well as the embedded transactions, using *validate(Block)* and *validate(Transaction)* respectively. Valid transactions are stored in the memory pool of the agent using *store(Transaction)*.

Here, there is no uncertainty since everything is crystal clear in both the blockchain and the memory pool. If there is some information missing, the agent simply waits for their arrival.

Upon validation, blocks are appended to the local blockchain using *append(Block)*. Valid blocks and transactions are diffused to the network to propagate the information using *diffuse(Block)* and *diffuse(Transaction)* respectively[8].

When an agent receives a transaction concerning a smart contract execution, it uses its *execute(Transaction)* behavior. Through the *getUnconfirmedTransactions()* other roles can request the pending transactions that are store in the Blockchain Maintainer's memory pool.

Block Proposer. When an agent aims to create blocks, it has three consecutive behaviors. The agent first uses *selectTransactions()* to carefully choose, from its memory pool, a set of unconfirmed transactions which is sufficient to fill a block while maximizing the total transaction fee. The agent then starts *createBlock(Block, Transaction[])* for creating a new block $h + 1$ which is linked to the last known head block h

[8]Note that the diffuse behaviors are available to every role and therefore not explicitly shown in our model.

in the *main chain* using a dedicated consensus algorithm. Upon successful creation, the agent uses *propose(Block)* to immediately broadcast it to the blockchain network.

Block Endorser. When an agent receives one or several block proposals for the same blockchain height, it uses *endorse(Block)* to choose one of them.

Investor. When an agent aims to increase its reward from the creation of blocks or invest in any entity / service of its choice, it uses *specifyInvestment()* to define the amount and target of the investment. The actual investment is done through the *invest(Amount, Investee)* behavior to carefully make an investment by taking into account its budget and the estimated return of its investment. If an investor wants to get part or all of its investment back, it can do so by using *withdraw(Amount, Investee)*.

Investee. An agent who is invested in by others uses *redistribute(Amount, Investor)* for carefully redistributing the obtained rewards to its investors on time.

Contractor. An agent implementing contractual behaviors will make use of its potentially many *contractBehavior()* to implement and provide a given functionality on the blockchain.

Oracle. An agent bridging the blockchain system with external systems (*i.e.,* Web services, other blockchain, etc. . .) will expose possibly many *oracleBehavior()* to provide the necessary communication medium so that other agents may exchange information with outside sources.

3.3.5 Interactions

Interactions are the means by which different roles exchange information or resources. The way roles interact in a system (*i.e.,* the way the interactions are realized) may have significant consequences on their behaviors. In the following, we identify and describe the possible interaction types found in blockchain systems in terms of *messaging* where a sending actor/object sends a message to a receiver actor/object and relies on that actor and its supporting infrastructure to then select and run some appropriate behavior.

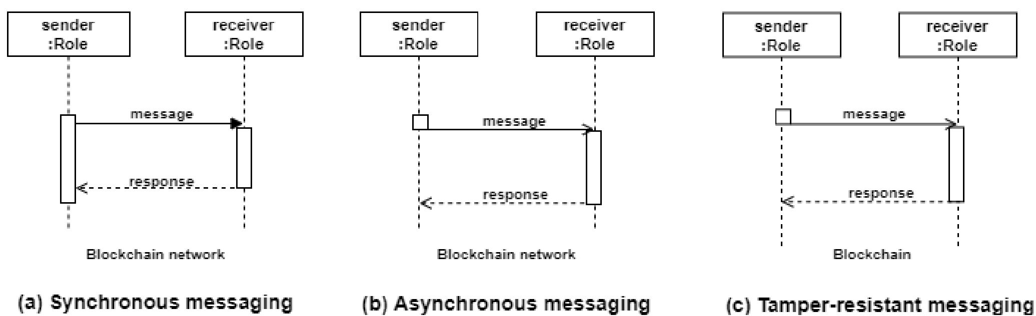


FIGURE 3.10: The interactions type for blockchain systems

- **Synchronous messaging** occurs between roles that are *communicating through the same blockchain network* (Figure 3.10a). Messages are delivered to the receiver and the sender's process is blocked till the receiver's process completes.

- **Asynchronous messaging** occurs between roles that communicate through the same blockchain network (Figure 3.10b). The message is sent to a queue where it is stored until the receiving role requests and then processes it. Meanwhile, the sender's process is not blocked.

- **Tamper-resistant messaging** relies on a blackboard communication scheme, using the replicated blockchain as a persistent medium (Figure 3.10c). It is a

kind of asynchronous messaging in which the sender publishes its message in the tamper-resistant replicated blockchain.

The generic interaction protocols given in Section 3.3.2 can be implemented in a concrete blockchain system by using the interaction formats given in this section. Different concrete realizations can use different interaction types depending on which agent plays which role.

3.3.6 Agent Types

We identified the following different types of agents that may exist in blockchain systems based on the agent definition given in Section 3.2.2: *Node* and *Smart Contract* (Figure 3.11).

Node agents are peers in the blockchain network that are deployed on a computer as a stand-alone software. They can communicate with node agents using *synchronous*, *asynchronous* and/or *tamper-resistant messaging*. However, they can only communicate with smart contract agents using *synchronous* or *tamper-resistant messaging* (Figure 3.11). Node agents take on responsibility such as maintenance, security, and dynamics of the blockchain system through the main generic roles : Transaction Proposer, Transaction Endorser, Block Proposer, Block Endorser, Blockchain Maintainer, Investee, Investor and Oracle. They ensure that the system is up and running and actively contribute to its main functions.

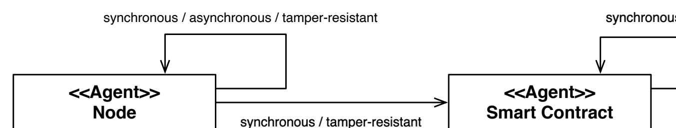


FIGURE 3.11: The agents for blockchain systems and their interaction models.

Smart Contract agents are immutable programs that are deployed on a blockchain data structure. Smart contract agents are not aware of their environment and are not able to observe it directly. Their communication scheme is more restricted as they can only communicate reactively through *Synchronous* or *Tamper-resistant messaging* with other agents.

They can *only* have *reactive* behaviors that are triggered by other agents. Smart Contract agents only play *Contractor*, *Group Manager* and *Investee* roles. Through the *Contractor* role, they can add functionalities to the blockchain system.

3.4 Case Studies

In this section, based on the roles and their corresponding nominal behaviors given in Section 3.3, we present how our generic model is able to represent the four different key blockchain technologies to date (*i.e.*, Bitcoin, Ethereum, Tendermint and Hyperledger Fabric). For each case study, we first give a specification of the system and then its organizational model. An example simulation implementation for the Bitcoin and Tendermint case studies is done using the MaDKit multi-agent platform[9] (Ferber et al., 2004) and is also available publicly[10].

[9]MaDKit, https://www.madkit.net, last access 13/09/2021.
[10]AGR4BS, https://gite.lirmm.fr/fmichel/agr4bs, last access 01/09/2021.

3.4.1 Bitcoin

Bitcoin has been created in 2008 as a peer to peer electronic cash system (Nakamoto, 2008), in other words it is a store of value or "digital gold". Bitcoin is an open and permissionless Proof-of-Work (PoW) based blockchain system. Bitcoin is revolutionary since it is the first blockchain system used globally. It provides a scripting language that allows its users to define scripts that are executing when some predefined conditions are met (Tschorsch and Scheuermann, 2016). However, this language lacks expressivity for hosting decentralized applications or complex services.

System Overview

There are mainly four types of entities in a Bitcoin blockchain system: users, miners, mining hardware and mining pools.

A user creates transactions with a set of input and output, as well as a fee to incentivize miners to process it. Then, it signs the transaction by itself (to endorse that the transaction is well-formed) and proposes the signed transaction to the blockchain system by relaying to its neighbors. When a user receives a proposed transaction from a neighbor, it first validates it against its local blockchain replica. If that transaction is valid, the user then adds it to the transactions memory pool and relays it to its own neighbors.

Miners are in charge of confirming the transactions proposed by users by organizing them as blocks. In return, they collect a static block reward and the totality of the fees of the selected transactions. For creating a new block, a miner first needs to select a head block as the valid head block to append its new one (*i.e.,* adding its hash inside its new block). This means choosing the head block of the longest chain or randomly among those of equal length (*i.e.,* the longest chain rule). Then it selects a set of unconfirmed transactions from its memory pool. Then the miner tries to solve a very hard cryptographic puzzle (only in a brute force manner) with a given difficulty using a dedicated mining hardware. This process, which is called PoW[11], requires spending a significant amount of energy and computational power to generate a desired hash value for the block. Upon success, the miner signs its block and finally proposes it to the whole blockchain system. Each participant receiving the proposed block, validates the block, appends it to the local blockchain and relays to its neighbors.

Mining hardware are dedicated hardware for hashing, mostly Application-Specific Integrated Circuits (ASICs) as of today. Miners must carefully weigh the costs of investing in mining hardware. A simple solution is comparing the purchase price and operating expenses (power, maintenance, rent, and so on), converted into BTCs, to the net mining returns in BTCs at the end of the machine's life (Taylor, 2017).

Another investment a miner can do to increase its chance of earning rewards is to join a mining pool. In a mining pool, miners mutualize their computing power to reduce the reward variance at a very small cost called the mining pool fee. There are basically two types of mining pools: centralized and decentralized. In the centralized setting, a mining pool leader distributes cryptographic workload among the pool members and collects the resulting block. The leader then shares the reward

[11] The PoW serves several purposes: (1) it protects the network against sybil attacks **where a malicious participant creates many identities in order to influence the consensus mechanism.** (2) it provides an election mechanism where the first miner solving the PoW is the de facto leader for the current height, and (3) it controls the growth rate of the blockchain by carefully setting the puzzle complexity to minimize the frequency of forks that are harmful for the blockchain. Every 2016 blocks, the difficulty is recomputed to match a target of approximately 10 minutes of mining required per block.

according to the distribution protocol of the pool (Romiti et al., 2019). In the decentralized setting, a smart contract of another blockchain system is used to regulate the redistribution of block rewards (due to the fact that Bitcoin currently does not support smart contracts). For example, in the P2POOL[12] mining pool, a side blockchain system is used for every contribution of its participants, and in SmartPool (Luu et al., 2017) an Ethereum smart contract is used to regulate the pool rewarding mechanism.

Organizational Model

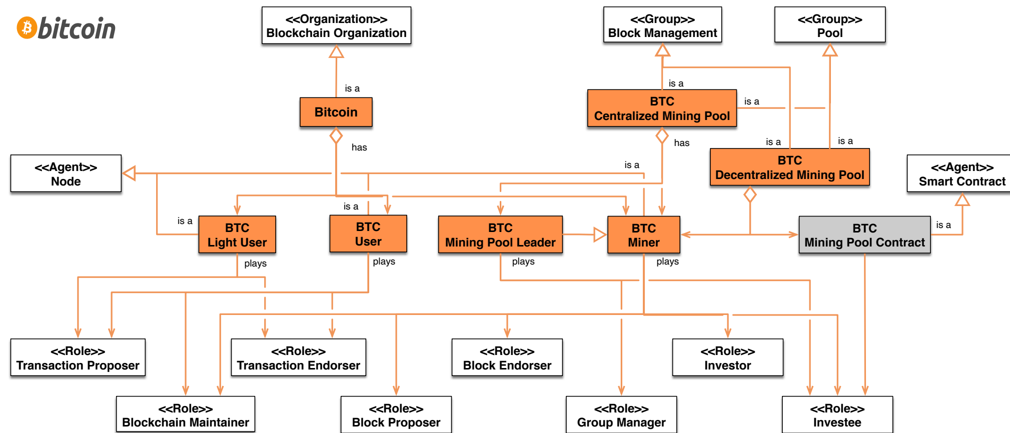
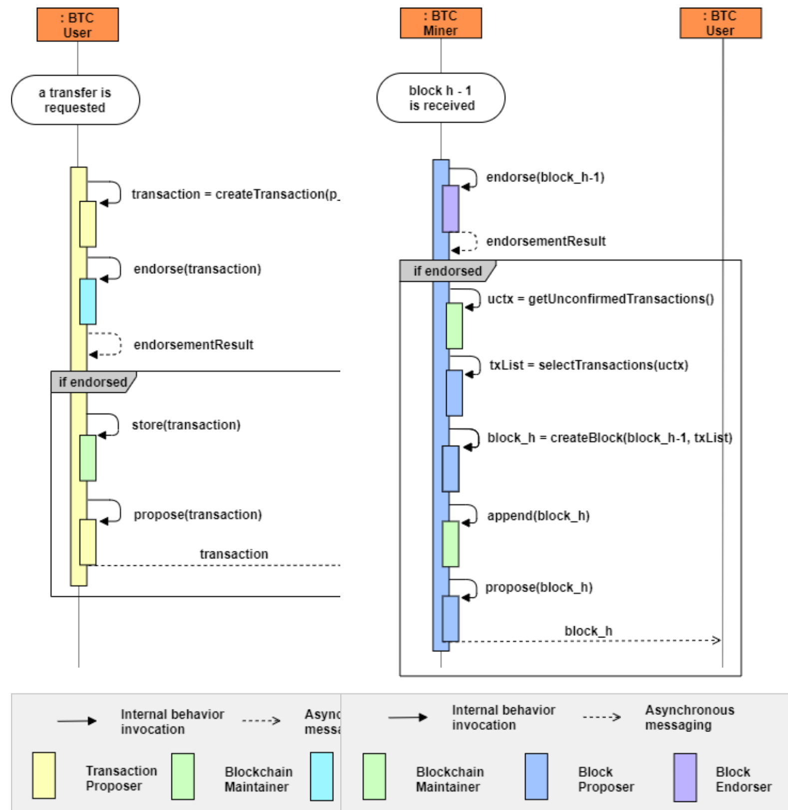


FIGURE 3.12: An organizational model of Bitcoin-like systems.

Using the AGR4BS model (Section 3.3), Bitcoin-like systems can be modeled as follows (Figure 3.12). We model users as *Node* agents (*i.e.*, BTC User) playing the roles *Transaction Proposer*, *Blockchain Maintainer* and *Transaction Endorser*, the latter one being a dummy transaction endorsement always returning *true* since the agent is signing its own transaction. Lightweight users (*i.e.*, BTC Light User), on the other hand, do not need to maintain a local blockchain and thus are modeled as *Node* agents playing the roles *Transaction Proposer* and *Transaction Endorser*. Miners are modeled as agents (*i.e.*, BTC Miner) playing the roles *Blockchain Maintainer*, *Block Proposer* and *Block Endorser* where *Block Endorser* uses the longest chain rule as a block endorsement policy. However, there is no explicitly defined Block Management group that miners belong to. Additionally, miners can also play both *Investor* and *Investee* roles to invest on themselves to increase their chance to succeed in creating and proposing blocks. Finally, we model mining pools as *Block Management* groups where miners collaboratively try to propose blocks and also as *Pool* groups where miners are *Investors* and leaders are *Investees*.

[12]P2POOL, http://p2pool.in/, last accessed on 05/03/2021.



(A) Sequence diagram of a Bitcoin transaction proposal (B) Sequence diagram of a Bitcoin block proposal

FIGURE 3.13

Figure 3.13a and Figure 3.13b illustrate concrete realizations of the *propose transaction* and the *propose block* behaviors respectively by showing the interactions between roles as sequence diagrams. The *propose transaction* behavior involves playing *Transaction Proposer*, *Transaction Endorser* and *Blockchain Maintainer* as defined in Section 3.3.2. In Bitcoin, these roles are played by the *BTC User* and *BTC Light User* agents, so that all interactions for proposing a transaction take place inside the proposing agent. The *propose block* behavior involves playing *Block Proposer*, *Blockchain Maintainer* and *Block Endorser* as defined in Section 3.3.2. In Bitcoin, all these roles are played by the *BTC Miner* agents, and consequently all interactions for proposing a block take place inside the proposing agent.

A global representation of our Bitcoin model using the cheeseboard representation is shown on Figure 3.14.

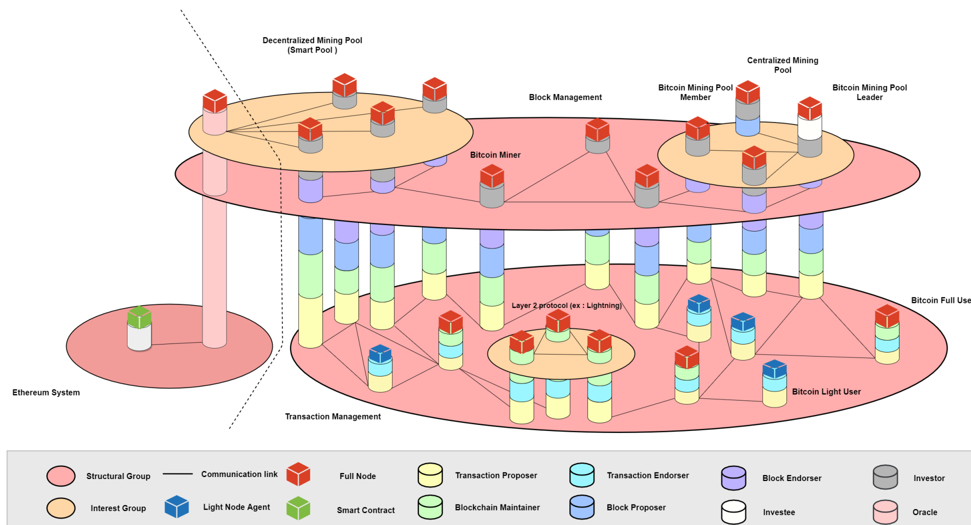


FIGURE 3.14: Cheeseboard diagram for an organizational view of a Bitcoin system

3.4.2 Ethereum 2.0

This subsection is a revised version of the original one published in (Roussille et al., 2022) to account for the current, stable design of the Ethereum 2.0 protocol.

Ethereum (Buterin, 2014; Wood, 2014), created in 2015, pioneered the use of smart contracts (Szabo, 1997) and decentralized applications powered by a turing complete smart contract programming language called Solidity. Ethereum is originally designed as a PoW based blockchain similar to Bitcoin. However, since this initial design suffers from severe scaling issues shared by most PoW blockchains, currently there is a transition to a new version (*i.e.*, Ethereum 2.0[13]) where Proof-of-Stake (PoS) is used. In this study, we focus on Ethereum 2.0, which is simply referred to as Ethereum hereafter[14].

System Overview

There are mainly four types of entities in an Ethereum blockchain system: users, validators, staking pools and delegators.

Ethereum users are very similar to those of Bitcoin (see Section 3.4.1). They create, sign and propose transactions, validate the diffused data, and maintain their local blockchains. The key difference is that Ethereum users can invoke smart contracts.

Validators are active participants of the consensus mechanism who have staked (i.e., *locked*) enough amount of coins [15] in a deposit smart contract. Periodically[16], a validators is chosen with an election frequency proportional to their staked coins (*i.e.*, Proof-of-Stake), and is allowed to produce a new block. To do so, it selects a set of unconfirmed transactions, gathers them in a block, and proposes that block to the committee formed by all validators through a two step BFT consensus protocol known as Gasper (Buterin et al., 2020).

[13]Eth2, `https://ethereum.org/en/eth2/`, last access on 04/06/2021.

[14]However, it should be noted that the actual Ethereum 2.0 is still partly undefined. Therefore, our view of the specifications might be subject to change.

[15]By the time of writing this thesis, it is 32 ETH.

[16]At roughly every 12 seconds.

The other members of the committee vote in favor or against the block proposal of the leader. If a consensus is reached, that block is to be appended to the local blockchains of all participants. The proposer of the accepted block is rewarded through the block reward as well as the transaction fees.

Participants who do not have enough coins to be validators can mutualize their stakes by *delegating* them in staking pools. If the delegated stakes cross the required threshold, the manager of the staking pool can act as a validator and thus can participate in the block creation process. Like in Bitcoin (see Section 3.4.1), there are basically two types of staking pools: centralized and decentralized.

In a centralized staking pool, the participants send their investments to a participant (*i.e.*, an exchange) that will stake it on the main blockchain (*i.e.*, beacon chain) to become a validator. In contrast, in a decentralized staking pool, the participants send their investments to the staking pool smart contract. This smart contract will then make those stakes available to node agents that are willing to contribute.

Organizational Model

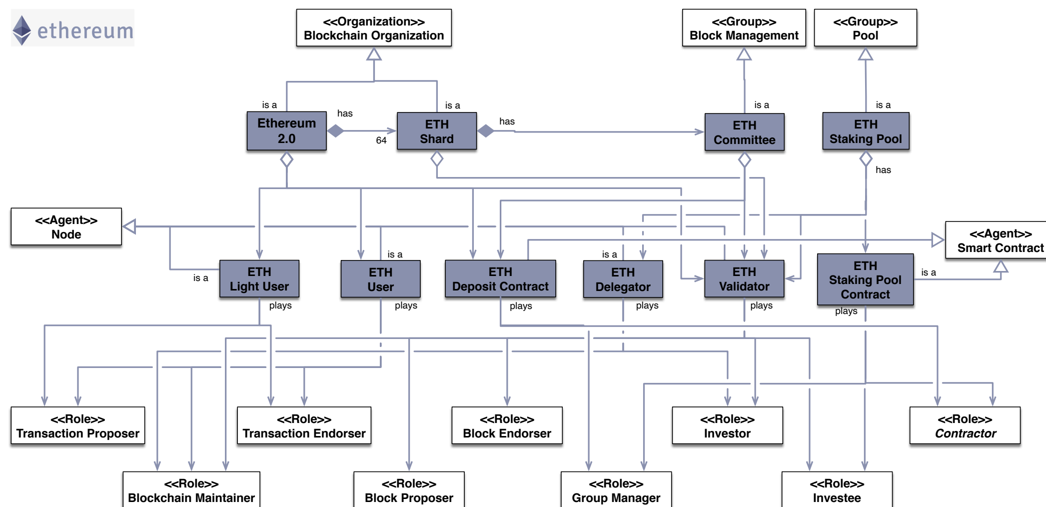


FIGURE 3.15: An organizational model of Ethereum 2.0 blockchain system.

Figure 3.15 presents the organizational structure of Ethereum. We model users similarly to Bitcoin users as node agents (*i.e.*, ETH User) playing the roles *Transaction Proposer*, *Blockchain Maintainer* and *Transaction Endorser*. Lightweight users (*i.e.*, ETH Light User), on the other hand, do not need to maintain a local blockchain and thus are modeled as *Node* agents playing the roles *Transaction Proposer* and *Transaction Endorser*. Validators are modeled as *Node* agents (*i.e.*, ETH Validator) playing the roles *Blockchain Maintainer*, *Block Proposer* and *Block Endorser* where *Block Endorser* uses the $\frac{2f}{3}$ rule[17] as a block endorsement policy. Additionally, validators can also play both *Investor* and *Investee* roles to increase their chance to enter into the committee. We model committees as *Block Management* groups (*i.e.*, ETH Committee). Besides, we model delegators as *Node* agents (*i.e.*, ETH Delegator) playing *Blockchain Maintainer* and *Investor*, and that belongs to a staking pool (*i.e.* ETH Staking Pool) modeled as a *Pool* group.

[17] A $\frac{2f}{3}$ majority of committee members must endorse (*i.e.*, vote for) the proposed block so that it is considered endorsed and may be broadcast to the rest of the network.

Figure 3.16 illustrates the concrete realization of *invest*, *redistribute* and *withdraw* behaviors by showing the interactions between roles as a sequence diagram. These behaviors involve playing *Investor*, *Investee* and *User defined* roles as defined in Section 3.3.2. In Ethereum, these roles are played by ETH Delegator and ETH Staking Pool Contract agents.

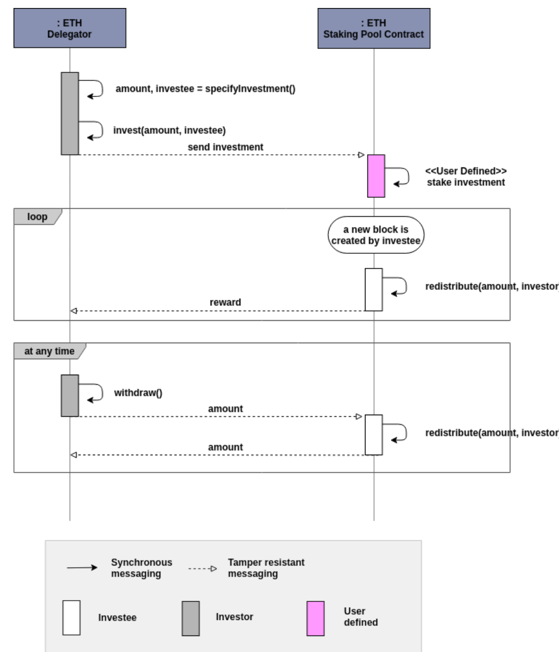


FIGURE 3.16: Sequence diagram of *invest*, *redistribute* and *withdraw* behaviors (Figure 3.6) for an Ethereum staking pool.

A global representation of our Ethereum model using the cheeseboard representation is shown on Figure 3.17.

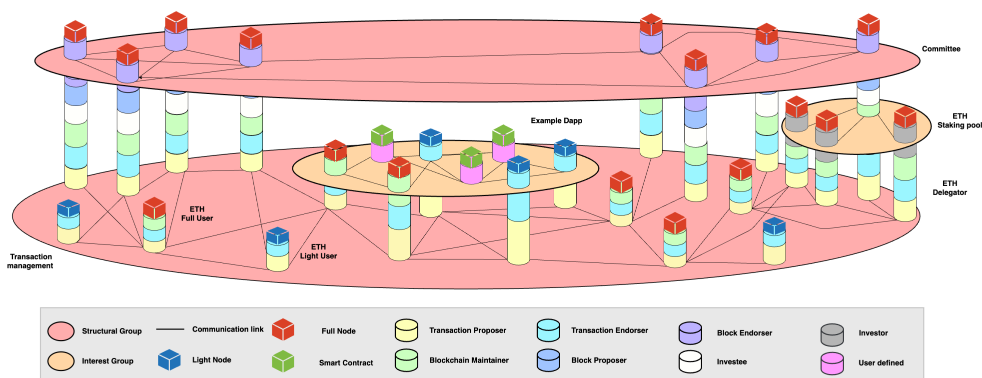


FIGURE 3.17: Cheeseboard diagram for an organizational view of an Ethereum 2.0 system.

3.4.3 Tendermint / Cosmos

Tendermint (Kwon, 2014) is an open Delegated Proof-of-Stake (DPoS) based blockchain protocol on which the Cosmos-like blockchain systems (Kwon and Buchman, 2016) are based on. Tendermint currently is a general purpose blockchain able to host any type of application through smart contracts written in a variety of supported

programming languages, the main example of such application is the Cosmos Network[18] connecting services from many different blockchains in order to create what is refered to as “the internet of blockchains”.

System Overview

In Tendermint, the consensus mechanism is based on a BFT algorithm where the committee leader is elected determistically through a round-robin fashion proportionally to its voting power. In fact, the whole Tendermint protocol is entirely deterministic and allows for extensive research. A particularity of Tendermint is the clear separation of the application layer and the core layer, allowing any programming language to be used to build decentralized applications.

There are mainly three types of entity in a Tendermint Blockchain System: Users, Delegators and Validators.

Validators are active participants, purposely staking currency to be part of the consensus mechanism of Tendermint (*i.e.,* BFT). This consensus algorithm evolves in epochs, rounds and phases.

At each epoch there are several rounds for block creation where a set of validators is selected as a committee to produce blocks.

At each round, a committee member is selected as the proposer to propose a block and then the round evolves in 3 phases: propose, prevote and precommit.

In the propose phase, the proposer selects a set of unconfirmed transactions, bundles them into a block and proposes it to the other committee members. In the prevote phase, the committee members check the proposal and decide whether to endorse such proposal or not.

In the last phase (*i.e.,* precommit), if a committee member receives sufficiently many proposals for the same block, then it can commit to it. Finally, if a committee member receives sufficiently many votes for the same block, it can decide for it (append it to its local chain) and be sure that: no other participant will decide for a different block; and that all the other committee members will decide for the same block as his. Once a committee member decides for a block, it also diffuses the decided block outside the committee.

Delegators are passive participants willing to be part of the consensus mechanism, but unable to do so reliably. Therefore, they delegate their stakes to existing Validators in exchange for a reward proportional to their contribution.

Users in Tendermint follow the same principle as for Ethereum (see Section 3.4.2). They are not involved in the consensus mechanism in any other way than proposing transactions. Users create transaction, sign and propose them to the network.

When a new unconfirmed transaction is received by a User, it first needs to validate it. If it is valid, the User will proceed in storing the valid unconfirmed transaction in its memory pool before broadcasting it to its peers.

When a new block is received, Users also validate it with respect to their current local blockchain. If the new block is considered valid, they will append it to their local replica before broadcasting the new block to their peers. Similarly to Ethereum, Tendermint Users can invoke smart contracts.

Organizational Model

Figure 3.18 presents the organizational structure of Tendermint. We model users similarly to Ethereum users as node agents (*i.e.,* TDM User) part of the Transaction

[18] https://cosmos.network/ last accessed on 13/09/2021

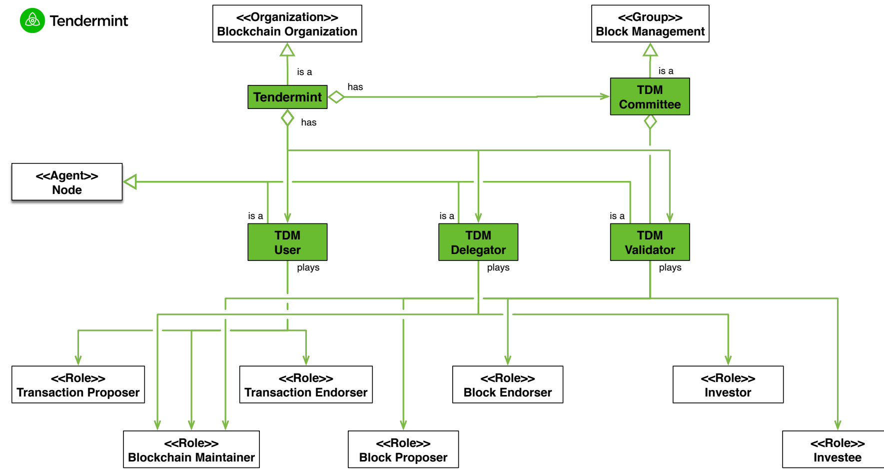


FIGURE 3.18: The organizational model of Tendermint-like systems.

Management group and playing the roles *Transaction Proposer*, *Blockchain Maintainer* and *Transaction Endorser*. Delegators (*i.e.*, TDM Validator) play both *Blockchain Maintainer* and *Blockchain Investor*. Finally, Validators (*i.e.*, TDM Validator) play the roles *Blockchain Maintainer*, *Block Proposer* and *Investee*. Committees are also modeled as Block Management groups (*i.e.*, TDM Committee).

FIGURE 3.19: Sequence diagram of the *propose block* behavior (Figure 3.5) in Tendermint.

Figure 3.19 illustrates the block proposition process in Tendermint. When a validator playing *Block Proposer* is elected as committee leader, it may construct a block proposal by fetching and selecting the pending unconfirmed transactions through the *selectTransactions()* behavior. The selected transactions are then bundled into a block by using the *createBlock(Block, Transaction[])* behavior.

The BFT Consensus can now take place between the leader and other committee members. First, the leader sends its proposal to the committee. Second, each

committee member as *Block Endorser* votes or not for the proposal and forwards its decision to every other member using *endorse(Block)*. Finally, the commit step takes place where each committee member as a *Blockchain Maintainer* applies the consensus result to its local blockchain using the *append(Block)* behavior and then diffuses it.

A global representation of our Tendermint model using the cheeseboard representation is shown on Figure 3.20.

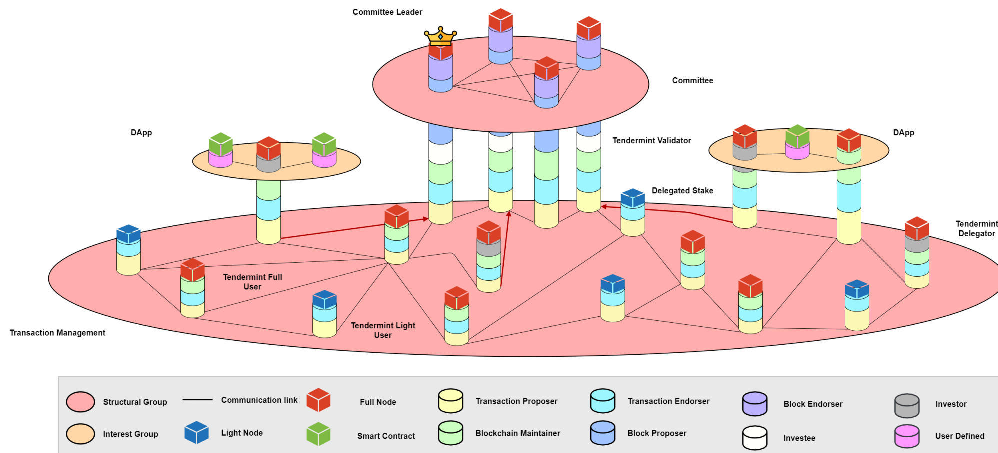


FIGURE 3.20: Cheeseboard diagram for an organizational view of a Tendermint system

3.4.4 Hyperledger Fabric

Hyperledger Fabric (Androulaki et al., 2018) is a permissioned blockchain system intended to hold applications for industrial ecosystems wherein participants trust and know each other. Given the trusted nature of the participants' interactions, Hyperledger Fabric uses a PBFT consensus realized by a trusted set of entities, therefore sacrificing decentralization for throughput.

System Overview

A Hyperledger Fabric system is basically composed of channels, organizations and ordering services.

Channels are the main communication mechanisms by which participants can communicate. Each channel is dedicated to maintaining a single independent ledger (*i.e.*, blockchain). Every transaction and block are diffused through the channel, given that the issuer is identified and authorized by one of the organizations operating on that channel.

A Hyperledger Fabric organization is an abstract entity, usually representing a real-world organization such as a company. An organization is composed of a Membership Service Provider (MSP), applications, peers and orderers.

A MSP defines the rights of members to act on a given channel and perform specific actions through a set of cryptographic signatures and certificates, therefore possibly aliasing the notion of identity with roles.

Peers store and maintain copies of blockchain(s) and chaincodes (*i.e.*, smart contracts). Peers also endorse newly created transactions according to an endorsement policy to allow them to move toward an inclusion in the blockchain. To do so, peers

simulate the execution of proposed transactions, sign them and return them back to the applications.

Applications[19] are entities that interact with peers to access a blockchain and smart contracts (*i.e.,* chaincode). Applications and smart contracts together form a decentralized application. When an application wants to interact with a smart contract, it creates a dedicated transaction and then asks the peers to endorse this transaction. Upon endorsement, the applications diffuses its transaction to the ordering service (*i.e.,* orderers of the corresponding channel).

Orderers are responsible for ordering the endorsed unconfirmed transactions and putting them into a block. All orderers operating on the same channel, regardless of being member of different organizations, form the ordering service for that channel. The orderers (*i.e.,* ordering service) rely on deterministic consensus algorithms where the proposed block is guaranteed to be final and correct (*e.g.,* BFT or single trusted authority). However, they do not use a predefined consensus algorithm and thus the block creation process can be different from one system to another.

Organizational Model

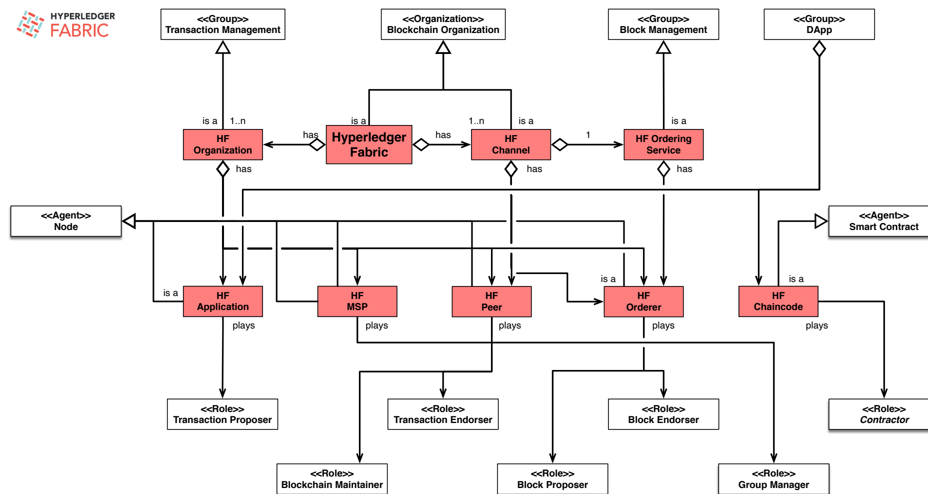


FIGURE 3.21: An organizational model of Hyperledger-like systems.

Figure 3.21 presents the organizational structure of Hyperledger Fabric. We model Hyperledger Fabric (HF) Organizations as *Transaction Management* groups and HF Ordering Services as *Block Management* ones. Membership Service Providers (MSP) are modeled as *Node* agents playing the *Group Manager* role. Applications are *Node* agents playing only the *Transaction Proposer* role. Peers, on the other hand, are *Node* agents playing both *Blockchain Maintainer* and *Transaction Endorser*. Orderers are *Node* agents playing *Block Proposer* and *Block Endorser* roles. Chaincodes are *Smart Contract* agents that can play *Contractor* role. Even though, it is not modeled explicitly in Hyperledger Fabric, applications and chaincodes together belong to user defined *DApp* groups.

[19] In Hyperledger Fabric documentation, *application*, *client* and *user* are used interchangeably but since in the architectural model the concept is called application we use it as they defined.

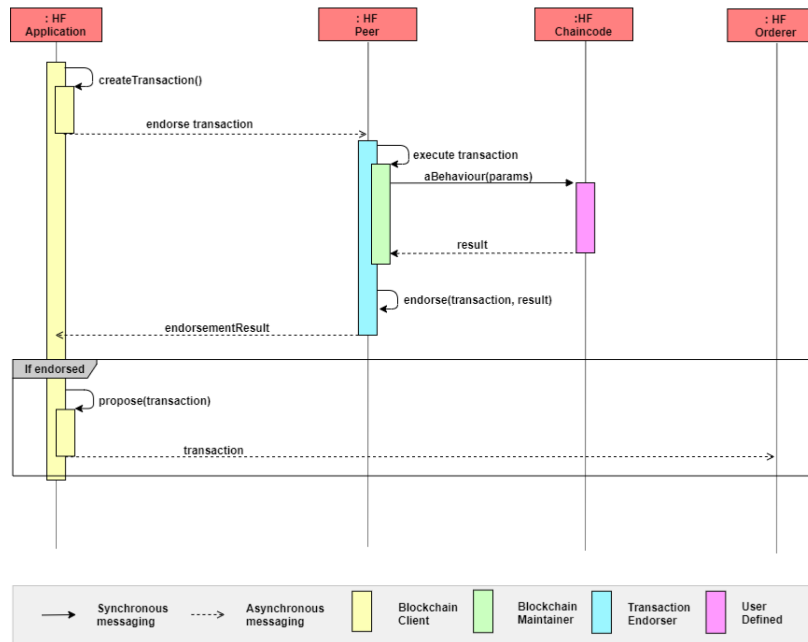


FIGURE 3.22: Sequence diagram of the *propose transaction* behavior (Figure 3.4) in Hyperledger Fabric.

Figure 3.22 illustrates the concrete realization of *propose transaction* by showing the interactions between roles as a sequence diagram. The *propose transaction* behavior involves playing *Transaction Proposer*, *Transaction Endorser*, *Blockchain Maintainer* and *Contractor* as defined in Section 3.3.2. In Hyperledger Fabric, these roles are played by *HF Application*, *HF Peer*, *HF Orderer* and *HF Chaincode* agents respectively, and consequently interactions for proposing a transaction take place between several agents.

A global representation of our Hyperledger Fabric model using the cheeseboard representation is shown on Figure 3.23.

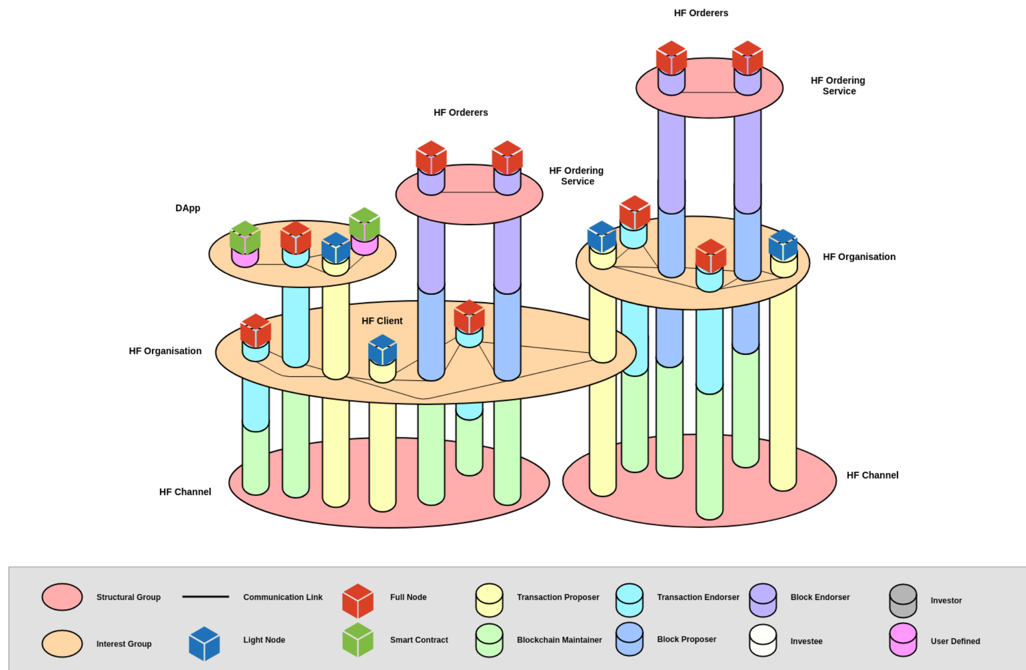


FIGURE 3.23: Cheeseboard diagram for an organizational view of a Hyperledger Fabric system

3.5 Modeling Attacks

In the previous section, we have shown the effectiveness of the AGR4BS model for modeling various blockchain systems. In this section, we shed light on some possible attacks on blockchain systems to illustrate how our model is able to model them at both the agent and organizational levels.

The growing interest for blockchain and cryptocurrencies incentivizes the exploration and exploitation of attacks that may target different parts of the system. For instance, some attacks target the network or the system itself (Apostolaki et al., 2017; Mirkin et al., 2020) either through partitioning, DDOS attacks or wormhole attacks.

Other attacks, such as the 51% attack, aim at acquiring an overwhelming share of the production capacities in the system (*e.g.*, computational power in a PoW blockchain or stakes in a PoS blockchain). With such power, an attacker or group of attackers can rewrite the blockchain as they wish, perform double spending attacks or censor any entity of their choice. Several elaborated ways to achieve such goal are explored in (Eyal and Sirer, 2014; Sapirshtein et al., 2016), where the attacker builds an adversarial chain and makes use of the bitcoin fork rule to waste a significant part of the honest participant's computational power.

Some attacks, such as (Nayak et al., 2016), combine both attack vectors into one, building an adversarial chain with the help of unknowing participants that were previously isolated from the network through an eclipse attack.

Different attack vectors also exist through smart contracts / DApp bug exploitation. The most well-known example is the DAO attack[20] which led to the thief of 3.6 millions Ethereum tokens.

Most DApp are susceptible to front-running attack (Eskandari et al., 2020a) where the attacker makes an unfair use of information related to events that were not yet

[20]https://blog.b9lab.com/the-dao-hack-in-eight-minutes-94919018692d last accessed on 06/07/2021

written on the blockchain (*i.e.*, pending transactions in a blockchain are essentially insight on future events). This list of attack vector is by no means exhaustive and only aims at covering the main events and concerns about current blockchain systems, highlighting the fact that blockchain systems are vulnerable.

The next figures, associated with some examples of attack, are purposely simplified and do not show every role to emphasize the organizational impact of such an attack.

In the following subsections, we will show briefly how AGR4BS can also be used for modeling three of the aforementioned attacks on blockchain systems.

3.5.1 Front Running

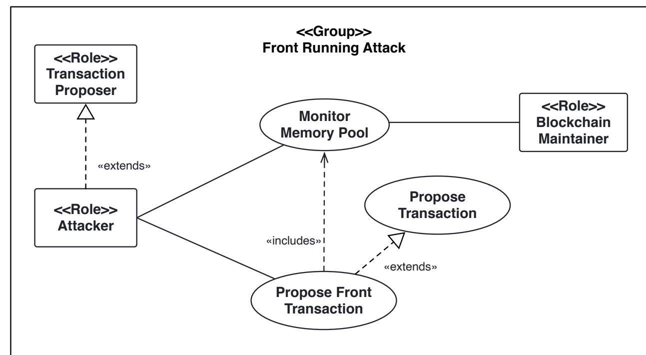


FIGURE 3.24: The organizational structure of a transaction based front-running attack.

In this attack, the attacker continuously monitors the pending transactions and takes advantage of the transaction selection / block creation process to front run another transaction in a profitable manner. For instance, on a decentralized exchange the attacker might see that a user intents to buy a massive amount of a given asset, and therefore expect for the price to increase significantly.

In such a situation, the attacker might try to front run that transaction to buy some of the asset before the honest user and profit from the attack. The attacker can do so either by issuing a transaction with a large fee, thus ensuring that block creators will select it or by being a block creator itself.

This attack does not make use of any bug nor deviates from the protocol in any way. It is only possible due to the fact that the blockchain makes future events and intents known to every participant beforehand.

We model this attack with an attacker role deviating from *Transaction Proposer* as showed in Figure 3.24. The attacker rely on the *Blockchain Manager* role to get current information about pending transactions through the *Monitor Memory Pool* behavior. When a profitable front-running scenario is found in the memory pool, the attacker creates a front-running transaction and proposes it to the network using the deviant *Propose Front Transaction* behavior.

3.5.2 Eclipse Attack

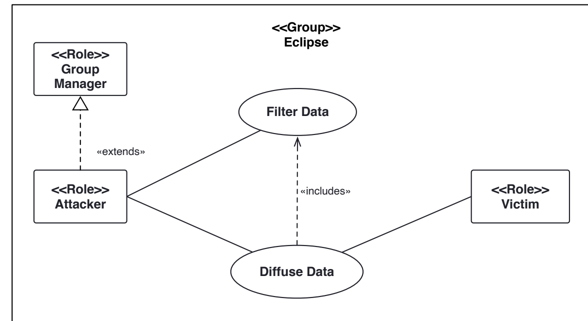


FIGURE 3.25: The organizational structure of an eclipse attack.

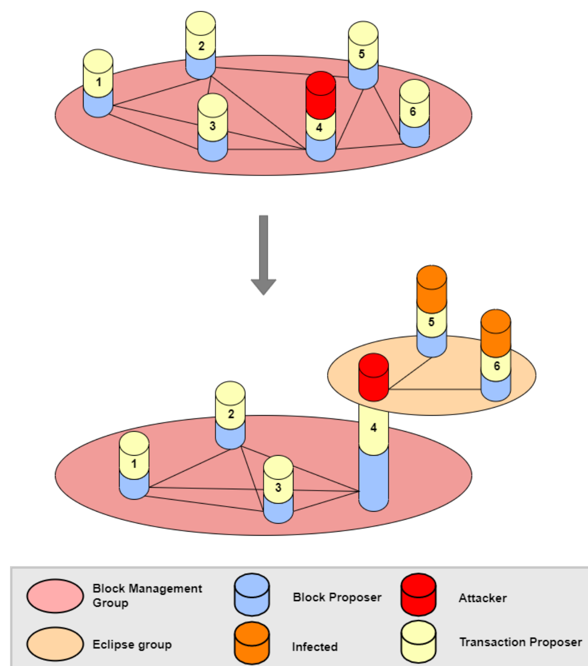


FIGURE 3.26: The cheeseboard transition diagram of an eclipse attack

In this attack (Apostolaki et al., 2017), the attacker interposes itself between its victim(s) and the rest of the system by partitioning the network. The attacker then controls what the victim(s) are sending and receiving.

An attacker might infect a network router or an agent directly to control both the incoming and outgoing traffic. Victim agents are not aware that they are infected and may receive different blocks and transactions than the rest of the system depending on what the attacker wishes to communicate.

The impact of such an attack could range from increased propagation delay to the creation of adversarial chains, possibly leading to double spending attempts. For example, in the case of the creation of an adversarial chain in a PoW blockchain, the victims might be mining on an adversarial chain built by the attacker without any way of knowing it, therefore contributing to the malicious intent of the attacker while behaving correctly according to the protocol as described in (Nayak et al., 2016).

The eclipse attack can be modeled using our generic model as shown in Figure 3.25. We also show its organizational impact on the system on Figure 3.26. The victim's view of the blockchain system is controlled by a deviation of the Blockchain Maintainer role from the attacker, which filters and diffuses only the data that it wishes its victims to see. Similarly, any data received from victim agents will be filtered before being held or relayed to the whole blockchain system.

We model agents which are isolated as agents having the Victim role, meaning that even though they believe to be connected to the whole blockchain system, they are only connected to the attacker or possibly other victims. The Victim role does not involve any behavioral deviation.

3.5.3 Wormhole Attack

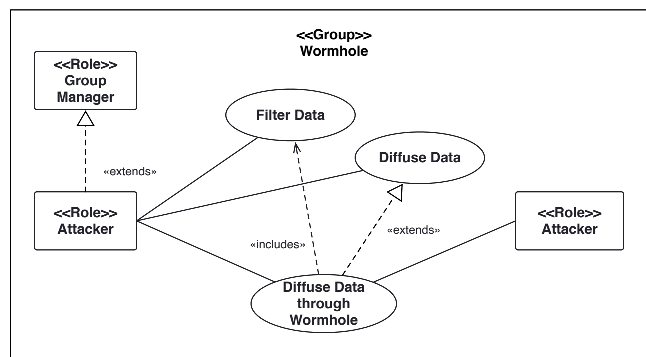


FIGURE 3.27: The organizational structure of a wormhole attack

In the wormhole attack, two or more attackers located at different points in the network establish an overlay network to transfer information faster than on the main P2P network. They do so through alteration / addition to the *Propose Transaction* and *Propose Block* roles and their related behaviors.

So, this attack aims at getting an unfair advantage over other participants, relying solely on the longer propagation time of the P2P network. Furthermore, the attackers may also choose to relay only relevant information according to their criteria, such as highly valuable transactions or new block proposal.

As shown in Figures 3.27 and 3.28 we model this attack through an Attacker role overloading the generic diffuse behavior. Every data received will be handled by the *Filter Data* behavior to assess of its relevance for the attackers before being transmitted through the wormhole using *Diffuse Data through Wormhole*.

Figure 3.28 shows how the attackers can create such wormhole to overcome a bad network topology. At any point in time, an attacker might choose to include a new agent to the group. The attackers can do so since they all extend the *Group Manager* role and can therefore expand their attack if it is deemed necessary and profitable.

3.6 Discussion

In this section, we first discuss the expressivity of AGR4BS followed by key differences between the blockchain systems that we modeled in Section 3.4 (Section 3.6.1). From those differences we discuss the robustness and the vulnerabilities of blockchain systems (Section 3.6.2). We then discuss the robustness and resilience of

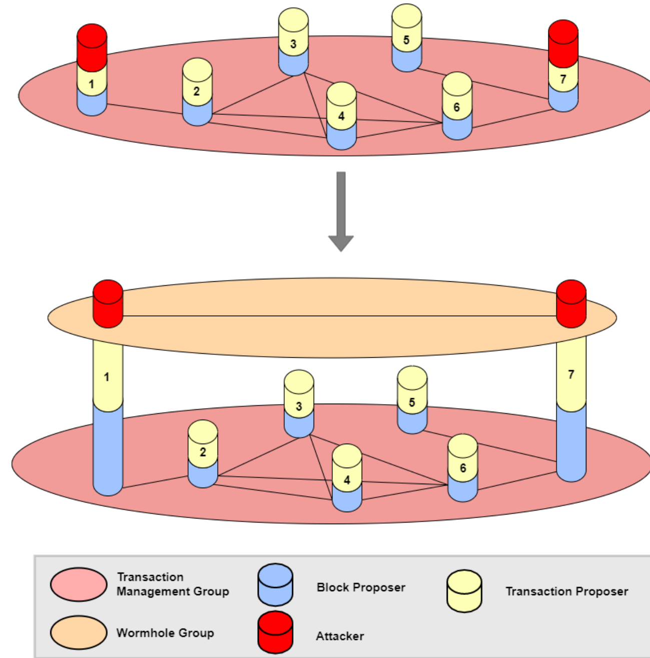


FIGURE 3.28: The cheeseboard transition diagram of a wormhole attack.

blockchain systems (Section 3.6.3). Finally, we conclude this section with a discussion about reliability (Section 3.6.4).

3.6.1 Expressivity of AGR4BS

As stated in Section 3.1, several frameworks are able to partially represent blockchain systems. AGR4BS is compatible with the core characteristics of blockchain systems.

The distributed characteristic is expressed through many agents present in the system which are required to execute a specific algorithm for both the blockchain and their individual incentives. The social characteristic of blockchain systems are easily represented through both inter- and intra-organization interactions. Each contributing agent is also economically incentivized through the blockchain reward system, which encompasses the economical nature of that environment. Finally, organizational modeling provides *modularity*, thus allowing rapid development and adaptation of AGR4BS to new blockchain designs. According to this, the case studies given in Section 3.4 show that AGR4BS provides useful abstractions for defining different types of existing blockchain systems.

Consequently, AGR4BS can also be used to construct dedicated modular software models (*e.g.*, prototypes, simulation models and so on) allowing us to benchmark and compare different types of blockchains based on common features through a generic approach, with high component re-usability.

3.6.2 Organizational Differences of Blockchain Systems

In this subsection, we discuss the organizational differences of blockchain systems based on the used abstractions, *i.e.*, group, agent and role.

Groups \ Systems		BTC	ETH	TDM	HF
Structural	Transaction Management	Implicit	Implicit	Implicit	Explicit
	Block Management	Implicit	Explicit	Explicit	Explicit
Interest	Overlay	Explicit	Explicit	Explicit	N/A
	Pool	Explicit	Explicit	N/A	N/A
	DApp	N/A	Explicit	Explicit	Explicit

TABLE 3.1: Group differences of blockchain systems

Group Differences

In Bitcoin (Section 3.4.1), both structural groups (*i.e.,* Transaction Management and Block Management) are *implicit groups*. The interest groups Overlay Network and Pool, on the other hand, are *explicit groups*.

In Ethereum (Section 3.4.2), while the transaction management group is *implicit*, the block management is *explicit*. The interest groups Pool and Decentralized Application are explicit.

In Tendermint (Section 3.4.3), the situation is similar to Ethereum.

In Hyperledger Fabric (Section 3.4.4), on the other hand, all groups are explicitly defined.

Entering into an *explicit* group is a complex process, since an agent has to be *authorized* to enter (see Section 3.3.2). Our study shows that in different explicit blockchain groups, there are different mechanisms for realizing such authorizations.

In Hyperledger Fabric, the *HF Organization* groups have an *HF MSP* agent playing *Group Manager* which is responsible for authorizing agents.

In Ethereum and Tendermint, on the other hand, the *ETH Committee* group's authorization logic is immutably defined on the blockchain and therefore has no dedicated entry point. Furthermore, in Ethereum 2.0, structural functionalities, and therefore structural groups, are managed by smart contract agents. This is a core difference with other blockchains, as the regulation mechanisms are inherent to the system, not one enforced by node agents or classical incentives.

As can be seen in Table 3.1, all interest groups are *explicit*. However, by merely looking at this, *it cannot be concluded that all interest groups are explicit*. In fact, it is quite possible for an interest group to be implicit (*e.g.,* an attack group).

Finally, note that different agents might have different (and possibly wrong) views of the same group. For instance, in the case of an Eclipse Attack (Section 3.5.2), even though the attacker is creating an *Interest Group*, the victims still see it as the genuine *Block Management Group*.

Agent Differences

Case studies also show that agents and roles are orchestrated in a different way in different blockchain systems (see Figures 3.14, 3.17, 3.20, 3.23). Consider, for instance, the endorse transaction behavior involving the Transaction Proposer and Transaction Endorser roles (Figure 3.4). While this behavior is implicitly realized by the *same* agent in Bitcoin (Figure 3.13a), it is explicitly realized by several agents in Hyperledger Fabric (Figure 3.22).

As another instance, one can focus on the endorse block behavior involving the Block Proposer and the Block Endorser roles (Figure 3.4). While this behavior is

implicitly realized by the *same* agent in Bitcoin (Figure 3.13b), it is explicitly realized by several agents in Tendermint (Figure 3.19).

Role Differences

Overall, not all generic role types are used and/or realized in every blockchain. While Bitcoin (Figure 3.14), Ethereum (Figure 3.17) and Tendermint (Figure 3.20) use all of them, the Investor and Investee roles are clearly irrelevant in Hyperledger Fabric (Figure 3.23). In contrast, Hyperledger Fabric requires a specific implementation of endorsement mechanisms for transactions, while in both Bitcoin and Ethereum, it serves no real purpose as endorsements are always realized implicitly.

3.6.3 Robustness and Resilience

Overall, looking at the blockchain systems discussed, it can be said that the very essential group for a blockchain system is *transaction management*, since nearly all agents play a role in it. Consequently, vulnerabilities in a transaction management group can affect the whole blockchain system. However, except for Hyperledger Fabric, this group is always implicitly defined.

As shown concretely by the case studies, AGR4BS allows us to see clearly the similarities and differences between agents as well as blockchain systems. By identifying the deviations of high-level behaviors of agents, it is thus possible to identify cross-cutting potential high-level vulnerabilities of blockchain systems. The solutions found for these high-level vulnerabilities might later be applied for all types of blockchains.

Regarding the resilience of existing groups, a clear emphasis is put on the block management group in the existing literature, this for several obvious reasons. The block management group allows for significant rewards if taken over by malicious agents. This group is also less reliant on other groups or the ledger than the transaction management one, and therefore a more manageable target.

However, there exist other less generic groups that can allow for even greater reward, such as DApp and especially DeFI implementations. But compromising those rely more on bug exploitation than on actual corruption, since they are by definition replicated over the whole blockchain.

3.6.4 Reliability

In Section 3.3.1, we did not emphasize one core conceptual distinction regarding the combination of roles and agents. When any given role is played by a Node agent, that one has the possibility to deviate from its nominal behavior, which echoes to the trustless paradigm of decentralized systems and more specifically to blockchain systems.

Still, smart contracts are stored both immutably and transparently in the blockchain data structure, and are deterministic by design. Therefore, when a functionality is required in a blockchain system, it should always be implemented through smart contracts if possible. The reason is that smart contracts bring trust in a trustless system through their transparency and immutability, as well as being replicated and therefore decentralized.

So, while any functionality could technically be implemented in Node agents, they do have the potential to purposely deviate from the nominal behavior. They are

also subject to failures and are not replicated by design. Node agents are therefore unreliable by opposition to smart contracts.

3.7 Conclusion

As of today, the blockchain technology is used as a basis for a wide range of applications ranging from mere cryptocurrencies to decentralized applications. However, we face highly competitive and complex cases that have technical problems (*e.g.*, data reliability, confidentiality, archiving) which are being constantly reshaped by user (*e.g.*, performance (# of transactions/minute), fees), technology (*e.g.*, consensus protocol, parameters, cost) and regulatory (*e.g.*, standards, laws, GDPR) requirements. Moreover, blockchain applications are intended to be deployed into various environments such as computers, smartphones, vehicles, drones, IoT devices and so on. Furthermore, the blockchain ecosystem is very active, dynamic and rich (*e.g.*, Bitcoin, Ethereum, Tendermint, Hyperledger, Sycomore). This defines a diverse and growing ecosystem wherein each blockchain solution relying on common principles while having their own characteristics.

In this context, several approaches for representing blockchain systems have been proposed, according to different modeling paradigms, to investigate different aspects of blockchain systems. In these studies, the designed models are not intended to be generic, since they focus on particular issues. Especially, the modeling is often done considering only one particular variation of high-level details, such as the used consensus protocol, or only one particular kind of blockchain (*e.g.*, Bitcoin).

Consequently, there is not a unified way of representing blockchain systems from which the blockchain community could benefit and capitalize on. So, we argued that there is a need for a realistic and highly flexible model able to represent a wide range of existing and future blockchain systems that may have widely different architectures and objectives.

To this end, after having introduced the necessary blockchain concepts (Chapter 2) and existing ways of modeling such systems, we have motivated our choice for an organizational modeling (Section 3.2) and proposed a generic organizational model for blockchain systems, namely AGR4BS (Section 3.3), whose main purpose is to provide a unification of existing blockchain implementations through a single model.

As far as we know, AGR4BS is the first organizational multi-agent blockchain model for blockchain systems. More notably, AGR4BS provides the necessary basic abstractions (allowing consensus on fundamental terms) to dissect existing blockchain systems and serves as a blueprint for exploring new alternative ones.

Especially, we have shown how we used AGR4BS to model different blockchain systems such as Bitcoin (Section 3.4.1), Ethereum (Section 3.4.2), Tendermint (Section 3.4.3) and Hyperledger Fabric (Section 3.4.4), thus demonstrating the genericity and adaptability of AGR4BS. Moreover, both the Bitcoin and Tendermint prototype implementations are available at : `https://gite.lirmm.fr/fmichel/agr4bs`.

Furthermore, in Section 3.5, we highlighted a few vulnerabilities of blockchain systems and their organizational consequences. Being able to represent divergent behavior at both the system and the agent level is mandatory to provide a complete view of any kind of blockchain system.

Lastly, in Section 3.6.1 we analyzed and discussed on the expressivity of AGR4BS.

This model serves as groundwork for the remaining of the thesis as it will be the common denominator of both the role based taxonomy defined in Chapter 4 and our blockchain simulator covered in Chapter 5.

Chapter 4

A role based taxonomy of incentive vulnerabilities

This chapter provides an in-depth description of our proposed taxonomy for blockchain incentive vulnerabilities. It is grounded in the AGR4BS model described in Chapter 3 and provides a coherent method to describe and rank vulnerabilities to guide research efforts and computational power on the more serious vulnerabilities. This chapter is an extension of a work realized during this Ph.D, aimed at categorizing blockchain attack vectors in line with the AGR4BS model (Roussille et al., 2023).

4.1 Related Work

A vulnerability can be formally defined as a weakness or flaw within a system that can be exploited by an external party to cause harm to the system. In this study, we focus on a specific type of blockchain system vulnerability: *incentive vulnerability*, which we define as a misalignment between (1) the behavior of an agent as expected by the protocol designers and (2) the behavior eventually obtained by following a utility based interpretation of the incentives.

This misalignment incentivizes participants to deviate from their nominal behavior (*i.e.,* external fault). In that sense, a behavior is said to deviate when not strictly adhering to the official implementation (*i.e.,* the nominal behavior). If such a deviation harms the system or its participants, one or several countermeasures must be designed and implemented to mitigate the deviation feasibility and/or its impact. Strictly speaking, an incentive vulnerability is the root cause of a deviation.

There are exhaustive reviews and surveys reported in the literature: (Alkhalifah et al., 2020; Hameed et al., 2022; Li et al., 2020; Saad et al., 2020; Sayeed et al., 2020) (see Table 4.1 for a comparison).

(Saad et al., 2020) define an attack taxonomy over the following three main categories: Structure attacks, Peer-to-Peer attacks and Application attacks. They list the known attacks, and discuss the existing or potential countermeasures.

(Hameed et al., 2022) define several taxonomies with a strong focus on industrial application of blockchain systems. Those taxonomies relate to design, security, privacy requirements, and security. They expose several attacks on a per-layer basis, with known or proposed countermeasures.

(Sayeed et al., 2020) propose a study focused on the Ethereum smart contracts / application layer. They implicitly provide a taxonomy through a categorization of the main types of attacks and discuss existing tools and techniques enabling some level of protection.

(Alkhalifah et al., 2020) define a taxonomy of blockchain threats and vulnerabilities over the following categories : Client's Vulnerabilities, Consensus Mechanisms Vulnerabilities, Mining Pool Vulnerabilities, Network Vulnerabilities and Smart Contract Vulnerabilities (Ethereum and EVM focuses).

(Li et al., 2020) survey the security of blockchain systems and propose a succinct taxonomy of blockchain risks covering encryption, consensus and transactions. They also propose a taxonomy of Ethereum's smart contracts vulnerabilities.

Exhaustive studies focus on the "How" and "Where" of attacks, defining how it impacts the system as well as in which layer it takes place. In this perspective, countermeasures are often restrained to only treat the problem's consequences (*i.e.,* detection systems, increased resilience).

Specific attack studies almost only focus on the "Why", explaining the reasons and incentives motivating the attack. The proposed countermeasures modify the system so that the attack is no longer incentivized, treating the root cause of the problem.

Our taxonomy aims to merge both approaches, covering most deviations with a focus on incentives, and a role-based classification for a natural use with reinforcement learning as shown in Table 4.1.

Caracteristics \ References	(Saad et al., 2020)	(Hameed et al., 2022)	(Sayeed et al., 2020)	(Alkhalifah et al., 2020)	(Li et al., 2020)	**Ours**
Layer of interest	- Application - Blockchain - Network	- Application - Blockchain - Network	- Application	- Application - Blockchain - Network	- Application - Blockchain - Network	- Blockchain
Proposes Countermeasures	Yes	Yes	Yes	Yes	Yes	Yes
Classification	Layer Based	Layer Based	Attack Type	Layer Based	Risk & Vulnerability	Role-Based
Incentives Focused	No	No	No	No	No	Yes

TABLE 4.1: Comparison of studies with a focus on taxonomies concerning the security of blockchains.

4.2 Taxonomy Characteristics

To classify, categorize and measure vulnerabilities, we use the following concepts: impact family, severity, risk, scale, priority score and system.

Impact Family relates to the expected impact of vulnerability exploitation. Three possibilities are considered: Fairness, Economics and Security. A Fairness impact arises whenever discrimination between agents occurs for any reason that is not part of the protocol. Also, any imbalance between the proportionality of invested resources and the reward is included in this family. An Economic impact happens when the system's economy is disturbed, such as an artificial transaction fee increase. A security impact occurs when a core property of the blockchain is compromised such as not being able to finalize newly created blocks, or a loosing integrity of the blockchain by including invalid data.

Severity defines the level of impact of a successful attack, and takes a value in Very High, High, Medium, Low, and Very Low, which are aliases for 1, $\frac{4}{5}$, $\frac{3}{5}$ $\frac{2}{5}$ and $\frac{1}{5}$ respectively. These levels are not based on a quantifiable notion of severity but are used to categorize vulnerabilities informally and thus help compute their respective priority scores.

'Very Low' implies that an agent or group of agents is mildly impacted but still functioning, with no quantifiable impact on groups or the system. 'Low' also implies that an agent, or a subgroup of agents, is impacted in a more meaningful way, possibly non-functional, while the group and blockchain system they are part of is still functional. A 'Medium' severity level impacts both agents and groups in a way

not jeopardizing the system, but implying consequences in at least one of its core properties, such as Fairness, Security or Economics. A 'High' severity level implies a non-negligible impact on the system. Finally, 'Very High' refers to an immediate threat, such as a general unfairness issue or halting of the system.

Risk refers to the feasibility of an attack in terms of resources required to conduct it. The risk levels are similar to the ones defined for severity: 'Very High', 'High', 'Medium', 'Low', and 'Very Low'. Those risk levels are also mapped to values similar to severity levels. 'Very High' signifies that the associated vulnerability is relatively easy to set up as it only requires a few resources. 'High' refers to an attack where some amount of resources must be committed to it, but still doable by most participants. 'Medium' means that an attack requires a non-trivial amount of resources. 'Low' and 'Very Low' are used to describe attacks requiring overwhelmingly large resources.

An important note regarding the definition of risk, and more specifically feasibility: the resource required to achieve a specific attack depends on the attack's type. For example, a mining-based attack requires computational power, while a network-related one requires many identities and bandwidth. Our resource definition is, therefore, fluid to accommodate various attack types.

Scale. As blockchains are decentralized, one must differentiate the risk and severity levels over the scale of the actual attack. In the scope of our work, we consider both a low-scale attack and a large-scale one. Depending on the attack type, the scale might be related to the number of attackers (*i.e.,* sybil attack), the total required computing power (*i.e.,* mining attack) or the economic value (*i.e.,* staking attack) required for the attack.

Priority Score ranks the identified vulnerabilities loosely. It is based on severity and risk and defined as the product of those variables. As we can compute low-scale and large-scale priority scores, we opt for a pessimistic approach and consider the attack to have an overall priority score equal to the maximum priority score across the different scales.

System describes the subset of blockchain systems vulnerable to a specific attack. Some systems may be independent, while others might be linked deeply to the underlying consensus mechanism: PoW, PoS, PoA, PBFT, Explicit Block and/or Transaction Endorsement, All.

In the following, we present each role and its deviations. For each role, we summarize all of its known deviations, their impact families, severities and risks in low and large scales, and their calculated priority scores (summarized in Table 4.2). Each subsection starts with a nominal behavior definition of a role, followed by possible deviations.

4.3 Role-based Taxonomy

Here we present the role-based taxonomy of vulnerabilities (see Table 4.2) that provides a classification of violable constraints and assumptions that are bound to the roles.

4.3.1 Block Proposer

Nominal behavior. Block Proposer selects a subset of the most relevant transactions, orders them, and tries to create a valid block, always extending the main chain according to the consensus protocol of the system and, if it succeeds, immediately

Role	Deviations Exploiting Incentive Vulnerabilities				Vulnerability Metrics						
	Deviation Name	Deviated Behavior	Impacted Roles	Reference	Impact Family	Low Scale		Large Scale		Priority	System
						Severity	Risk	Severity	Risk	Score	
Block Proposer	Censure of Transaction	selectTransactions	Transaction Proposer	(Gürcan et al., 2017)	Fairness	●○○○○	●●●●○	●●●○○	●●●○○	0.16	All
	Selective Block Propagation	proposeBlock	Blockchain Maintainer Block Proposer	N/A		●○○○○	●○○○○	●●●○○	●●○○○	0.24	All
	Consensus delay	createBlock proposeBlock	All	N/A		●○○○○	●○○○○	●●●●●	●●●●○	0.80	PBFT
	Selfish / Stubborn Block Creation	createBlock proposeBlock	Blockchain Maintainer Block Proposer	(Eyal and Sirer, 2014)	Fairness Security	●○○○○	●●●●●	●●●●●	●●●●○	0.80	PoW & PoS*
	Maximal Extractable Value	selectTransaction createBlock	Transaction Proposer	(Daian et al., 2020)	Fairness Economics	●●○○○	●●●●○	●●○○○	●●●●○	0.32	All
Block Endorser	Censure of Blocks	endorseBlock	Block Proposer Transaction Proposer	N/A	Fairness	●○○○○	●●●●○	●●●●○	●○○○○	0.16	Explicit Endorsement
Transaction Endorser	Censure of Transactions	endorseTransaction	Transaction Proposer	(Piriou et al., 2021)		●○○○○	●●●●○	●●●○○	●○○○○	0.16	Explicit Endorsement
Transaction Proposer	Double Spending	createTransaction	All	(Chohan, 2018)	Fairness	●○○○○	●○○○○	●●●●●	●○○○○	0.25	All
	Front Running	createTransaction	Transaction Proposer	(Eskandari et al., 2020b)	Economics	●●○○○	●●●●●	●●●○○	●●●●●	0.60	All
Blockchain Maintainer	Skip Transaction Validation	validateTransaction	None	(Luu et al., 2015)	Security	●○○○○	●●●○○	●●●●●	●●○○○	0.40	All
	Skip Block Validation	validateBlock	None			●○○○○	●●●○○	●●●●●	●●○○○	0.40	All
	Skip Transaction Execution	validateTransaction executeTransaction	None			●○○○○	●●●○○	●●●●●	●●○○○	0.40	All
	Skip Transaction Diffusion	diffuseTransaction	Blockchain Maintainer	(Ersoy et al., 2018)	Fairness	●○○○○	●●●●○	●●●●○	●●●●○	0.64	All
Oracle	Corrupted Oracle	Dedicated Oracle behavior	Contractor Investor Investee	(Caldarelli, 2020)	Economics	●●○○○	●●●●○	●●●●●	●●○○○	0.40	All
Investee	No / Partial Redistribution	redistribute	Investor	N/A	Fairness Economics	●●○○○	●●●●○	●●●○○	●●○○○	0.32	All

TABLE 4.2: The taxonomy of role-based incentive vulnerabilities.
Very Low: ●○○○○ , Low: ●●○○○ , Medium : ●●●○○ , High: ●●●●○ ,
Very High : ●●●●●

proposes it to its neighbors.

Censure Transaction. Through a deviation of the *selectTransactions* behavior, a Block Proposer may censure some transactions and therefore impacts Fairness. This is the case when a Block Proposer purposely excludes from its selection mechanism specific transactions coming from Transaction Proposer, even though they are financially attractive. This is an identity/address-based censure whose purpose is to delay or even forbid transactions involving a specific sender or receiver. While several blacklisted addresses are already purposely excluded from the network, the same behavior applied to non-criminal addresses is an illegitimate censure.

For this deviation to be impactful, a majority of block proposer must be willing to enforce the censure due to the complexity associated with having an overwhelming majority in blockchain systems. While significantly delayed, the agents or groups targeted by such censure can still rely on the remaining nominal participants or become a Block Proposer. However, a single block proposer may choose to censure any other participant; this requires few resources and has little to no impact.

Selective Block Propagation. The block proposal to the network might be intentionally skewed through a deviation of the *proposeBlock* behavior. For example, suppose an agent wishes to delay a competing Block Proposer and Blockchain Maintainer. In that case, it might propose its new block to all its peers except that competing one, thus slightly delaying its competitor's knowledge update. On a large scale, the targeted agent(s) may have a significant delay with the rest of the network, thus lowering their potential for valid block creation.

Consensus Delay. Consensus delay, or halting, is mainly related to PBFT consensus-inspired blockchain systems, where block proposers either propose conflicting blocks or do not propose through a combination of deviations from the *createBlock* and *proposeBlock* behaviors. A consensus-level attack impacts every participant. In such a configuration, consensus participants, often called *validators*, may collude to reach the 33% threshold of malicious nodes in the committee.

Selfish / Stubborn Block Creation A Block Proposer might not propose a block linked to the current consensual head of the public main-chain but rather on an

adversarial fork that is potentially private. This is done with another combination of deviations from the *createBlock* and *proposeBlock* behaviors. Other Blockchain Maintainers and Block Proposers are the primary victims of such a deviation.

Such a deviation is mainly linked to Proof-of-Work (PoW) blockchains and has been studied extensively (Eyal and Sirer, 2014). Recent work on the Ethereum 2.0 consensus mechanism indicates that a similar behavior is possible (Neuder et al., 2021; Schwarz-Schilling et al., 2021). A similar attack on Ethereum 2.0 will be covered in depth and subject to an experimentation detailed in Chapter 6.

Maximal Extractable Value Another vulnerability targeting the economics of public blockchains is the possibility of reordering transactions for the highest financial gain[1] (Daian et al., 2020). This optimization results from a deviation in *selectTransaction* and *createBlock* directly impacting the Transaction Proposers.

While it is rational for a miner to do so, Miner / Maximal Extractable Value (MEV) takes advantage of front-runners mostly looking for profitable arbitrage opportunities. This dynamic eventually raises the blockchain fees and reduces accessibility.

While MEV and front-runners serve Decentralized Finance (DeFi) economic equilibrium, they hamper the overall economy. They may create blocks with such attractive rewards that other miners might be incentivized to attempt to create a fork and capture the reward for themselves. Additionally, the impact on fairness is evident as the order of transactions is purposely modified.

Maximizing the reward from a block creation is rational individually or in a group, such as a mining pool. Because of this, the system becomes less accessible due to higher fees but it is still usable.

4.3.2 Block Endorser

Nominal behavior. Block Endorser vouches for blocks to be included in the chain following a block endorsement policy.

Censure Block. Block Proposers might purposely refuse to endorse blocks with specific characteristics by deviating from the *endorseBlock* nominal behavior, thus directly impacting the Block Proposer who created the block and, indirectly, the Transaction Proposers whose transactions are included in it.

Depending on the endorsement policy, such actions prevent the block from proceeding into the blockchain for non-consensual reasons and therefore impact the Fairness of the system. If such agents were to misbehave, they could impact or even stop the production of blocks.

This censure is relatively easy to set up for an individual agent. However, it has little to no impact as Block Proposers can and should always submit their proposals to several endorsers. Conducting this attack at a large scale requires most of the Block Endorsers to deviate from the nominal behavior, which is unlikely to happen thanks to the decentralized nature of the system.

4.3.3 Transaction Endorser

Nominal behavior. Transaction Endorser vouches for the inclusion of a transaction following an endorsement policy.

Censure Transaction. Similarly to the *Endorse Block* behavior, *Endorse Transaction* is subject to a malicious deviation from the *endorseTransaction* behavior, leading

[1] Quantifying Blockchain Extractable Value: How dark is the forest? - `https://arxiv.org/abs/2101.05511` last accessed on : 10-28-2022

to censorship of one or several Transactions Proposers. Any endorser could refuse to endorse specific transactions. Such actions could forbid the transaction to proceed any further in explicit endorsement schemes, such as in the Hyperledger Fabric blockchain[2].

However, at a larger scale, its impact is more severe as it is possible to lock participants out of the system. Still, this requires a majority of endorsers to deviate from the nominal behavior, thus reducing the risk.

4.3.4 Transaction Proposer

Nominal behavior. Transaction Proposer creates a valid transaction with a payload and the right fee for its inclusion in a block and then proposes it to the system.

Double Spending. In the context of a blockchain that allows forking in its protocol, an agent might deviate from the nominal *createTransaction* behavior for a double-spending attempt (Chohan, 2018). This deviation is usually paired with a form of forking attack such as selfish mining by knowingly proposing two conflicting transactions on different candidate chains. Such a deviation, if successful, impacts every participant of the blockchain system.

Front Running. As transactions are public and broadcast through the network before their inclusion in a block, every participant is aware of future events before they occur. For example, this allows a front-runner to take advantage of incoming large buy/sell orders on decentralized exchanges to front-run (Eskandari et al., 2020b) such transaction through another deviation of the *createTransaction* behavior, impacting other Transaction Proposers.

Front-runners can get priority through the fee mechanism, that is the primary selection criteria of Block Creator when selecting which transactions to include in a potential new block.

Note that front running is not a deviation. This behavior is rational and allowed by the protocol, but it is obviously harmful and can be therefore considered as an incentive vulnerability.

Front running is deeply linked to MEV as any front-runner is theoretically willing to give up to 99.99% of its profit as a fee to the Block Creator, thus increasing its power and influence in public blockchain Systems.

However, the impacts vary depending on the scale of the attack, *i.e.,* the number of participants involved in front-running transactions, looking for opportunities.

While a few front-runners may only have a mild impact on the overall system, when this strategy is widely adopted, there are consequences for the front-run users and the global economy as it fuels artificial fee growth. Front running is a serious issue regarding both the Economics and Fairness of blockchain systems.

4.3.5 Blockchain Maintainer

Nominal behavior. Blockchain Maintainer validates all newly received blocks and transactions. Valid transactions are stored in the memory pool, valid blocks are appended to the local blockchain, and all its transactions are executed.

Skip Transaction Validation. As transaction validation is not rewarded, rational agents may be incentivized to skip it by deviating from the *validateTransaction* behavior, potentially sacrificing the overall security and correctness of the system to gain an advantage in both time and computing resources. Such a deviation has no

[2] Hyperledger Fabric, `https://www.hyperledger.org/use/fabric`, accessed on 09/12/2022.

impact on other participants as long as it is local. However, if it were widespread, all participants would be impacted as the ledger coherency is no longer ensured.

This vulnerability is known as the Verifier's Dilemma (Luu et al., 2015). The potential inclusion of invalid data into the blockchain is hazardous as it threatens the system's stability. As stated in Chapter 2, participants that maintain the blockchain are interested in maintaining the system's stability but also in keeping the unrewarded amount of work to a minimum.

Skip Transaction Execution. In the nominal case, when an agent receives a transaction linked to a smart contract invocation, it should execute them using this behavior. The agent might not know in advance if the contract contains faulty logic, such as a hack of the execution environment to produce a potentially harmful result or simply invalid actions. The transaction execution time is also unknown and may be costly for the agent validating it. As smart contracts execution are linked to events and transactions, the primary vulnerability of this behavior is similar to that of the *validate(Transaction)*, where an agent would skip the execution by deviating from both the *validateTransaction* and *executeTransaction* behaviors.

Skip Block Validation. Validating a block may be costly for a Blockchain Maintainer. So, to gain a slight advantage, it may simply skip this step and append/propagate invalid blocks by deviation from the *validateBlock* behavior. Validating a block implies validating its structure and the embedded transactions, which eventually requires executing them.

Skip Transaction Diffusion. When transaction diffusion is not incentivized, rational agents may be skewed toward selfish behavior. This involves not sharing a new transaction with its peers by deviating from the *diffuseTransaction* nominal behavior. This deviation may even be profitable for *Block Creators* in open PoW systems as it reduces competition on the memory pool.

Current blockchain systems do not explicitly reward transaction diffusion. Instead, they implicitly rely on the stake that contributors (*i.e.*, Block Proposers and Blockchain Maintainers) have in the system. No transaction diffusion would hamper the system's usability by its users and possibly lead to centralization. However, for the reasons mentioned above, such an attack is improbable as it is against the interest of every rational contributor.

4.3.6 Oracle

Nominal behavior. The *Oracle* role holds the behavior collection responsible for Oracle functionalities, that is, bridging outside information to the blockchain.

Corrupted Oracle. An oracle node might be corrupted and transmit erroneous data on purpose by deviating from one of its dedicated behaviors or simply due to faulty logic. This would lead the blockchain system to make decisions based on incorrect information. Additionally, the data source might be corrupted while the Oracle is working nominally.

Both cases are nearly indistinguishable from one another and can lead to serious consequences. Trusting external oracle data is known as the Oracle problem. It poses a paradox between the necessity of oracles for real-world usage of the blockchain and the trustless nature of blockchain systems as described in (Caldarelli, 2020).

4.3.7 Investee

Nominal behavior. Investee receives investments from investors, provides a service, and redistributes the rewards to investors proportionally to their respective contributions.

No, Partial Redistribution. If an investee does not properly redistribute wealth earned thanks to its investors because of a deviation from the *redistribute* behavior, it may gain a financial advantage. However, this would come at the cost of a loss of reputation in the open blockchain system and hurt both the Fairness and Economics of the system due to its impact on Investors, Contractors, and other Investees. Such a behavior could be easily monitored and blacklisted. This has already been observed in the Tezos blockchain[3].

4.4 Conclusion

This chapter introduced a taxonomy of blockchains incentive vulnerabilities for networked intelligent systems based on the AGR4BS presented in Chapter 3. It can help researchers and developers better understand the different types of blockchains available and make informed decisions when designing these systems. The presented taxonomy computes the priority scores for each incentive vulnerability and characterizes them as role deviations concerning nominal behavior and inventives. It then lists and ranks several known vulnerabilities but provides a way to quantify and classify newly found ones. This taxonomy provides the foundation for characterizing incentive vulnerabilities and supports a role-based classification scheme. Based on the results, the rest of the thesis will be focused on high priority scores vulnerabilities (Table 4.2) such as *Consensus Delay* and *Selfish Block Production* both linked to the *Block Proposer* role. Given the scale and complexity of blockchain systems and their participants' autonomy, the approach best suited to incentive vulnerability exploration and discovery is, in our opinion, Multi-Agent Reinforcement Learning (MARL). This approach allows for the study of participants with rational objectives (*i.e.,* profit) or non-rational ones (*i.e.,* impact). MARL can be applied to ensure a secure update process if the incentive mechanism undergoes modifications. Additionally, the multi-agent interactions could be represented and learned to discover realistic behavior shedding light on previously unknown vulnerabilities, which could then be studied using more interpretable methods. To conduct such studies, a blockchain simulator compatible with MARL must be developed, this will be the topic of Chapter 5 that will add the missing piece for an automatic vulnerability study in Chapter 6.

[3]Tezos, `https://tezos.com/`, last accessed on 10-12-2022

Chapter 5

A generic blockchain simulator

This chapter describes the generic blockchain simulator we developed during the thesis, the code and embedded documentation can be accessed on GitHub at the following url : https://github.com/hroussille/agr4bs. We first cover the existing simulators and explain why they are not suitable for us in Section 5.1, then we discuss the intent with which our simulator was developed in Section 5.2. Finally, we go over the generic and blockchain specific components in Section 5.3 and Section 5.4 to give a clear overview of its architecture and how it can be used to simulate any blockchain systems.

5.1 Related Work

There exist several so-called generic blockchain simulators in the literature. Most of them are covered in (Albshri et al., 2022). They do adopt an event based mechanism (Babulak and Wang, 2008), aligned with the nature of blockchain systems. However, they are not always adopting an agent based approach, and some of them only consider the network itself, focusing on transactions and block diffusion times according to a specific network topology, and not on incentive analysis, sometimes not even allowing such studies due to design choices.

The closest simulators to our needs are BlockSim (Alharby and Moorsel, 2019), MAX (Gürcan, 2024) and DAGSim (Zander et al., 2019).

BlockSim, because it is written in Python, supports multiple models such as Bitcoin and Ethereum 1.0 and is easy to use or modify in our opinion.

Still, it does not use an agent based approach, and therefore does not have the Role and Behavior granularity needed for specific incentive based studies.

MAX on the other hand is built on MaDKit (Gutknecht and Ferber, 2000), and adopts a clear agent-oriented approach now grounded in the AGR (Ferber et al., 2004) model. Despite the fact that MAX only lacks performance because of its non-event based time management, it is arguably the most expressive and extensible model that exists today. This because the policy of a single agent can be modified as needed, granted that the researchers are able to read and modify the existing code. It also supports several models such as Bitcoin, Hyperledger and Tendermint.

Finally, DAGSim is also an agent based simulator allowing for a fine granularity. But it is strongly opinionated towards DAG based chains and therefore, lacks genericity for our purpose.

None of those is designed with AI in mind, and while adding such compatibility is possible, we assumed that creating a new simulator based on the AGR4BS model from Chapter 3 and inspired by the state of the art would be beneficial for further study and hopefully, for the blockchain research ecosystem as a whole.

It must be noted that another generic simulator was developed concurrently to ours: JABS (Yajam et al., 2023) features several consensus algorithms such as Nakamoto style, PoS and PBFT to only name a few. This simulator is very efficient, allowing several thousand of nodes to run in a single simulation. However, it does so using a simplistic representation of blockchain nodes, which is not suitable for our needs as it lacks expressivity.

5.2 Design rationale

5.2.1 Modularity

Due to the diverse nature of blockchain systems and the multitude of possible roles that can be played within them, the simulator is modular by design through groups, roles and behaviors. One should be able to easily swap the blockchain, state or network implementation or plug in additional roles. This modularity is key for the simulator to stay relevant in the future, as blockchain systems are subject to rapid evolution. For this reason, every high level component is exposed as an abstraction that developers can extend to fit their specific needs.

5.2.2 Reproducibility

As this simulator is built as a tool for research purposes, the results that it may yield must be explainable and, most importantly, reproducible by the scientific community. While a multithreaded program can be deterministic, there are many side effects induced by the Operating System which should be taken into account (*e.g.*, Interruptions, Preemptive Process Scheduling). This makes the single threaded approach both simpler to implement and unaffected by the aforementioned side effects. Therefore, the simulator runs in a single thread, but one can use it in a multi-threaded fashion if reproducibility is less important than performance.

5.2.3 Reinforcement Learning Compatibility

The simulator must be usable for RL studies. Since the most widely used frameworks and sets of environments, such as Gym (Brockman et al., 2016), are written in Python, we use this language to favor accessibility and ease of use.

It must be noted however that we do not enforce the Gym environment API, as it would put unnecessary constraints such as synchronous agent actions, which does not fit the intended goal of simulating blockchain systems which are asynchronous by nature. The simulator being modular, one can always create a custom environment implementation to enforce any required constraint.

5.3 Generic components

In this section, we give a description of the high level components of the simulator to give a clear understanding of how they interact with each other to ultimately model any blockchain system while not being limited to it.

5.3.1 Messages

This simulator is fully event-driven. In the scope of our simulator, an event is always associated to a *Message*, and vice versa. The UML description of a *Message* can be

found in Figure 5.1. We describe in the following sections how a *Message* reception effectively triggers behaviors of an *Agent* upon reception.

A message must be sent by and to someone which is represented by the *origin* and *recipient* properties, which are the unique identifiers of the concerned *Agents*.

The reception *date* is automatically computed by the *Network*, depending on the virtual network conditions in place when the message is sent. This attribute is the primary key used to order messages. In the case where two messages share the same *date*, a secondary key called *nonce* is used to compare them, thus ensuring reproducibility.

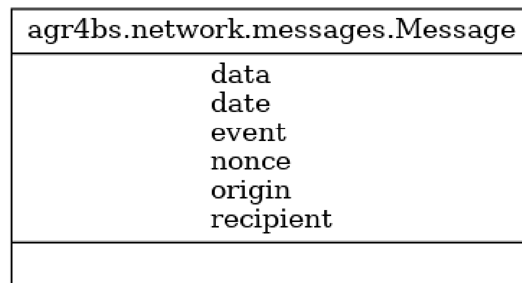


FIGURE 5.1: UML Diagram of the Message class

5.3.2 Network

The *Network* is a singleton or a set of singletons responsible for providing a means of communication between *Agents*. It can emulate network instability with arbitrary long network delays and message drop probability. The UML description of the *Network* can be found in Figure 5.2.

As mentioned in 5.3.1, the network enforces total ordering of all the *Messages* sent. This is done with the use of a priority queue: messages with the lowest *date* and lower *nonce* will be consumed first.

It is important to mention that the *Network* does not deliver any messages, its purpose is only to record and order messages.

Agents that are holding a reference to a *Network* can send two types of messages using *send_message* and *send_system_message*. The former is used to simulate communication: related messages can be delayed by up to N milliseconds and even dropped with a *drop_rate* probability. The latter is used for system messages. Those cannot be delayed nor dropped and are mostly used during initialization.

The entity consuming *Messages* from the *Network* can verify if there is any queued message with *has_message* and consume them one by one through *get_next_message*.

The *Network* class is essentially a multi producer multi consumer system where *Agents* are the message producers.

5.3.3 Factory

All dynamic components accept an optional *Factory* parameter on construction. The UML description of the *Factory* can be found in Figure 5.3. The *Factory* is centralizing all data structures and implementations required for the simulation. The default *Factory* only gives access to the *Network*, but it is intended to be extended to fit the developers' needs.

In the context of blockchain systems, the implementation of *Block*, *Transaction* and *Blockchain* are typically accessed through the *Factory*. This implies that the behaviors

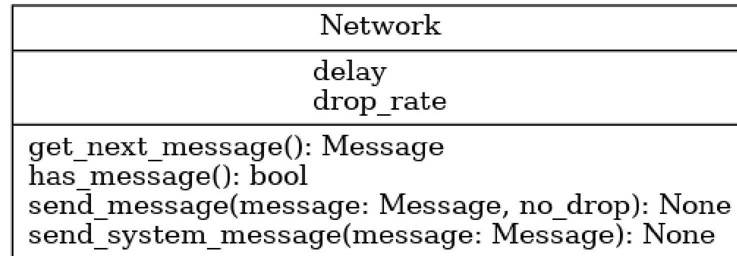


FIGURE 5.2: UML Diagram of the Network class

are unaware of the implementation details of the components they get through the factory, allowing to easily change the simulation details without impacting the *Roles*.

The default implementation of *build_network* is stateful and treats the *Network* as a singleton shared by all entities requesting it from the *Factory*.

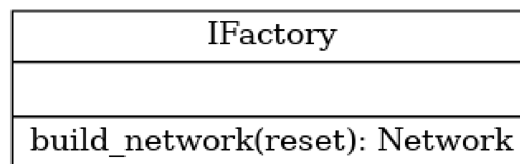


FIGURE 5.3: UML Diagram of the Factory class

5.3.4 Agents

The primary component is the *Agent*. This abstraction is able to endorse one or several *Role*. *Role* is defined as a collection of low level primitives, like in AGR4BS. The UML description of the *Agent* can be found in Figure 5.4.

An agent may represent a node in the blockchain, a smart contract, or any other entity deemed necessary to the simulation. It is uniquely identified by its name and able to take on several non-conflicting *Roles*, *i.e.,* an agent cannot endorse two different implementations of the same role.

The framework allows for various agent types that will be defined later on. The *Agent* class is a generic abstraction, akin to a role container, but is unable to communicate, as it lacks the primitives to do so. This allows for the subtypes of *Agents* to define or omit those communication primitives when needed.

One can add a *Role* to an *Agent* with *add_role*, or check if an *Agent* has a specific *Role* with *has_role*. *has_behavior* allows to verify that a specific behavior is available, according to the played *Roles*. If a *Role* is played by an *Agent*, the underlying implementation can be fetched with *get_role* and removed with *remove_role*.

Context

All *Agent* instances have a *Context*: a data structure, owned by the *Agent*, that *Roles* are allowed to modify when they are added, executed and removed. *Context* can be viewed as the *Agent* state.

Context is a secured container processing *ContextChange* that are standardized descriptions of changes to apply or revert to the *Context* with the *apply_context_change* and *revert_context_change* methods, respectively. It enforces idem-potency for additions and keeps a reference count for any property held within it, ensuring that removing a *Role* will not remove properties that are also used by other *Roles*.

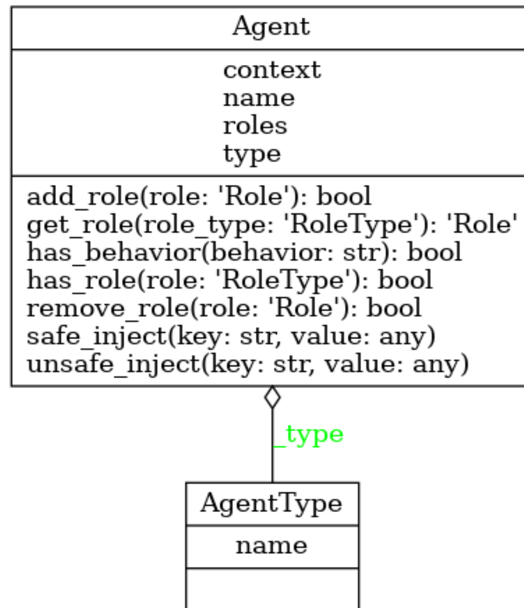


FIGURE 5.4: UML Diagram of the Agent class

When a *Role* is mounted to the *Agent* owning the context, the *Agent* queries the *Role* for its *ContextChange* and uses the *mount* method to get the dictionary of keys and values to add to the *Context*. Upon removal of a *Role*. the *unmount* method is called to get the list of properties to remove from the context.

The UML diagram shown in Figure 5.5 cannot represent the dynamics of the *Context*: one or more properties can be added to the class instance at runtime when a *ContextChange* is processed. Similarly some properties may be removed dynamically when a *ContextChange* is reverted.

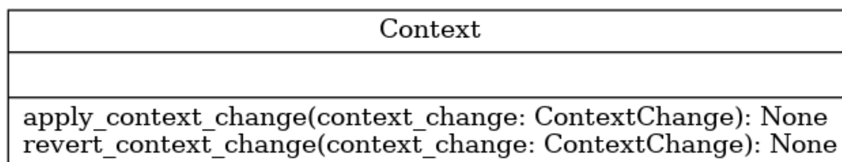


FIGURE 5.5: UML Diagram of the Context class

5.3.5 ExternalAgents

ExternalAgent is a subclass of *Agent* implementing all communication and event primitives. So, it is able to send and broadcast both normal and system *Messages*. Additionally, *ExternalAgent* can receive messages through the *handle_message* method, which will internally trigger any linked event with *fire_event*. The *fire_event* is also exposed publicly, so that developers may trigger any event at any time. This can be useful during testing or debugging.

ExternalAgent can also accept scheduled behaviors from *Roles*. When a *Role* exposes such a feature, the *ExternalAgent* is able to send itself a system message, containing the behavior to be triggered, with a delivery date set in the future. Upon reception of a scheduled *Message*, all the relevant handlers are executed through *run_schedulable_handler*.

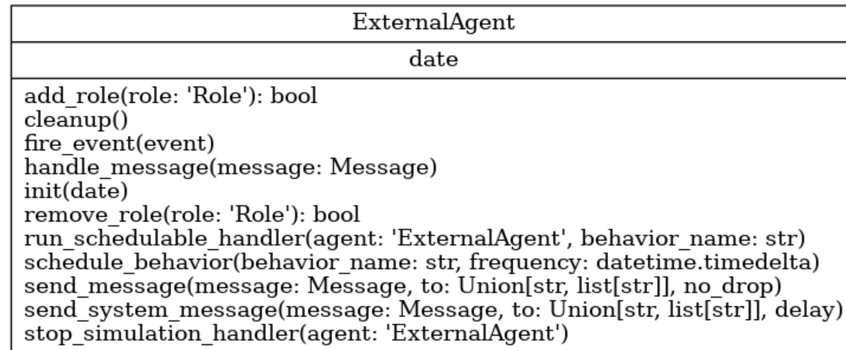


FIGURE 5.6: UML Diagram of the Agent class

5.3.6 Groups

The notion of *Group* is very well defined in AGR (Ferber et al., 2004) and used extensively in Chapter 3 to model various blockchain systems. In practice, groups are not a concrete entity and therefore do not require a specific class or data structure. For this reason, the simulator abstract groups as subnetworks shared only by a subset of *Agents* that they can use to communicate privately.

5.3.7 Roles

The *Roles* are the backbone of the simulator. Composing them into an *Agent* leads to the high level functionalities that the developers require. The UML description of the *Role* can be found in Figure 5.7. Each *Role* has a specific *type*, which ensures that an *Agent* cannot use conflicting *Roles*, and can specify the *agent_type* that it is expected to be mounted on. Because *Roles* are meant to be composed, they may exhibit some dependency relationships.

In that case, the developer can either bundle the dependent *Roles* together in a single *Role* or specify each *Roles* dependencies in the *dependencies* array, and mount the *Roles* in order of dependency. The generic *Agent* implementation checks if all the dependencies are met before adding a *Role*.

The most important part of a *Role* is its *behaviors*, which will be automatically extracted and bound to the *Agent* at runtime. The *Role* class does not have any *behavior* since it is expected to be inherited.

The *behavior's* property is technically a function which will dynamically scan the *Role* instance for static functions marked for exportation (*@export* modifier). This allows the developers to choose which behavior to expose or hide from other *Roles*.

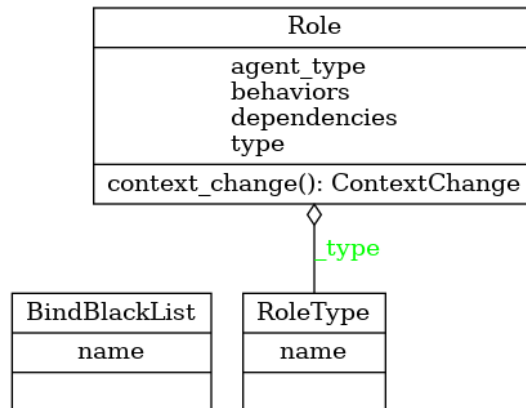


FIGURE 5.7: UML Diagram of the Role class

Declaring a behavior

Behaviors part of a *Role* will be added dynamically to an *Agent* when it endorses that *Role*. For this reason, *behaviors* must be declared as static methods and explicitly marked for exportation as shown in Listing 5.1.

```
@staticmethod
@export
def do_something(agent: Agent):
    """
        This behavior will be exported
    """
    pass
```

LISTING 5.1: Declaring a behavior

Binding behavior to events

If a behavior needs to be triggered whenever a particular *Message* or event is received, the simulator provides the means to do so with the *@on* modifier which, in this case, is used to set behavior metadata. This metadata will be read by the *Agent* that will internally bind the events to the specified behavior as shown in Listing 5.2. The *@on* modifier accepts an arbitrary number of event names if the developers ever need to bind a behavior to more than one event.

```python
@staticmethod
@export
@on('myEvent')
def event_bound_behavior(agent: Agent):
    """
        Runs when agent receives the event 'myEvent'
    """
    pass
```

LISTING 5.2: Biding behavior to events

Scheduling behavior

Alternatively, it is also possible to schedule behaviors to run periodically as shown in Listing 5.3.

```python
@staticmethod
@export
@every(minutes=2)
def scheduled_behavior(agent: Agent):
    """
        Runs every 2 (virtual) minutes
    """
    pass
```

LISTING 5.3: Scheduling behaviors

The agent registering that behavior will send itself system messages every 2 virtual minutes as defined in Section 5.3.4.There is no limit on the number of scheduled behaviors.

5.3.8 Environment

The *Environment* is a special kind of *Agent* used to manage the *Agents* and simulation parameters. It holds the list of simulated *Agents* and allows the propagation of high level functionalities to all of its members, such as initialization and cleanup operations.

The lifecycle of the *Environment* starts with the addition of agents, then the initialization process can be started with the *init* method, which will be propagated to any *Agent* contained in the *Environment*. From this point on, the simulation can take place until *stop* is called. The same environment can be reused given that the *cleanup* method is called, thus resetting the *Agents* and simulation state.

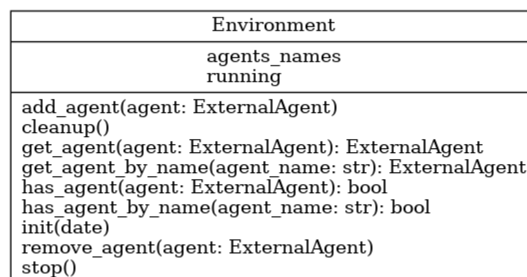


FIGURE 5.8: UML Diagram of the Environment class

5.3.9 Scheduler

Scheduler is the one running the simulation by holding references to both the *Environment* and the *Network*. Developers can call the *run* method that will execute the *step* method repeatedly until the stop condition is reached or there are no more messages to consume in the *Network*.

For each step, the top most priority message is extracted from the queue and the *current_time* is set to the message delivery *date*. The *Message* is then given to the recipient *ExternalAgent* through its *handle_message* method if it is still part of the simulation, otherwise the *Message* is discarded.

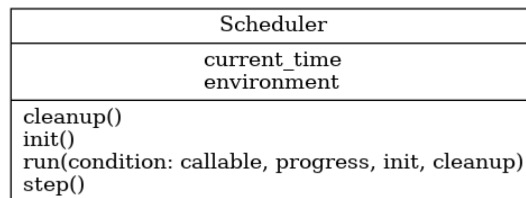


FIGURE 5.9: UML Diagram of the Scheduler class

5.3.10 Overview

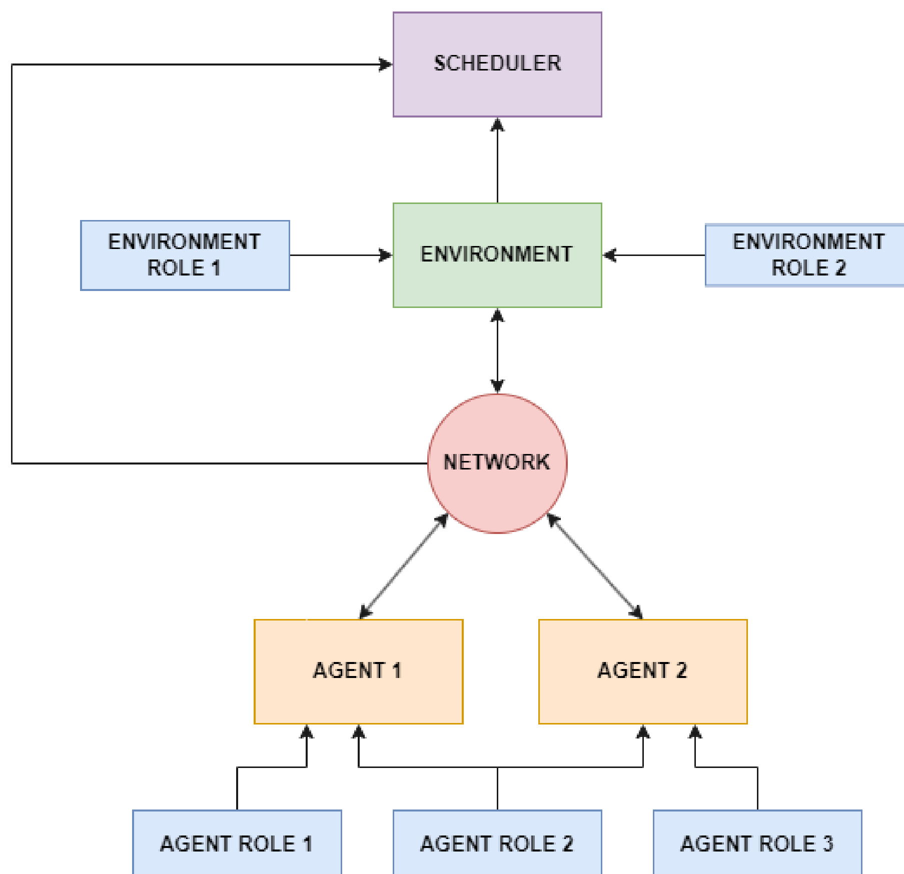


FIGURE 5.10: Overview of the simulator's architecture

```python
""" Create the Environment with a custom Role """
env = agr4bs.Environment()
env.add_role(agr4bs.roles.SomeEnvironmentRole())

""" Create 10 Agents with a custom Role and add them to the Environemnt
    """
for i in range(10):
    agent = agr4bs.ExternalAgent(f"agent_{i}")
    agent.add_role(agr4bs.roles.SomeAgentRole())
    env.add_agent(agent)

""" Define the starting time of the simulation """
epoch = datetime.datetime.utcfromtimestamp(0)
scheduler = agr4bs.Scheduler(env, current_time=epoch)

""" Initialize all the simulation components """
scheduler.init()

""" Define the stop condition : 10 (virtual) minutes of simulation """
def condition(environment: agr4bs.Environment) -> bool:
    return environment.date < epoch + datetime.timedelta(minutes=10)

scheduler.run(condition)
```

LISTING 5.4: API Overview

5.4 Blockchain Specific Components

This section describes the components required for simulating any blockchain system. Those include *Transactions*, *Blocks* and of course, the *Blockchain* itself alongside its *State*. Most of those components are public abstractions intended to be extended to fit specific requirements.

5.4.1 Transaction

A *Transaction* is a paid, public exchange of information and / or value between two participants. While the sender must be an an External Agent, the receiver may be:

- An External Agent
- A Smart Contract

The UML description of the *Transaction* can be found in Figure 5.11. Most cryptographic operations are abstracted away, this includes for instance signatures which are required for *Transaction* in all public blockchains.

When creating a *Transaction* one must provide some required parameters such as the *origin* (*i.e.*, sender's unique identifier), the *fee* to incentivize block producers to include the *Transaction* in a *Block*, the *nonce*, *value* and recipient of the *Transaction* denoted by the parameter *to*.

The *nonce* is used to provide a total order for the transactions issued by a single account. Its value should be set to the number of previously sent transactions by *origin* plus one.

The *payload* is optional. It is an array of bytes used to communicate information and parameters with smart contracts accounts.

As in real blockchain systems, this *payload* field can be used to permanently store vanity data on-chain assuming that the *Transaction* has been included.

When an *Agent* creates a *Transaction*, its *hash* is automatically computed, uniquely identifying the *Transaction* and allowing it to be diffused to other participants for an inclusion in the blockchain. Other participants receiving a broadcasted *Transaction* will verify its validity by calling *compute_hash*.Verifying if a *Transaction* is fully valid requires to know the full *State* of the blockchain, which will be defined later on.

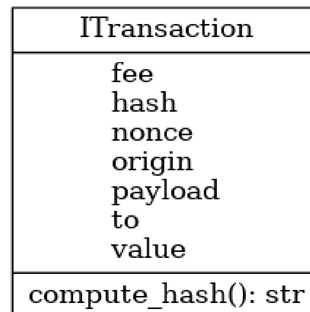


FIGURE 5.11: UML Diagram of the Transaction class

5.4.2 Block

A *Block* is essentially a container for *Transactions* and meant to be cryptographically linked to a parent *Block*.The only exception to this rule is the genesis block, which, by definition, is the first block of the chain. The UML description of *Block* can be found in Figure 5.12.

When an *Agent* creates a *Block* it must provide a list of *Transactions* to include, as well as the *creator* unique identifier (*e.g., itself*) and the *parent_hash* which is arguably the most important parameter defining the hash of the parent *Block*. The block will internally compute its *total_fee* based on the included transactions, and automatically set its *hash* using *compute_hash*.

Similarly to the *Transaction*, when an *Agent* receives a new *Block* it first checks its validity by calling *compute_hash*. Additional validity checks also require the *State* of the blockchain to be known and must be implemented in the relevant *Roles*.

The *height* and *invalid* properties are initially set to the height of the parent *Block* plus one and `false` respectively, but *Agents* receiving this *Block* will override those values in their local blockchain based on their view of the blockchain.

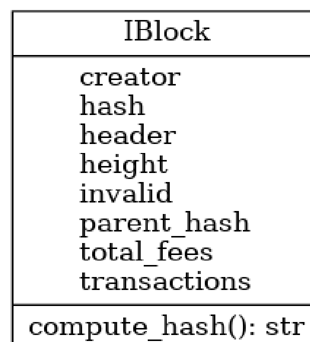


FIGURE 5.12: UML Diagram of the Block class

5.4.3 Blockchain

The basic blockchain implementation is abstract, meaning that it cannot be instantiated because some key components are missing, such as the fork choice rule and the constraints to respect on block addition. The UML description of the *Blockchain* can be found in Figure 5.12.

On creation, the *Blockchain* only requires a *genesis Block*, which is by default the *head* of the chain.

Block addition can be done with two primary methods: *add_block* and *add_block_strict*. *add_block_strict* only allows adding a block if the parent *Block* is known and included in the chain. *add_block* is used to add one or many blocks for which the parent block may not already be known. In that case, the block will be recorded in a staging area, and automatically unstaged if its parent *Block* is added. *add_block* will automatically release any staged blocks, and call *add_block_strict* for all *Blocks* that are ready for inclusion. Both functions will update the *head* of the chain, possibly using the blockchain specific implementation of *find_new_head* if the addition of blocks did not extend what was previously considered the *head*.

Aside from the core functionality of *Block* addition, the base *Blockchain* implementation also exposes a set of low-level helper methods. One can easily check if a *Block* is part of the *Blockchain* data structure with *get_block*, get all the children of an included *Block* with *get_children*, or even extract the whole chain using *get_chain* or just a subchain between two *Blocks* with *get_subchain*.

In most blockchain systems, a block can be considered invalid. This can be reflected on the data structure by using *mark_invalid* that will set the *invalid* flag of the block and all of its children to `true`.

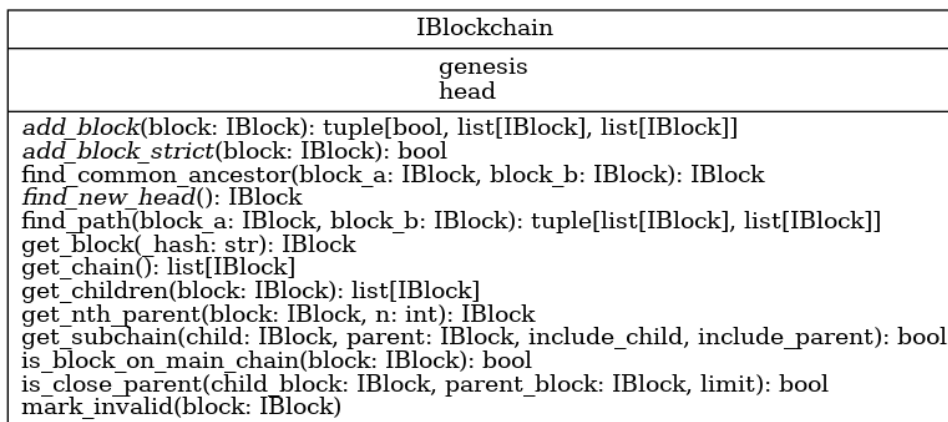


FIGURE 5.13: UML Diagram of the Blockchain class

5.4.4 State

To fully define the *State*, we must first define the notion of *Account*, *StateChange* and *Receipt*.

Account

The *Account* class holds all the information relative to a particular *Agent* on the *Blockchain*. The UML description of the *Account* can be found in Figure 5.14. An *Account* is composed of a *balance* (*i.e.*, how much that accounts owns in the native

currency of the blockchain), an optional *InternalAgent* which is the underlying implementation of the smart contract represented by an on-chain *Agent*, a *name* (*i.e.*, the on-chain unique identifier of the *Agent* owning that *Account*), a *nonce* keeping track of the count of public transactions of the *Account*, and finally a *storage* which is a simple key, value dictionary holding arbitrary information managed by the smart contract of that *Account* if any.

An *Account* can be copied using the *copy* method, and mutated. Mutation can be of several types such as a *balance* change through *add_balance* and *remove_balance*, a *nonce* change with *increment_nonce* and *decrement_nonce* or a *storage* change using *set_storage_at* or *update_storage*. All the aforementioned changes are the result of the execution of a *Transaction* against a given *State*. The base implementation of *State* only returns copies of the *Accounts* to avoid side effects and ensure consistency.

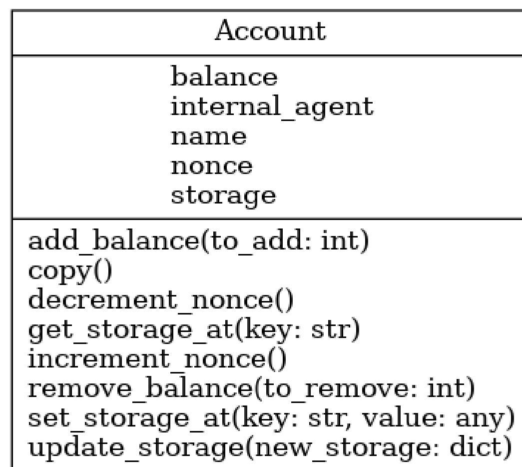


FIGURE 5.14: UML Diagram of the Account class

StateChange

A *StateChange* is a precise, atomic description of an operation to execute on the *State*. All *StateChange* can be reverted by applying an opposite *StateChange* in the case that a *Block* and all of its included *Transactions* are to be reverted following a blockchain reorganization. The UML description of the *StateChange* is given in Figure 5.15.

A *StateChange* only holds an *account_name* and a *type*. It is intended to be subclassed to fully cover the allowed state operations. The *revert* method of any *StateChange* must return an opposite *StateChange*. The default implementation defines the following changes *CreateAccount*, *DeleteAccount*, *AddBalance*, *RemoveBalance*, *IncrementAccountNonce*, *DecrementAccountNonce* and *UpdateStorage*. Developers are encouraged to add their own specific *StateChange* if they decide to extend the definition of an *Account* to fit some simulation specific need.

Account State Change

When an *Account* needs to be created or deleted, it is done using a *CreateAccount* or a *DeleteAccount*. The *State* will interpret them correctly according to their type, and mutate the underlying accounts. An *Account* is created the first time it is interacted with and may only be deleted if the transaction that created it is to be reverted.

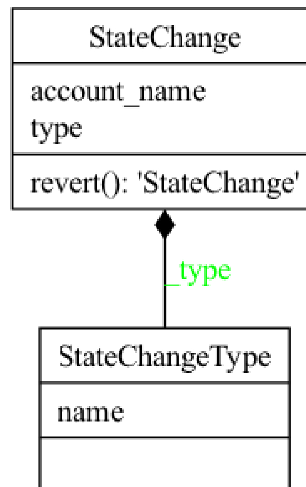


FIGURE 5.15: UML Diagram of the StateChange class

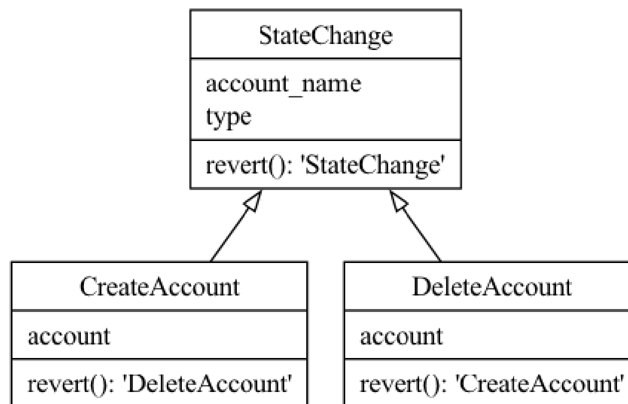


FIGURE 5.16: UML Diagram of the CreateAccount and DeleteAccount classes

Nonce State Change

Whenever a *Transaction* is processed against the current *State*, the first *StateChange* will be an *IncrementAccountNonce*, thus updating the number of *Transactions* sent by the specified *Account*. If a *Transaction* is reverted, the transaction count of the sending *Account* must be decremented, which is done with the opposite *StateChange* called *DecrementAccountNonce*.

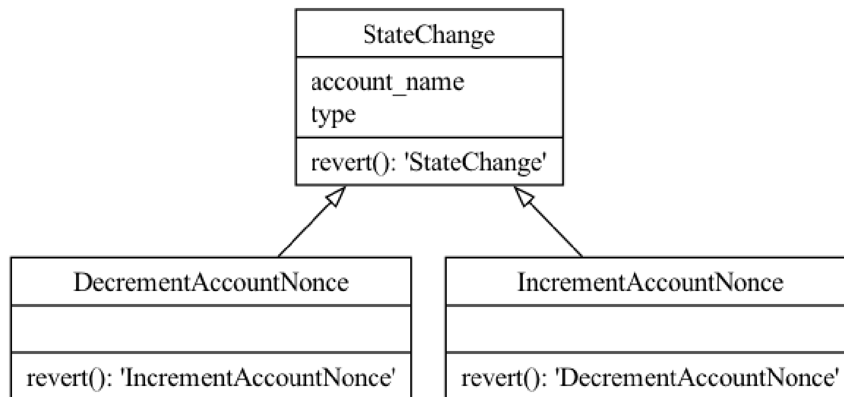


FIGURE 5.17: UML Diagram of the IncrementAccountNonce and DecrementAccountNonce classes

Balance State Change

Each *Account* balance may be updated whenever a *Transaction* is processed, either because it is sending or receiving value. *State* represents such operations using two *StateChanges*: *AddBalance* and *RemoveBalance*. The only required parameter is the *value*, which represents the delta to apply to the balance.

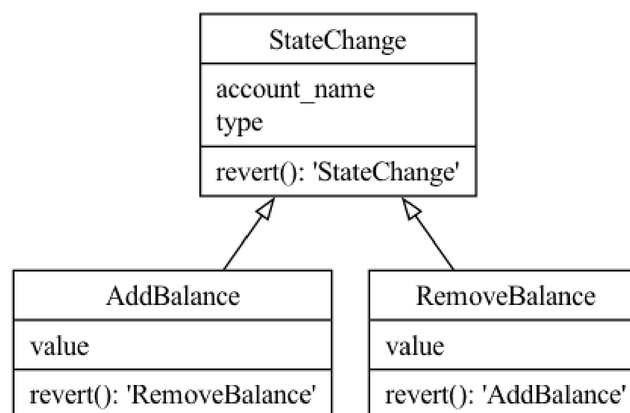


FIGURE 5.18: UML Diagram of the AddBalance and RemoveBalance classes

Storage State Change

The last *StateChange* is used to mutate an *Account* storage. It is only useful when using a *VM* implementation that supports smart contracts. The base implementation expects dictionaries of differences, one for the transition from the initial state to the resulting one, and another one for the opposite direction.

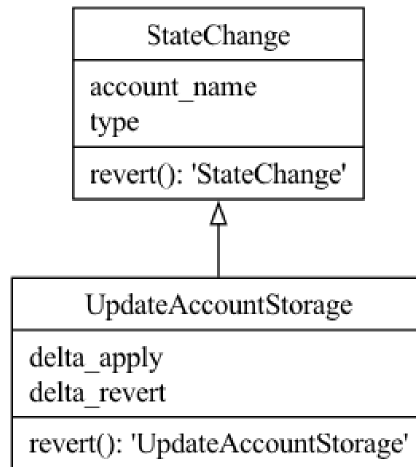


FIGURE 5.19: UML Diagram of the UpdateAccountStorage classe

Receipt

After a *Transaction* is processed by applying all of its *StateChange*, the *Receipt* is kept available in the *State*. The UML description of the *Receipt* can be found in Figure 5.20.

A *Receipt* contains overall information about the execution of the *Transaction* such as the *reverted* flag. When a Transaction was *reverted*, the reason can be specified using the `revert_reason` field of the *Receipt*.

Regardless of the execution status of the *Transaction*, all of the *StateChanges* that it generated are saved in the *Receipt* and accessible through the *state_changes* property, so that this can be used if the *Transaction* ever needs to be reverted.

A *Receipt* also holds a direct reference to the *Transaction* it describes under the *tx* property.

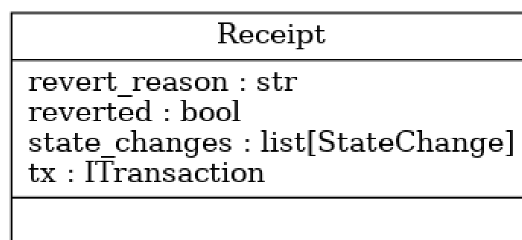


FIGURE 5.20: UML Diagram of the Receipt class

State

With all the above components defined, we can now explain the behavior of the *State*. The *State* data structure is a wrapper around *accounts* managing all read and write operations on behalf of the caller.

State exposes an RPC like API to read the underlying state, allowing anyone to query *Account* related information. One can verify if an account exists with *has_account* and, if it does, get its balance with *get_account_balance*, and its nonce with *get_account_nonce* or a specific storage key with *get_account_storage_at*.

Additional helper functions are available to get a deep copy of an *Account* through *get_account* or extract only its storage or smart contract implementation with *get_account_storage* and *get_account_internal_agent* respectively.

State mutation is done using *apply_state_change* or *apply_batch_state_change*, where the *State* owner (*i.e.,* an ExternalAgent) would execute one or several *Transactions* contained in a *Block* against a copy of the state and apply all the *StateChanges* contained in their receipts on the original. This mutation API also implies that applying or reverting a *Transaction* is functionally equivalent from the point of view of the *State*, it is simply applying a list of *StateChanges* in both cases.

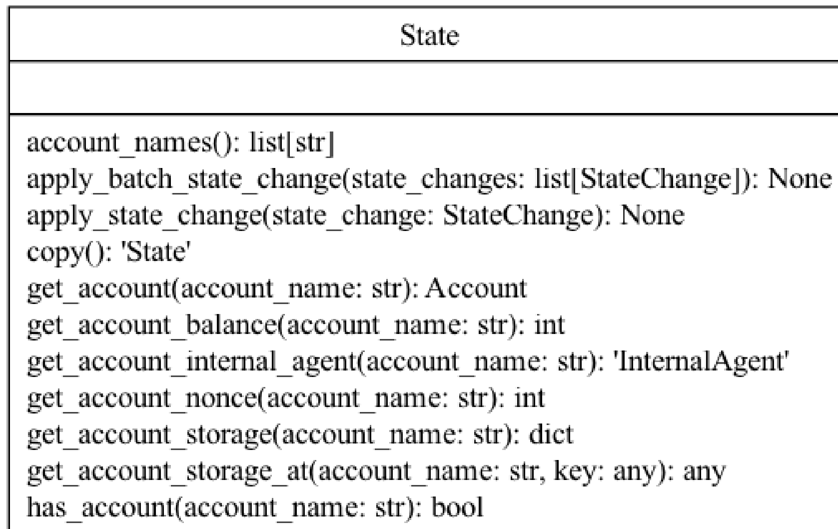


FIGURE 5.21: UML Diagram of the State class

5.4.5 Virtual Machine

The last blockchain specific is the Virtual Machine, or VM. Its only purpose is to execute a *Transaction* against the current *State* and return the *Receipt*. This Virtual Machine can be made to support smart contracts invocations if necessary. It exposes a single *process_tx* method that should execute an arbitrary transaction assumed to be valid, and return a *Receipt*.

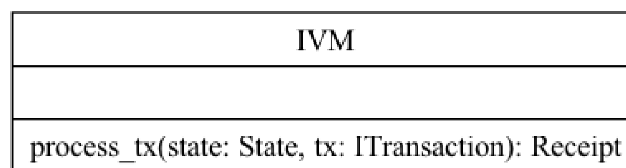


FIGURE 5.22: UML VM of the State class

5.4.6 Factory

All the blockchain specific components defined previously must be accessible to all *Agents* and *Roles*. The UML description of the blockchain specific *Factory* is given in Figure 5.23 to allow the building. The base *Factory* is therefore extended to return all blockchain specific components necessary during the simulation.

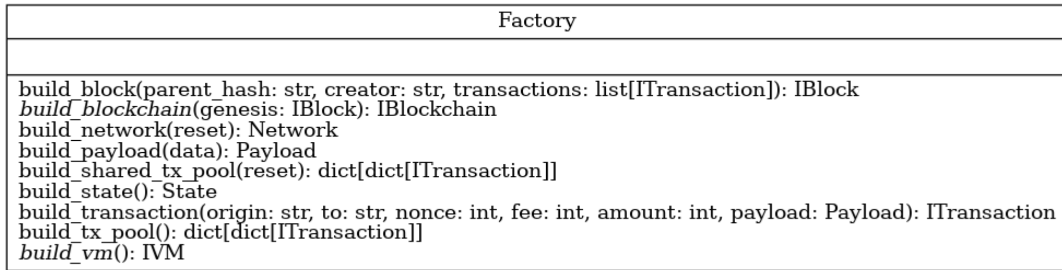


FIGURE 5.23: UML Diagram of the blockchain specific Factory class

5.4.7 Roles

The *Roles* need to be defined according to the specific blockchain to simulate. We recommend, but do not enforce, following the *Role* definitions from Chapter 2.

Chapter 6

Case Study: MARL Analysis of Incentive Vulnerability in Ethereum 2.0

In this chapter we showcase a MARL experiment aiming at studying an incentive vulnerability present in the Ethereum 2.0 protocol. This vulnerability is an Ethereum specific version of selfish block creation. It was chosen according to the taxonomy defined in Chapter 4 and the analysis is conducted using our role-based simulator defined in Chapter 5, both of which being grounded within the AGR4BS model from Chapter 3.

This attack fits within a high priority score as shown in Table 4.2, as it is relatively easy to setup by the attackers and involves a very well known blockchain, namely Ethereum 2.0.

In this attack, the attacker(s) are economically incentivized to fork the canonical chain in order to steal the attestations included in honest blocks, it uses some specific protocol functionalities against itself and cooperative malicious behavior to ensure that the fork is favored against what was previously considered the main chain.

This chapter starts by further explaining the Ethereum 2.0 protocol in Section 6.1, we then present the attack in Section 6.2. The settings of the experiment are detailed in Section 6.3 and the results in Section 6.4. Finally, we conclude this chapter in Section 6.5.

6.1 Ethereum 2.0

In this section we give more details about the Ethereum 2.0 protocol than in Chapter 3, so that the reader can better understand the considered attack. An even more detailed explanation can be found in (Pavloff et al., 2023), or in the Ethereum 2.0 book[1] which was used extensively for the implementation of the Ethereum 2.0 model in the simulator defined in Chapter 5. Throughout this section the term *Validator* will be used to refer to the *ETH Validator* defined in Chapter 3.

6.1.1 Epochs and Slots

The Ethereum 2.0 protocol actions all take place within well defined time frames called *Epoch* and *Slot* as shown in Figure 6.1. An epoch is composed of 32 slots each lasting 12 seconds. For each slot, a validator will be elected block proposer and given the opportunity to create a new block. The slot duration of 12 seconds can therefore be seen as the block time when the system is working nominally.

[1] `https://eth2book.info` last accessed on 14/09/2024

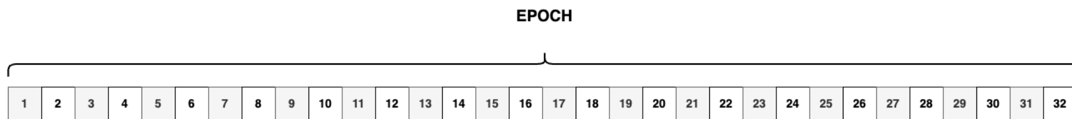


FIGURE 6.1: An Ethereum 2.0 Epoch composed of 32 Slots

6.1.2 Checkpoints

The Ethereum 2.0 consensus operates on epochs, the first block of an epoch is called a *checkpoint*, that is, a block for which votes will be issued to influence its status. A checkpoint can be in one of three states at any given time: *proposed*, *justified* or *finalized*.

A new checkpoint, refereed here as the target, is by default in the *proposed* state until a supermajority vote (i.e., > *2/3rd of the total stakes*) links it to a previously *justified* checkpoint, refered here as the source. The source checkpoint becomes finalized and all the blocks before it are considered final too. The target checkpoint can now be used as the source in the next epoch to finalize another chunk of the blockchain. This process is represented in Figure 6.2.

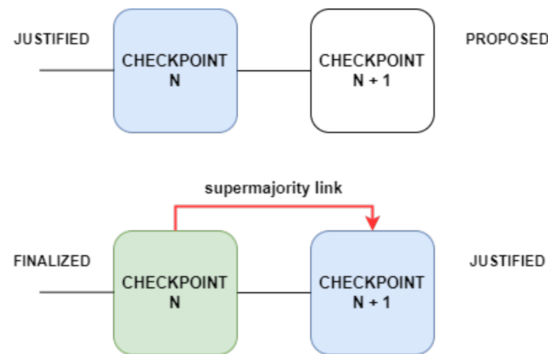


FIGURE 6.2: Ethereum 2.0 finalization process where a source, justified checkpoint (blue) is linked by a supermajority link (red) to a target, proposed checkpoint (white), leading to the justification of the target and the finalization (green) of the source.

This two step finalization process on checkpoints, and therefore on epoch, is what gives Ethereum 2.0 a finalization time of 12 minutes in the best case scenario.

6.1.3 Validator Duties

In Ethereum 2.0, validators have two main duties: Attestations and Block Proposition which are represented by the Block Endorser and Block Proposer roles in Figure 3.15.

Attestation

An Ethereum 2.0 attestation is a form of block endorsement as defined in Chapter 3. Each slot has $\frac{1}{32}$ of the validators considered as attesters, meaning that during a full epoch, all active validators will be participating by means of the attestations to the justification and finalization process described in Section 6.1.2. Ethereum 2.0 attestations are broadcast and can be included in the next block at the earliest as shown in Figure 6.3.

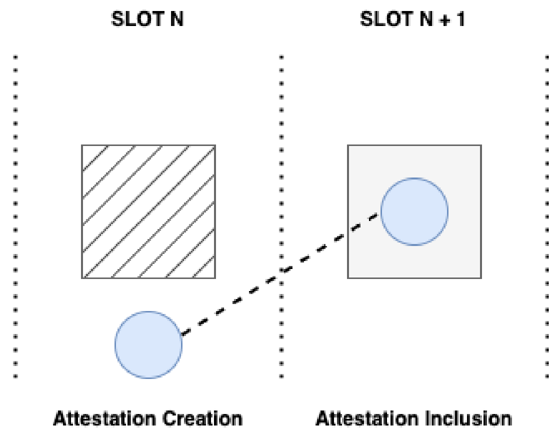


FIGURE 6.3: An attestation produced at slot N can be included in a block at slot N + 1 at the earliest.

An attestation contains 3 information: a *root*, a *source* and a *target*. The *root* is the current head of the blockchain, as seen by the validator at the moment the attestation was created. This will be used dynamically by the fork choice rule to resolve forks. The *source* is the last justified checkpoint, and the *target* is the checkpoint of the current epoch. Both will be used to finalize an additional portion of the chain. An honest validator will submit an attestation when receiving a new block or if one third of the slot time has passed.

For convenience, any subsequent representation of attestations will display them in the slot where they are included, and linked to the block targeted by their head vote unless otherwise specified.

Block Proposition

For every slot, one validator is considered the only legitimate entity to propose a new block. Being selected is an opportunity to create and propose a new block, but not an obligation in any way as there is currently no penalty for failing to propose a new block.

As Ethereum 2.0 is a PoS system, each validator has a stake of s_i. The total amount of active stakes is noted $\sum_{i=0}^{i=k} s_k$. The probability for a validator to be selected as proposer at any given slot is therefore $\frac{s_i}{\sum_{i=0}^{i=k} s_k}$. A block proposer is incentivized to include all pending and valid attestations. Figure 6.4 shows how proposed blocks include attestations created at the previous slot.

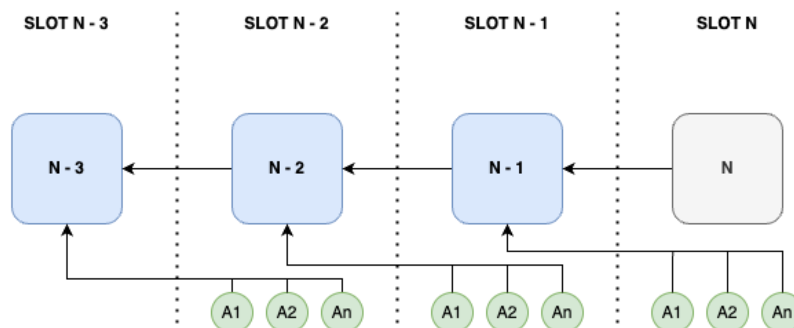


FIGURE 6.4: High level Ethereum 2.0 validator duties

6.1.4 The Fork Choice Rule

The fork choice rule is a core protocol mechanism that determines which chain is the canonical chain in the event of competing forks. Ethereum 2.0 uses the LMD GHOST (Buterin et al., 2020) (Latest Message Driven Greediest Heaviest Observed SubTree) rule.

LMD GHOST selects the fork with the most *head* votes from validators, prioritizing the heaviest chain as the valid one. It consists of the following steps:

1. Select the latest attestations sent by each validator.
2. For each attestation, add a weight proportional to the validator active stake to the block refered to by the attestation and all of its parents.
3. Starting from the latest finalized block, traverse the chain and chose the highest weight candidate in case of a fork.

' The resulting block is guaranteed to have no child, therefore, to be a suitable head of the chain. An illustration of this process is shown in Figure 6.5.

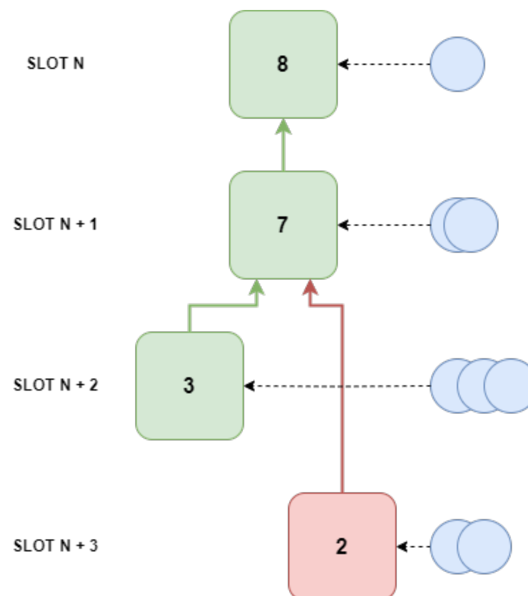


FIGURE 6.5: Ethereum 2.0 fork choice rule: LMD-GHOST: A fork between the block of slots $N + 2$ and $N + 3$ is resolved by applying the LMD-GHOST algorithm based on the validators attestations (blues). We assume that validators stakes are uniform for simplicity. Attestations are displayed where they are created.

An important note on the fork choice rule is that it takes into account the attestations of all validators regardless if those attestations are already included into a block. It precomputes any unrealized justification and finalization to run GHOST only on viable branches (i.e., branches that are descendant of the last justified checkpoint)

Proposer Boost

The original LMD-GHOST algorithm was proven to be vulnerable to so called balancing attacks (Neu et al., 2021), allowing for long lasting fork threatening the ability of the chain to finalize blocks.

To counter this, the proposer boost was created. It is essentially a significant amount of artificial weight added to timely blocks (i.e., *block proposed within the first third of a slot*). This weight is defined as $p * a$ where p is currently set at 0.4 and a represents $\frac{\sum_{i=0}^{i=k} s_k}{32}$, the average per-slot voting power. It is only taken into account during the slot that the proposed block belongs to, and favours liveness for forks resolutions as it gives a great power to honest and timely block poposers to influence the canonical chain.

6.1.5 Rewards and Penalties

As a public blockchain, The Ethereum 2.0 protocol must reward the expected behavior and punish the unwanted ones. All protocol rewards are expressed in terms of the *base reward* noted R, which is defined as the expected reward of an optimally performing validator.

$$R = S_i * (64/(4 * \sqrt{(\sum_{i=0}^{i=k} s_k)})) \tag{6.1}$$

Several components, each bound to specific duties, come together to compute the actual reward of a validator for a given epoch as shown in Table 6.1.

Duties	Weight
Timely source vote	$W_s = 14$
Timely target vote	$W_t = 26$
Timely head vote	$W_h = 14$
Participated in a sync committee	$W_{sync} = 2$
Proposed a block when it was due	$W_b = 8$

TABLE 6.1: Weights of the different validator duties.

We purposely ignore the sync committee rewards as it is linked to a technicality of the Ethereum 2.0 attestation aggregation protocol and is not relevant here.

The notion of timeliness is defined as the delay in slots between the attestation emission and its inclusion in a block of the canonical chain. If the timeliness requirement is not met, the reward is nullified for the specific duty.

The fork choice rule defined in Subsection 6.1.4 requires up to date head votes, therefore the timeliness requirement is set to 1. Target votes can be valuable as long as they refer to the current or previous epoch only, therefore the timeliness requirement is set to 32. Finally, the timeliness requirement for source votes is arbitrarily set to 5.

Ultimately, if a validator fulfilled all of its duties during a given epoch, it may expect a reward of:

$$\frac{W_s + W_t + W_h + W_{sync} + W_b}{64} = R \tag{6.2}$$

Block proposition may seem under rewarded, but proposers also get the transactions fees and $\frac{8}{64} * R$ for each valid attestation that they include in their proposed block. This incentivizes proposers to include as many valid attestations as they can and helps the protocol as it decreases the economic viability of witholding attacks.

The penalties are simple, if a validator misses a source or target vote, the penalty is equal to what it would have been rewarded if they were valid. Head votes and block propositions can only be rewarded: missing them does not incur any penalty.

Slashing

Some actions are indubitably malicious and lead to a significant penalty called a *slashing*. There are currently only three slashable offences, all relating to equivocation:

1. Double block proposition for the same slot
2. Double voting
3. "Surround" attestations

1) is self explanatory. 2) is triggered if one validator issues two or more attestations with different target checkpoints in the same epoch. This can happen during a long lasting fork, but only one attestation per epoch is allowed. 3) is slightly more complex: it is triggered if one validator makes an attestation that surrounds or is surrounded by one of its previous attestations.

A proven slashable offence immediately leads to 1/32 of the guilty validator's balance to be burned and it is registered for a forced exit.

6.2 Selfish Block Creation in Ethereum 2.0

This section explains the potential vulnerability inspired by (Carlsten et al., 2016) that we will be exploring trough a MARL based experimentation.

6.2.1 Attack description

As shown in the previous section, the Ethereum 2.0 protocol uses a fork choice rule, and therefore allows for forks. Those forks may happen because of the network delays that can affect any distributed system, but they may also be caused by malicious block proposers that purposely create a fork as depicted in Figure 6.6.

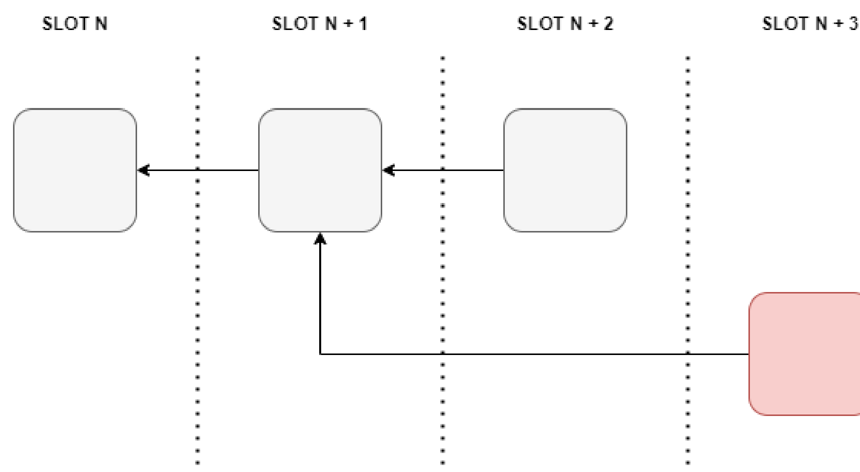


FIGURE 6.6: A malicious block proposal that purposely does not extend the head of the canonical chain.

This attack seems free at first, but honest validators that were attesters at slot $N+2$ will have issued and propagated their attestations for the block at slot $N+2$ as soon as they received it.

Since the fork choice rule described in 6.1.4 is stateful, most honest validators will consider the block at slot $N + 2$ to be the canonical chain and disregard the block at slot $N + 3$ as depicted by Figure 6.7.

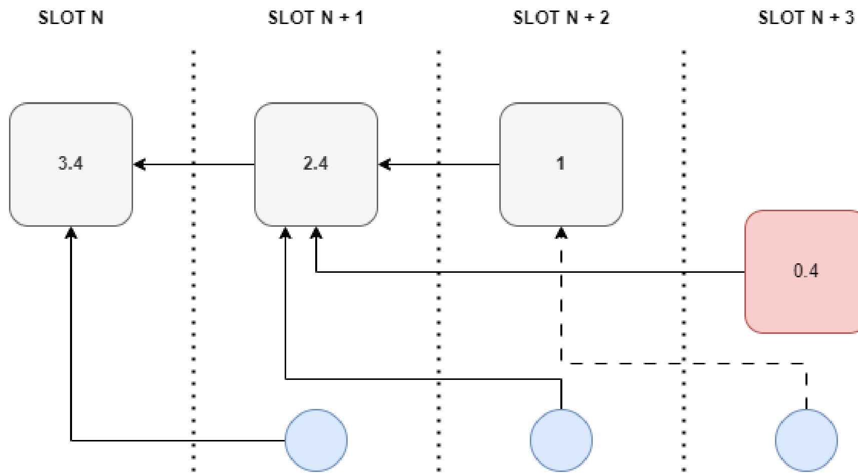


FIGURE 6.7: In this scenario with 32 validators, each having equal stakes the validators favors the block at slot $N + 2$ because the attestation issued at that same slot were not yet aware of block $N + 3$. The proposer boost ($0.4 * \frac{32}{32}$) is not enough to counteract the not yet included attestation (dashed line) and the fork choice rule does not select the malicious fork.

Note that if the malicious block proposer does include the honest attestations created at slot $N + 2$, the result is the same since the fork choice rule takes them into account regardless of their inclusion in a block. It may however want to include all attestations present in the forked block at slot $N + 2$ to get the $\frac{8}{64} * R$ reward for valid attestations. This strategy was omitted from the diagram for simplicity.

If we add accomplices then the malicious group may force the fork to be selected by the honest participants in two different ways.

The first case is when the attester at slot $N + 2$ is malicious too and knows that the proposer at slot $N + 3$ will attempt a fork. In this situation the malicious attester votes for a previous block to avoid giving any weight to the forked block $N + 2$ as depicted in Figure 6.8.

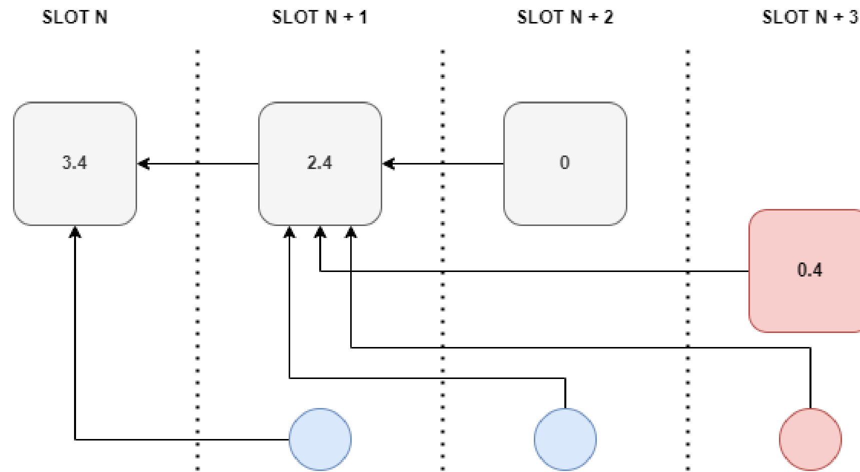


FIGURE 6.8: The malicious attester at slot $N+2$ does not respect for fork choice rule and votes for the block at slot $N+1$, therefore the malicious block at slot $N+3$ is favored by the fork choice rule because of the proposer boost, therefore, honest validators attest to it too.

The second strategy is possible if the attester at slot $N+3$ is also malicious and attests to the malicious block of the same slot as shown in Figure 6.9. The malicious attestation is taken into account by the fork choice rule before its inclusion in a following block. This leads to a tie between both candidate head block that is broken in favor of the malicious block when considering the proposer boost, assuming that the proposal was timely.

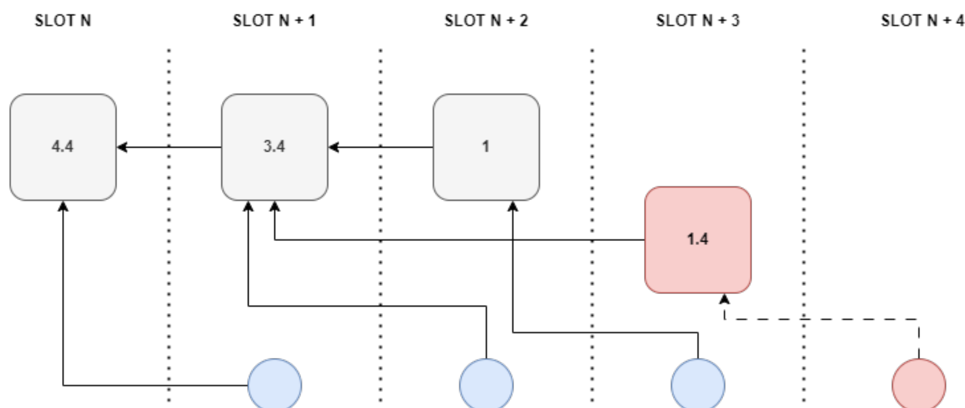


FIGURE 6.9: The malicious attester at slot $N+3$ immediately votes for the malicious block, disregarding the honest fork choice rule and favoring the malicious block.

6.2.2 Attack motivation

This attack requires cooperation from the malicious group, and can be motivated either by the will do disrupt the system, or by the profit that stems from the attestation(s) of the forked block which are included in the malicious block.

Since we are focusing on rational attacks, we only consider the economic motivation, where the forked blocks contain at least one attestation. Moreover, including two or more attestations, that is, stealing the forked block attestations, would make the malicious strategy more profitable than the honest one. The re-inclusion of the forked block attestations were purposely omitted from the previous figures to bring

light on the mechanics of the attack: Figure 6.10 shows the end goal of the attack. The protocol allows such re-inclusion since, when the fork happens, both branches have their own state, therefore the attestation(s) is valid in each of them.

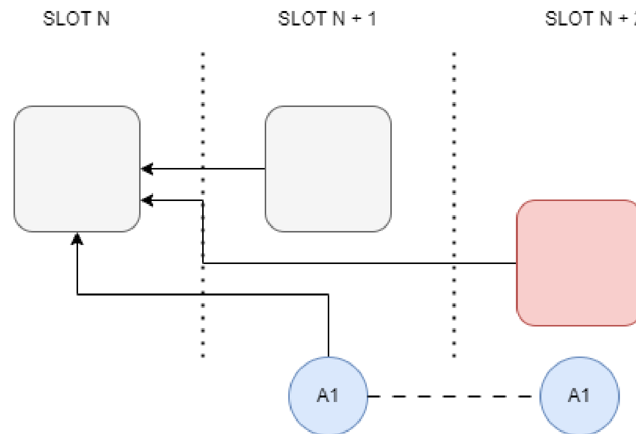


FIGURE 6.10: The malicious proposer includes the attestation(s) from the forked block in its own, hoping to steal the proposer reward.

6.3 Simulation Settings

This section details the settings and parameters used in our simulation. We cover the *Environment*, *Agents* and their *Actions* as well as the hyper parameters ruling this simulation.

6.3.1 Environment

The environment is a correct implementation of the Ethereum 2.0 protocol embedded within the simulator, described in Chapter 5, featuring all necessary components.

Epochs, Slots, Attester selection and Block Proposer election are handled by special role given to the environment. This role will trigger the slot update every 12 virtual seconds and the epoch update every 32 slots. Upon changing epoch, it will recompute the list of block proposers and attesters for all slots of the new epoch and communicate it to the validators using a system message. All simulated messages are subject to network delays averaging 100 milliseconds and can be lost with a probably of 1/100. Each simulation is composed of 500 repetitions of a 1 epoch long game.

6.3.2 Agents

From a blockchain point of view, the simulation contains 32 agents, each of them can be honest of malicious on a per-role basis. Both groups are validators and therefore play the roles of *Block Proposer*, *Block Endorser* and *Blockchain Maintainer*.

The honest agents strictly follow the protocol at all times using the default Ethereum 2.0 implementation: they can never deviate from it. The malicious group can deviate from the nominal behavior to create a fork, but never commit any slashable offense.

The deviations are implemented in subclasses of the *Block Proposer* and *Block Endorser* roles. The modified *Block Proposer* role will not automatically attach any new

block to the current consensual head, but may use the parent block of the head instead. The modified *Block Endorser* role can chose to deviate from the protocol implementation to issue a head vote targeting either the parent of the consensual head, a malicious block created at the same slot or, the consensual head if it decides to be honest. It will never misbehave on the *target* or *source* votes during the simulation. Out of the 32 agents, 8 of them are using the malicious *Block Proposer* role and 10 of them are using the malicious *Block Endorser* role.

Agents are arranged as shown in Figure 6.11 as it exposes both the proposers and the attesters to the four scenarios of interest during an epoch.

1. Malicious proposer and malicious attester at the same slot (slots 2, 14 and 26)
2. Malicious attester preceding a malicious proposer(slots 6, 18 and 30)
3. Malicious proposer with no malicious attester at the current or previous slot (slots 10 and 22)
4. Malicious attester with no malicious proposer at the current or next slot (slots 8, 11, 20 and 23)

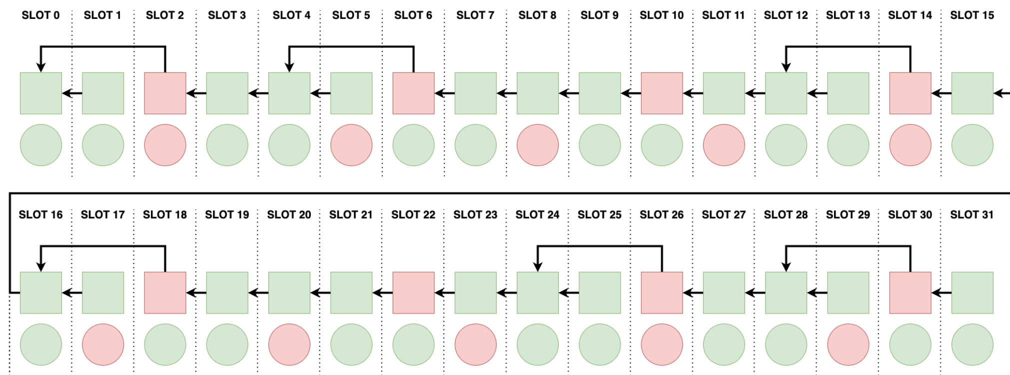


FIGURE 6.11: The 32 agents are deterministically arranged during an Ethereum 2.0 epoch with 8 malicious block proposers (red square), 24 honest block proposers(green square), 10 malicious block endorsers (red circle) and 22 honest block endorsers (green circle).

All blockchain agents are therefore proposing one block and one attestation per epoch.

From a reinforcement point of view, we only consider two agents represented by the malicious proposer policy and the malicious attester one (shared by the malicious agents).

6.3.3 Models

The behaviors of both the proposer and the attester are implemented by neural networks trained with Deep Q Network (DQN) Mnih et al., 2013. The neural networks are trained to minimize the Q value estimation error of state action pairs. In any given state, the selected action is the one that maximizes the expected reward, the policy $\pi(s)$ selecting the action for the current state is therefore :

$$\pi(s) = \operatorname*{argmax}_a Q(s, a) \tag{6.3}$$

Where s and a are the current state and action respectively. Both states and actions will be described shortly after for each model.

In practice, the action selection process is initially probabilistic, with probabilities proportional to the Q values estimations. During the training of the neural networks, it slowly transitions to a deterministic process and is fully deterministic at the end of the simulation.

We note $\pi_p(s)$ and $Q_p(s,a)$ the policy and Q value estimation of the proposer, and $\pi_a(s)$, $Q_a(s,a)$ the attester ones.

6.3.4 Actions

All actions are discrete and mutually exclusive, they only allow the agents to chose between the *honest* and *malicious* strategy.

The deviated *Block Proposer* may choose to be *honest* or *malicious* upon creating a new block.

- Honest: creates a new block extending the head of the canonical chain.
- Malicious: creates a new block extending the parent of the head of the canonical chain.

The deviated *Block Endorser* may also choose to be *honest* or *malicious* when attesting, but the execution of the malicious strategy will depend on the context of the blockchain at the time the action is done.

- Honest: create an attestation whose head vote is for the head of the canonical chain.
- Malicious : create an attestation whose head vote is for the current block if it is malicious or for the parent of the head of the canonical chain if the next proposer is malicious.

If an attester selects the *malicious* action in a context where neither of the next or current proposer is malicious, it will issue a head vote for the parent of the head of the canonical chain as this would lead to an incorrect head vote, and is therefore equivalent to a penalty.

6.3.5 Observations

Block Proposer

Block proposers observations are a vector $[M_{i-1}, M_i]$ where:

- $M_{i-1} \in 0, 1$, whether the previous attestation was honest (0) or malicious (1).
- $M_i \in [0, 1]$, the probability that the current attestation will be malicious if the current block is malicious.

Block Endorser

Block endorsers are a vector of observations $[F_i, F_{i+1}]$ where :

- $F_i \in 0, 1$, whether the current block is honest (0) or malicious (1).
- $F_{i+1} \in [0, 1]$, the probability that the next block will be malicious if the current attestation is malicious.

Observations in the asynchronous environment

From the above description of the observations, M_{i-1} and F_i can be inferred from the current state of the blockchain agents.

However, M_i and F_{i+1} cannot as they are both representing probabilities of future actions. $M(i)$ is therefore set to 0 if the current attester is not playing a deviated *Block Endorser* role, $\frac{e^{Q_a([1,0],1)}}{\sum_i e^{Q_a([1,0],i)}}$ otherwise. Similarly, F_{i+1} is set to 0 if the next proposer is not playing a deviated *Block Proposer* role, $\frac{e^{Q_p([1,0],1)}}{\sum_i e^{Q_p([1,0],i)}}$ otherwise.

The proposer and attester policies are therefore co-dependent. While this correctly models the decision process of this attack vector, it must be noted that this induces instability, non-stationarity, and possibly oscillatory behaviors in the training process.

6.3.6 Reward Functions

Malicious agents share a global, per epoch cooperative reward function with is defined as:

$$G_i = \frac{\sum_{k=0}^{k=N}(S_k^i - S_k^{i-1})}{N} \tag{6.4}$$

Where S_k^i is the staked balance of the malicious agent k at the epoch i. The staked balance increases with protocol rewards such as block proposal or valid and timely attestations. Malicious agents can increase their staked balance by being honest, but a more profitable strategy is to cooperate in order to create forks.

Since the above reward function may be difficult to learn and prone to instability due to the interactions between the proposer and attester policies, we decided to experiment with a second reward function, more biased towards the objective of creating forks defined as :

$$G_i = F_i^+ - F_i^- \tag{6.5}$$

Where F_i^+ is the number of successful forks during epoch i and F_i^- is the number of failed forks during epoch i. This second reward function heavily incentivizes agents to create forks but fails to express any rational economical objective.

6.4 Results

In this section we show the results obtained using both reward functions defined in subsection 6.3.6 and briefly discuss them. All results are averaged across 10 different simulations with similar settings, each lasting 500 Ethereum epoch for a real time duration of around 30 minutes. For all plots we display the 90% confidence interval to let the reader know about the stability or instability of the learning process.

Experiment 1: Average malicious gain reward

Using the average malicious gain protocol reward function defined in Equation 6.3.6 we did not observe any convergence. The agents failed to cooperate and never managed to outperform the honest agents as shown in Figure 6.12.

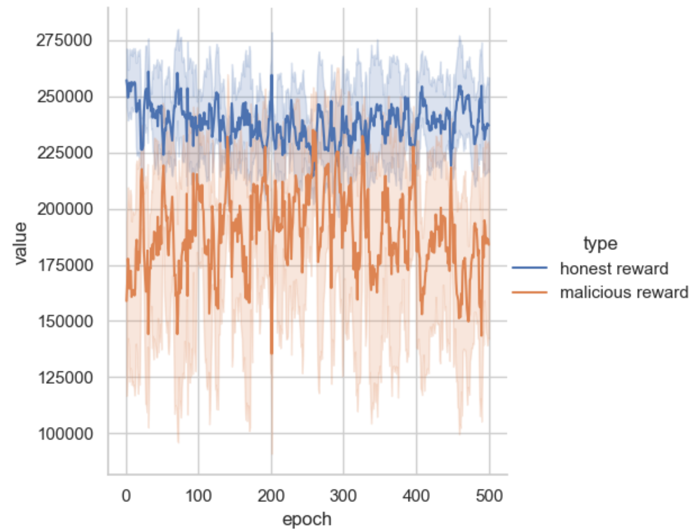


FIGURE 6.12: Average protocol reward of both honest and malicious blockchain agents in simulations using the average malicious reward criterion.

As expected, this sub optimal performance is linked to an inability to create more successful forks than failed ones as shown in Figure 6.13.

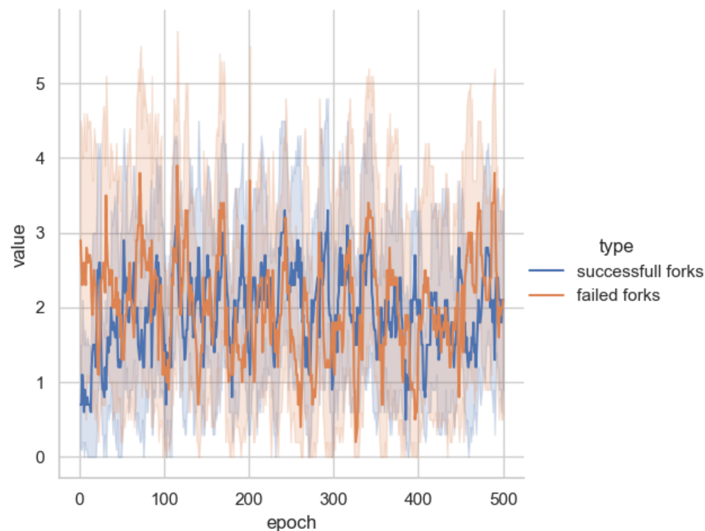


FIGURE 6.13: Number of successful and failed forks in simulations using the average malicious reward criterion.

Those results are inconclusive, but do not show that a rational financial criterion does not lead to the creation of forks. The suspected reasons for the lack of convergence will be discussed later in this section.

Experiment 2: Forks reward

Using the number of forks reward function defined in Equation 6.3.6 we observed that the agents managed to cooperate and consistently outperformed the honest agents as shown in Figure 6.14

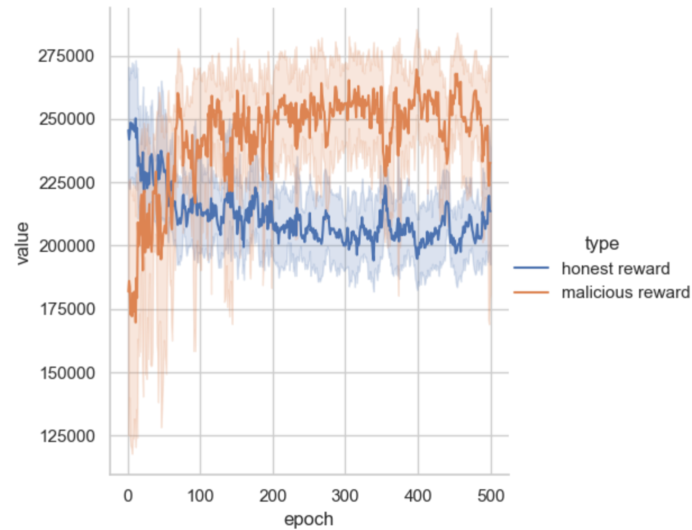


FIGURE 6.14: Average protocol reward of both honest and malicious blockchain agents in simulations using the number of successful and failed forks reward criterion.

Figure 6.15 shows that the malicious agents quickly learned to cooperate and create significantly more successful forks than failed ones.

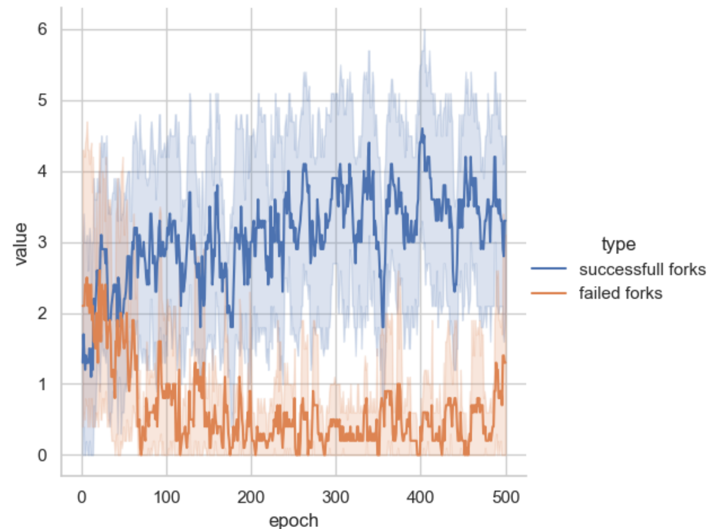


FIGURE 6.15: Number of successful and failed forks in a simulation in simulations using the number of successful and failed forks reward criterion.

However, they failed to reach a consistent and stable optimal policy where 6 successful forks and 0 failed forks would be created at each epoch.

6.4.1 Discussion

To the extent of our experimentation, the first reward function did not yield any positive result because it is highly volatile due to non-stationarity caused by the interference between both policies. Indeed, an optimally performing proposer policy would see it's reward estimation error change greatly simply because of the actions of a sub-optimal attester policy. It shows that MARL results are to be taken with caution. In this case, the most profitable strategy is for the malicious *Block Proposers* and *Block Endorsers* to cooperate.

The second reward function is biased as it is no longer an exploratory experiment, the agents are directly incentivized to maximize the number of successful forks. In this setting, the malicious agents outperformed the honest ones, but never truly converged on the optimal policies which would create 6 successful forks and no failed ones. The resulting strategies are sub-optimal and unstable as the number of forks shown in Figure 6.15 exhibit a relatively high variance throughout the simulation.

By increasing the simulation duration we observed oscillations where the performance of the policies dropped significantly before slowly regaining the levels shown in Figure 6.15 if they are still able to explore the state, action space (*i.e., if the policies are not yet deterministic*).

6.5 Conclusion

This experiment showed the usability of MARL within our simulator from Chapter 5 with both a negative and positive results. From the simulation results, we cannot directly conclude that an economical objective leads to the creation of forks from malicious agents, but we can conclude that an objective targeted towards forks leads to superior protocol rewards for the malicious agents.

The simulation showcased some limitations associated to the usage of MARL in highly complex blockchain environments such as the Ethereum 2.0 protocol.

Since blockchain systems are asynchronous, multi-agent systems, most environment that do not oversimplify the task at hand will encounter a similar degree of non-stationarity and oscillatory behaviors that can greatly impact the outcome of a simulation. While MARL is likely to be a suitable candidate for some scenarios, it does require additional work compared to more natural, synchronous environments and is by no means a silver bullet.

Other approaches that do not make strict assumption on the stationarity of the state transition and reward functions should be considered as well. A prime example of such method would be genetic algorithms (Lambora et al., 2019), which can naturally evolve and improve multiple policies in inherently unstable environments at the cost of increased computational requirements.

Chapter 7

Conclusion

This chapter concludes this thesis by providing a short summary of the different contributions in Section 7.1. We then discuss the limits of this work in Section 7.2 as well as the perspectives in 7.3.

7.1 Summary

Driven by the need to advance the research field in AI based study of blockchain systems, this thesis presented a global, role-based, organizational framework for blockchain system simulation using AI, including one MARL case study on Ethereum 2.0.

We presented the role-based modelization framework in Chapter 3, called AGR4BS. Its modularity and expressivity allows it to represent various blockchain systems such as Bitcoin, Hyperledger Fabric, Tendermint or Ethereum 2.0, which differ in permission scheme, algorithm and structure. Through its atomic components, namely *groups*, *roles* and *behaviors*, it allows the representation of various attacks and, more importantly, lays the necessary foundations on which rely the two other contributions of this thesis.

So, in chapter 4, we have proposed a qualification of blockchain protocol incentive vulnerabilities in the context of the AGR4BS model. The obtained taxonomy covers most known vulnerabilities, and links them to one or several existing roles. Through risk and vulnerability metrics, it permits the computation of a priority score, thus, guiding future studies towards the roles that appear to be the most at risk to harm the blockchain system if they were to be corrupted.

In chapter 5, we have seen that existing blockchain simulators are neither role-based nor compatible with AI, and specifically MARL. We thus argued on the need of a new tool, explicitly integrating the concepts of role and behavior, so that it can be used for specific incentive based studies. So, we have developed a new role-based, modular simulator, whose basic building blocks are the *groups*, *roles* and *behaviors* of AGR4BS. Chapter 5 thoroughly detailed the characteristics of the simulator, which core ones are: *Modularity* through a group, role and behavior structure. *Reproducibility* obtained by a virtualized time and networking management. *MARL / AI compatibility* by using python and allowing the developers to modify any role and replace one or several behaviors by AI models of their choice.

Moreover, we have followed an open source approach and made the simulator available at https://github.com/hroussille/agr4bs.

This simulator already supports PoW blockchains such as Bitcoin and Ethereum 1 as well as newer PoS chains like Ethereum 2.0.

We have finally brought the previous contributions together in chapter 6 with two simulation focused on Ethereum 2.0. Those experiments uses MARL to explore the potential of a vulnerability involving selfish block creation in Ethereum 2.0.

Agents were trained through MARL to optimize an economical objective didn't converge to the optimal strategy and never outperformed the honest ones. By changing the objective criterion to incentivize the agents to create forks, we did obtain a positive result showing increased protocol rewards for the malicious agents involved in selfish block creation. This has shown the usability of MARL within our framework, but more generally, the use of AI in the context of blockchain vulnerability exploration and analysis.

7.2 Limits

We identified three main limits in the work we have done during this thesis.

Firstly, the metrics used the compute the priority scores in Chapter 4 are informal and subjective. The impact of a vulnerability exploitation can have reputational consequences, which, in turn, leads to economical and therefore, security implications. Such an impact is difficult to qualify and quantify, which could lead to invalid ranking, thus defeating the very purpose of the priority score.

Secondly, the use of MARL does allow for studies of highly complex environments but this complexity may lead to the inability for the agents to learn anything from the environment due to complex interactions that violate some of the base assumptions of reinforcement learning such as stationarity. Still, the results are not as explainable nor objective as what could be achieved with game theory. Complex environments must be simplified to some extent to permit any study which ultimately induces a bias in the experiments. The definition of the Agent's actions and observations inherently limit the agent's capabilities. Some strategies become out of reach simply because the required observations are missing, similarly, the actions of the agents cannot be fully free as it would make the search space intractable. The environment itself may include bugs or faults that can bias the whole experiment and significantly influence the outcome. Furthermore, as MARL are essentially a form of statistical learning, meaning that an exhaustive search is impossible, therefore a negative results is not a guarantee of a protocol's safety.

Finally, our work can be used to evaluate and monitor the security of various protocols, but we insist on the fact that it does not directly proposes solution to mitigate any vulnerability: it is an automatic vulnerability detection framework.

7.3 Perspectives

Following this work, and as mentioned in Section 7.2, the taxonomy presented in Chapter 4 should be refined to compute the priority scores based on objective metrics for both risk and vulnerability.

In order to broaden the scope of applications of this thesis, more well known protocols should be developed and added to the base blockchain model library of the simulator such as Tendermint and Solana.

Similarly, we believe that experimenting with other automatic approaches such as genetic algorithms and comparing the results with MARL based ones is interesting, it could help the researchers select the best tools given a specific scenario of interest.

As the blockchain ecosystem is now mostly focused on performance and throughput optimizations, for instance using sharding, the simulator should be extended to support non monolithic blockchains to stay relevant in the long term.

This would allow the framework to be used as a continuous monitoring security system before, during and after the upgrade processes of fast moving blockchains, the main candidate for this being Ethereum 2.0.

Blockchain technology continues to evolve and is set to become an integral part of global financial and digital infrastructure. Ensuring that such systems are secure is a top priority. Our work leverages MARL and introduces a novel, adaptive way of identifying vulnerabilities in these decentralized systems. As the complexity of blockchains increases, the synergy with AI and blockchain will be paramount to assess their security.

Bibliography

Abbas, Hosny Ahmed (Apr. 2015). "Realizing the NOSHAPE MAS Organizational Model: An Operational View". In: *Int. J. Agent Technol. Syst.* 7.2, 75–104. ISSN: 1943-0744.

Abbott, Russell J. (Nov. 1983). "Program Design by Informal English Descriptions". In: *Commun. ACM* 26.11, 882–894. ISSN: 0001-0782.

Albshri, Adel et al. (July 2022). "Blockchain Simulators: A Systematic Mapping Study". In: pp. 284–294. DOI: 10.1109/SCC55611.2022.00049.

Alharby, Maher and Aad van Moorsel (2019). "BlockSim: A Simulation Framework for Blockchain Systems". In: *SIGMETRICS Perform. Eval. Rev.* 46.3, 135–138. ISSN: 0163-5999. DOI: 10.1145/3308897.3308956. URL: https://doi.org/10.1145/3308897.3308956.

— (2020). "BlockSim: An Extensible Simulation Tool for Blockchain Systems". In: *Frontiers in Blockchain* 3, p. 28. ISSN: 2624-7852.

Alkhalifah, Ayman et al. (Aug. 2020). "A Taxonomy of Blockchain Threats and Vulnerabilities". English. In: *Blockchain for Cybersecurity and Privacy*. 1st ed. United States: CRC Press, pp. 3–25. ISBN: 9780367343101. DOI: 10.1201/9780429324932.

Alpern, Bowen and Fred B. Schneider (1985). "Defining liveness". In: *Information Processing Letters* 21.4, pp. 181–185. ISSN: 0020-0190.

Amoussou-Guenou, Yackolley et al. (2020). "Rational vs byzantine players in consensus-based blockchains". In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS* 2020-May (May), pp. 43–51. ISSN: 15582914.

Anceaume, Emmanuelle et al. (2019). "Blockchain Abstract Data Type". In: *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*. SPAA '19. Phoenix, AZ, USA: Association for Computing Machinery, 349–358. ISBN: 9781450361842.

Androulaki, Elli et al. (2018). "Hyperledger fabric: a distributed operating system for permissioned blockchains". In: *Proceedings of the thirteenth EuroSys conference*, pp. 1–15.

Apostolaki, M., A. Zohar, and L. Vanbever (2017). "Hijacking Bitcoin: Routing Attacks on Cryptocurrencies". In: *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 375–392.

Babulak, Eduard and Ming Wang (Jan. 2008). "Discrete Event Simulation: State of the Art". In: *International Journal of Online Engineering (iJOE)* 4, pp. 60–63. DOI: 10.5772/9894.

Badertscher, Christian et al. (2020). *Consensus Redux: Distributed Ledgers in the Face of Adversarial Supremacy*. Cryptology ePrint Archive, Paper 2020/1021. https://eprint.iacr.org/2020/1021. URL: https://eprint.iacr.org/2020/1021.

Bano, Shehar et al. (2019). "SoK: Consensus in the Age of Blockchains". In: *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. AFT '19. Zurich, Switzerland: Association for Computing Machinery, 183–198. ISBN: 9781450367325.

Bellman, Richard (1957). "A Markovian decision process". In: *Journal of mathematics and mechanics*, pp. 679–684.

Blais, Marc-Andrė and Moulay A Akhloufi (2023). "Reinforcement learning for swarm robotics: An overview of applications, algorithms and simulators". In: *Cognitive Robotics*.

Bonneau, Joseph (2016). "Why Buy When You Can Rent? - Bribery Attacks on Bitcoin-Style Consensus". In: *Financial Cryptography Workshops*. URL: https://api.semanticscholar.org/CorpusID:18122687.

Brewer, Eric (July 2000). "Towards robust distributed systems". In: p. 7. DOI: 10.1145/343477.343502.

Brockman, Greg et al. (2016). *OpenAI Gym*. arXiv: 1606.01540 [cs.LG].

Buchman, Ethan, Jae Kwon, and Zarko Milosevic (2018). "The latest gossip on BFT consensus". In: *CoRR* abs/1807.04938. Tendermint. arXiv: 1807.04938.

Busoniu, Lucian, Robert Babuska, and Bart De Schutter (2008). "A comprehensive survey of multiagent reinforcement learning". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2, pp. 156–172.

Buterin, Vitalik (2014). "A next-generation smart contract and decentralized application platform". In: *White Paper* January, pp. 1–36. URL: http://buyxpr.com/build/pdfs/EthereumWhitePaper.pdf.

Buterin, Vitalik et al. (2020). "Combining GHOST and Casper". In: *CoRR* abs/2003.03052. arXiv: 2003.03052. URL: https://arxiv.org/abs/2003.03052.

Cachin, Christian, Rachid Guerraoui, and Lus Rodrigues (2011). *Introduction to Reliable and Secure Distributed Programming*. 2nd. Springer Publishing Company, Incorporated. ISBN: 3642152597, 9783642152597.

Cai, Wei et al. (2018). "Decentralized Applications: The Blockchain-Empowered Software System". In: *IEEE Access* 6, pp. 53019–53033.

Caldarelli, Giulio (2020). "Understanding the blockchain oracle problem: A call for action". In: *Information (Switzerland)* 11.11, pp. 1–19. ISSN: 20782489.

Carlsten, Miles et al. (2016). "On the instability of bitcoin without the block reward". In: *Proc. of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, pp. 154–167.

Castro, Miguel and Barbara Liskov (1999). "Practical Byzantine Fault Tolerance". In: *Proceedings of the Third Symposium on Operating Systems Design and Implementation*. OSDI '99. New Orleans, Louisiana, USA: USENIX Association, 173–186. ISBN: 1880446391.

Chen, Dong et al. (2021). "Powernet: Multi-agent deep reinforcement learning for scalable powergrid control". In: *IEEE Transactions on Power Systems* 37.2, pp. 1007–1017.

Chen, Weiya, Kunlin Zhou, and Chunxiao Chen (2016). "Real-time bus holding control on a transit corridor based on multi-agent reinforcement learning". In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, pp. 100–106.

Chohan, Usman W. (2018). "The Double Spending Problem and Cryptocurrencies". In: *SSRN Electronic Journal* n/a.n/a, 11p. ISSN: 1556-5068. DOI: 10.2139/ssrn.3090174.

Ciatto, Giovanni et al. (Jan. 2020a). "Blockchain-Based Coordination: Assessing the Expressive Power of Smart Contracts". In: *Information* 11 (1), p. 52. ISSN: 2078-2489.

Ciatto, Giovanni et al. (2020b). "From agents to blockchain: Stairway to integration". In: *Applied Sciences (Switzerland)* 10 (21), pp. 1–22. ISSN: 20763417.

Criado, N., E. Argente, and V. Botti (2013). "THOMAS: An agent platform for supporting normative multi-agent systems". In: *Journal of Logic and Computation* 23.2, pp. 309–333.

Daian, Philip et al. (2020). "Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability". In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 910–927. DOI: 10.1109/SP40000.2020.00040.

Damadam, Shima et al. (2022). "An intelligent IoT based traffic light management system: deep reinforcement learning". In: *Smart Cities* 5.4, pp. 1293–1311.

Decker, C., J. Seidel, and R. Wattenhofer (2016). "Bitcoin Meets Strong Consistency". In: *Proceedings of the 17th International Conference on Distributed Computing and Networking Conference (ICDCN)*. PeerCensus.

Dignum, Virginia, Javier Vázquez-Salceda, and Frank Dignum (2005). "OMNI: Introducing Social Structure, Norms and Ontologies into Agent Organizations". In: *Programming Multi-Agent Systems*. Ed. by Rafael H. Bordini et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 181–198. ISBN: 978-3-540-32260-3.

El Faqir, Youssef, Javier Arroyo, and Samer Hassan (2020). "An Overview of Decentralized Autonomous Organizations on the Blockchain". In: *Proceedings of the 16th International Symposium on Open Collaboration*. OpenSym 2020. Virtual conference, Spain: Association for Computing Machinery. ISBN: 9781450387798.

Ersoy, Oğuzhan et al. (2018). "Transaction Propagation on Permissionless Blockchains: Incentive and Routing Mechanisms". In: *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. IEEE, pp. 20–30. DOI: 10.1109/CVCBT.2018.00008.

Eskandari, Shayan, Seyedehmahsa Moosavi, and Jeremy Clark (2020b). "SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain". In: *Financial Cryptography and Data Security*. Ed. by Andrea Bracciali et al. Cham: Springer Inter. Publishing, pp. 170–189. ISBN: 978-3-030-43725-1.

— (2020a). *SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain*. Vol. 11599 LNCS. Springer International Publishing, pp. 170–189. ISBN: 9783030437244. arXiv: 1902.05164.

Eyal, Ittay and Emin Gün Sirer (2014). "Majority is not enough: Bitcoin mining is vulnerable". In: *International Conference on Financial Cryptography and Data Security*. Springer, pp. 436–454.

Ferber, Jacques (1999). *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison-Wesley Reading.

Ferber, Jacques, Olivier Gutknecht, and Fabien Michel (2004). "From Agents to Organizations: An Organizational View of Multi-agent Systems". In: *Agent-Oriented Software Engineering IV*. Ed. by Paolo Giorgini, Jörg P. Müller, and James Odell. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 214–230. ISBN: 978-3-540-24620-6.

Filho, Jugurta Lisboa and José Luis Braga (2017). "UML: Unified Modeling Language". In: *Encyclopedia of GIS*. Ed. by Shashi Shekhar, Hui Xiong, and Xun Zhou. Cham: Springer International Publishing, pp. 2345–2346.

Foerster, Jakob et al. (2016). "Learning to communicate with deep multi-agent reinforcement learning". In: *Advances in neural information processing systems* 29.

Foundation Lindsay X. Lin, Legal Counsel at Interstellar and Stellar Development (Jan. 5, 2019). "Deconstructing Decentralized Exchanges". In: *Stanford Journal of Blockchain Law & Policy*. https://stanford-jblp.pubpub.org/pub/deconstructing-dex.

Gao, Shang et al. (2019). "Power Adjusting and Bribery Racing: Novel Mining Attacks in the Bitcoin System". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: Association for Computing Machinery, 833–850. ISBN: 9781450367479. DOI: 10.1145/3319535.3354203. URL: https://doi.org/10.1145/3319535.3354203.

Garay, Juan, Aggelos Kiayias, and Nikos Leonardos (2015). "The Bitcoin Backbone Protocol: Analysis and Applications". In: *Advances in Cryptology - EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Berlin, Heidelberg: Springer Berlin Heid., pp. 281–310. ISBN: 978-3-662-46803-6.

Giannoccaro, Ilaria and Pierpaolo Pontrandolfo (2002). "Inventory management in supply chains: a reinforcement learning approach". In: *International Journal of Production Economics* 78.2, pp. 153–161. ISSN: 0925-5273. DOI: https://doi.org/10.1016/S0925-5273(00)00156-0. URL: https://www.sciencedirect.com/science/article/pii/S0925527300001560.

Gilad, Yossi et al. (2017). "Algorand: Scaling byzantine agreements for cryptocurrencies". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, pp. 51–68.

Giorgini, Paolo, Manuel Kolp, and John Mylopoulos (2006). "Multi-Agent Architectures as Organizational Structures". In: *Autonomous Agents and Multi-Agent Systems* 13, pp. 3–25.

Goodman, LM (2014). "Tezos—a self-amending crypto-ledger White paper". In: *URL: https://www. tezos. com/static/papers/white paper. pdf*.

Grondman, Ivo et al. (2012). "A survey of actor-critic reinforcement learning: Standard and natural policy gradients". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6, pp. 1291–1307.

Gürcan, Önder (2020). "On Using Agent-based Modeling and Simulation for Studying Blockchain Systems". In: *JFMS 2020 - Les Journées Francophones de la Modélisation et de la Simulation - Convergences entre la Théorie de la Modélisation et la Simulation et les Systèmes Multi-Agents*. Cargèse, France. ISBN: 9782364937574.

Gürcan, Önder (2024). "Multi-Agent eXperimenter (MAX)". In: *arXiv preprint arXiv:2404.08398*.

Gürcan, Önder, Antonella Del Pozzo, and Sara Tucci-Piergiovanni (2017). "On the Bitcoin Limitations to Deliver Fairness to Users". In: *On the Move to Meaningful Internet Systems. OTM 2017 Conferences*. Ed. by H. Panetto et al. Cham: Springer International Publishing, pp. 589–606.

Gutknecht, Olivier and Jacques Ferber (July 2000). "The MadKit Agent Platform Architecture". In: vol. 1887. ISBN: 978-3-540-42315-7. DOI: 10.1007/3-540-47772-1_5.

Gürcan, Önder (2019). "Multi-Agent Modelling of Fairness for Users and Miners in Blockchains". In: *Highlights of Practical Applications of Survivable Agents and Multi-Agent Systems. The PAAMS Collection*. Ed. by F. De La Prieta et al. Cham: Springer International Publishing, pp. 92–99. ISBN: 978-3-030-24299-2.

Hameed, Khizar et al. (2022). "A taxonomy study on securing Blockchain-based Industrial applications: An overview, application perspectives, requirements, attacks, countermeasures, and open issues". In: *J Ind Inf Integr* 26, p. 100312. ISSN: 2452-414X. DOI: https://doi.org/10.1016/j.jii.2021.100312.

Herlihy, Maurice (2018). "Atomic Cross-Chain Swaps". In: *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*. PODC '18. Egham, United Kingdom, pp. 245–254.

Herlihy, Maurice and Sergio Rajsbaum (1995). "Algebraic topology and distributed computing a primer". In: *Computer Science Today: Recent Trends and Developments*. Ed. by Jan van Leeuwen. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 203–217. ISBN: 978-3-540-49435-5.

Hou, Charlie et al. (2021). "SquirRL: Automating Attack Discovery on Blockchain Incentive Mechanisms with Deep Reinforcement Learning". In: *Network and Distrib. Syst. Security Sympos. (NDSS)*.

Howard, Ronald A (1960). "Dynamic programming and markov processes." In.

Hübner, Jomi Fred, Jaime Simão Sichman, and Olivier Boissier (2002). "MOISE+: Towards a Structural, Functional, and Deontic Model for MAS Organization". In: *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*. AAMAS '02. Bologna, Italy: Association for Computing Machinery, 501–502. ISBN: 1581134800.

Ishii, Shin, Wako Yoshida, and Junichiro Yoshimoto (2002). "Control of exploitation–exploration meta-parameter in reinforcement learning". In: *Neural networks* 15.4-6, pp. 665–687.

Kwon, Jae (2014). "TenderMint : Consensus without Mining". In: *the-Blockchain.Com* 6, pp. 1–10. URL: tendermint.com/docs/tendermint.pdf.

Kwon, Jae and Ethan Buchman (2016). "Cosmos : A Network of Distributed Ledgers". In: *White Paper*.

Lambora, Annu, Kunal Gupta, and Kriti Chopra (2019). "Genetic algorithm-A literature review". In: *2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon)*. IEEE, pp. 380–384.

Lamport, Leslie (2001). "Paxos Made Simple". In: URL: https://api.semanticscholar.org/CorpusID:1936192.

Larman, C. (2004). *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development (3rd edition)*. Upper Saddle River, NJ, USA: Prentice Hall PTR. ISBN: 0131489062.

Li, Wenting et al. (2017). "Securing proof-of-stake blockchain protocols". In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology: ESORICS 2017 International Workshops, DPM 2017 and CBT 2017, Oslo, Norway, September 14-15, 2017, Proceedings*. Springer, pp. 297–315.

Li, Xiaoqi et al. (2020). "A survey on the security of blockchain systems". In: *Future Generation Computer Systems* 107, pp. 841–853. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2017.08.020. URL: https://www.sciencedirect.com/science/article/pii/S0167739X17318332.

Liang, Xiaodan et al. (2018). "Cirl: Controllable imitative reinforcement learning for vision-based self-driving". In: *Proceedings of the European conference on computer vision (ECCV)*, pp. 584–599.

Lillicrap, T. et al. (2016). "Continuous control with deep reinforcement learning". In: *CoRR* abs/1509.02971.

Liu, Zhihan et al. (2022). "Welfare maximization in competitive equilibrium: Reinforcement learning for markov exchange economy". In: *International Conference on Machine Learning*. PMLR, pp. 13870–13911.

Lo, Sin Kuang et al. (2020). "Reliability analysis for blockchain oracles". In: *Computers and Electrical Engineering* 83, pp. 1–10. ISSN: 00457906.

Lowe, Ryan et al. (2017). "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., pp. 6379–6390. URL: https://proceedings.neurips.cc/paper/2017/file/68a9750337a418a86fe06c1991a1d64c-Paper.pdf.

Luu, Loi et al. (2015). "Demystifying incentives in the consensus computer". In: *Proceedings of the ACM Conference on Computer and Communications Security* 2015-Octob, pp. 706–719. ISSN: 15437221.

Luu, Loi et al. (2017). “SmartPool: Practical decentralized pooled mining”. In: *Proceedings of the 26th USENIX Security Symposium*, pp. 1409–1426. ISBN: 9781931971409.

Lyu, Xueguang et al. (2021). “Contrasting centralized and decentralized critics in multi-agent reinforcement learning”. In: *arXiv preprint arXiv:2102.04402*.

Malcolm, Grant (2009). “Sheaves, Objects, and Distributed Systems”. In: *Electronic Notes in Theoretical Computer Science* 225. Proceedings of the Irish Conference on the Mathematical Foundations of Computer Science and Information Technology (MFCSIT 2006), pp. 3–19. ISSN: 1571-0661.

Marchesi, Lodovica, Michele Marchesi, and Roberto Tonelli (2020). “ABCDE –agile block chain DApp engineering”. In: *Blockchain: Research and Applications* 1.1, p. 100002. ISSN: 2096-7209.

Matignon, Laetitia, Guillaume J. Laurent, and Nadine Le Fort-Piat (2012). “Independent reinforcement learners in cooperative Markov games: a survey regarding coordination problems”. In: *The Knowledge Engineering Review* 27.1, 1–31. DOI: 10.1017/S0269888912000057.

Meldman-Floch, Wyatt (2018). “Blockchain Cohomology”. In: *CoRR* abs/1805.07047. arXiv: 1805.07047.

Meynkhard, Artur (2019). “Fair market value of bitcoin: Halving effect”. In: *Investment Management & Financial Innovations* 16.4, p. 72.

Miller, Andrew K. et al. (2016). “The Honey Badger of BFT Protocols”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. URL: https://api.semanticscholar.org/CorpusID:9679805.

Mirkin, Michael et al. (2020). “BDoS: Blockchain Denial-of-Service”. In: *2020 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, pp. 601–619.

Mnih, Volodymyr et al. (2013). “Playing Atari with Deep Reinforcement Learning”. In: *CoRR* abs/1312.5602. arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

Nakamoto, Satoshi (2008). “Bitcoin: A peer-to-peer electronic cash system”. In: *Decentralized Business Review*, p. 21260.

Nayak, Kartik et al. (2016). “Stubborn mining: Generalizing selfish mining and combining with an eclipse attack”. In: *Proceedings - 2016 IEEE European Symposium on Security and Privacy, EURO S and P 2016*, pp. 305–320.

Neu, Joachim, Ertem Nusret Tas, and David Tse (2021). *Ebb-and-Flow Protocols: A Resolution of the Availability-Finality Dilemma*. arXiv: 2009.04987 [cs.CR]. URL: https://arxiv.org/abs/2009.04987.

Neuder, Michael et al. (2020). *Selfish Behavior in the Tezos Proof-of-Stake Protocol*. arXiv: 1912.02954 [cs.CR].

— (2021). *Low-cost attacks on Ethereum 2.0 by sub-1/3 stakeholders*. arXiv: 2102.02247 [cs.CR]. URL: https://arxiv.org/abs/2102.02247.

Nowak, Thomas (2010). “Topology in Distributed Computing”. MA thesis. Austria: Vienna University of Technology.

Ongaro, Diego and John Ousterhout (2014). “In Search of an Understandable Consensus Algorithm”. In: *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*. USENIX ATC’14. Philadelphia, PA: USENIX Association, 305–320. ISBN: 9781931971102.

OpenAI et al. (2019). *Dota 2 with Large Scale Deep Reinforcement Learning*. arXiv: 1912.06680 [cs.LG].

Papoudakis, Georgios et al. (2019). “Dealing with non-stationarity in multi-agent deep reinforcement learning”. In: *arXiv preprint arXiv:1906.04737*.

Pavloff, Ulysse, Yackolley Amoussou-Guenou, and Sara Tucci-Piergiovanni (2023). "Ethereum Proof-of-Stake under Scrutiny". In: *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*. SAC '23. Tallinn, Estonia: Association for Computing Machinery, 212–221. ISBN: 9781450395175. DOI: 10.1145/3555776.3577655. URL: https://doi.org/10.1145/3555776.3577655.

Peterson, Matthew, Todd Andel, and Ryan Benton (2022). "Towards detection of selfish mining using machine learning". In: *International Conference on Cyber Warfare and Security*. Vol. 17. 1, pp. 237–243.

Piriou, Pierre-Yves et al. (2021). "Justifying the Dependability and Security of Business-Critical Blockchain-based Applications". In: *2021 Third Inter. Conf. on Blockchain Computing and Applications (BCCA)*. IEEE, pp. 97–104. DOI: 10.1109/BCCA53669.2021.9656962.

Rocha, H. and S. Ducasse (2018). "Preliminary Steps Towards Modeling Blockchain Oriented Software". In: *2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pp. 52–57.

Rodriguez, Sebastian, John Thangarajah, and Michael Winikoff (2021). "User and System Stories: An Agile Approach for Managing Requirements in AOSE". In: *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems* i, pp. 1064–1072.

Romiti, Matteo et al. (2019). "A deep dive into Bitcoin mining pools: An empirical analysis of mining shares". In: *arXiv*, pp. 1–19. ISSN: 23318422. arXiv: 1905.05999.

Roussille, Hector, Önder Gürcan, and Fabien Michel (2022). "AGR4BS: A Generic Multi-Agent Organizational Model for Blockchain Systems". In: *Big Data and Cognitive Computing* 6.1, 41p. ISSN: 2504-2289. DOI: 10.3390/bdcc6010001. URL: https://www.mdpi.com/2504-2289/6/1/1.

Roussille, Hector, Önder Gürcan, and Fabien Michel (2023). "A Taxonomy of Blockchain Incentive Vulnerabilities for Networked Intelligent Systems". In: *IEEE Communications Magazine* 61.8, pp. 108–114. DOI: 10.1109/MCOM.005.2200904.

Rummery, Gavin A and Mahesan Niranjan (1994). *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK.

Saad, Muhammad et al. (2020). "Exploring the Attack Surface of Blockchain: A Comprehensive Survey". In: *IEEE Communications Surveys and Tutorials* 22.3, pp. 1977–2008. ISSN: 1553877X. DOI: 10.1109/COMST.2020.2975999.

Sagar, P. V. and M. P. K. Kishore (2019). "Sheaf Representation of an Information 664 System". In: *International Journal of Rough Sets and Data Analysis (IJRSDA)* 6 (2), pp. 78–83.

Saks, Michael and Fotios Zaharoglou (1993). "Wait-Free <i>k</i>-Set Agreement is Impossible: The Topology of Public Knowledge". In: *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*. STOC '93. San Diego, California, USA: Association for Computing Machinery, 101–110. ISBN: 0897915917.

Sapirshtein, Ayelet, Yonatan Sompolinsky, and Aviv Zohar (2016). "Optimal selfish mining strategies in bitcoin". In: *International Conference on Financial Cryptography and Data Security*. Springer, pp. 515–532.

Sayeed, Sarwar, Hector Marco-Gisbert, and Tom Caira (2020). "Smart Contract: Attacks and Protections". In: *IEEE Access* 8, pp. 24416–24427. ISSN: 21693536. DOI: 10.1109/ACCESS.2020.2970495.

Schwarz-Schilling, Caspar et al. (2021). "Three Attacks on Proof-of-Stake Ethereum". In: *IACR Cryptol. ePrint Arch.* 2021, p. 1413. URL: https://api.semanticscholar.org/CorpusID:239024587.

Silver, David et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529. Article, 484 EP –. URL: http://dx.doi.org/10.1038/nature16961.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Szabo, Nick (1997). "Smart Contracts: Formalizing and Securing Relationships on Public Networks". In: *First Monday* 2.9. ISSN: 1396-0466.

Taylor, Michael Bedford (2017). "The evolution of bitcoin hardware". In: *Computer* 50.9, pp. 58–66. ISSN: 00189162.

Toroghi Haghighat, Alireza and Mehdi Shajari (Mar. 2019). "Block withholding game among bitcoin mining pools". In: *Future Generation Computer Systems* 97.

Tschorsch, Florian and Björn Scheuermann (2016). "Bitcoin and beyond: A technical survey on decentralized digital currencies". In: *IEEE Communications Surveys and Tutorials* 18.3, pp. 2084–2123. ISSN: 1553877X.

Wang, Taotao, Soung Chang Liew, and Shengli Zhang (2021a). "When blockchain meets AI: Optimal mining strategy achieved by machine learning". In: *International Journal of Intelligent Systems* 36.5, pp. 2183–2207.

Wang, Wenbo et al. (2019). "A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks". In: *IEEE Access* 7, pp. 22328–22370.

Wang, Zhaojie et al. (2021b). "BSMRL: Bribery Selfish Mining with Reinforcement Learning". In: *Data Mining and Big Data: 6th International Conference, DMBD 2021, Guangzhou, China, October 20–22, 2021, Proceedings, Part I 6*. Springer, pp. 1–10.

Wang, Zhaojie et al. (2021c). "ForkDec: accurate detection for selfish mining attacks". In: *Security and Communication Networks* 2021, pp. 1–8.

Watkins, Christopher JCH and Peter Dayan (1992). "Q-learning". In: *Machine learning* 8, pp. 279–292.

Werner, Sam M. et al. (2021). *SoK: Decentralized Finance (DeFi)*. arXiv: 2101.08778 [cs.CR].

Wolfram, D. and J. Goguen (1991). "A Sheaf Semantics for FOOPS Expressions". In: *Object-Based Concurrent Computing*.

Wood, Gavin (2014). *Ethereum: A secure decentralised generalised transaction ledger*. http://bitcoinaffiliatelist.com/wp-content/uploads/ethereum.pdf. Accessed: 2016-08-22.

Wooldridge, Michael (2009). *An introduction to multiagent systems*. John wiley & sons.

Yajam, Habib, Elnaz Ebadi, and Mohammad Ali Akhaee (2023). "JABS: A Blockchain Simulator for Researching Consensus Algorithms". In: *IEEE Transactions on Network Science and Engineering*, pp. 1–12. DOI: 10.1109/TNSE.2023.3282916.

Yang, Guoyu et al. (2020). "IPBSM: an optimal bribery selfish mining in the presence of intelligent and pure attackers". In: *International Journal of Intelligent Systems* 35.11, pp. 1735–1748.

Zander, Manuel, Tom Waite, and Dominik Harz (2019). "DAGsim: Simulation of DAG-Based Distributed Ledger Protocols". In: *SIGMETRICS Perform. Eval. Rev.* 46.3, 118–121. ISSN: 0163-5999. DOI: 10.1145/3308897.3308951. URL: https://doi.org/10.1145/3308897.3308951.

Zhang, Jianting et al. (Aug. 2020). "SkyChain: A Deep Reinforcement Learning-Empowered Dynamic Blockchain Sharding System". In: pp. 1–11.

Zhang, Kaiqing, Zhuoran Yang, and Tamer Başar (2021). "Multi-agent reinforcement learning: A selective overview of theories and algorithms". In: *Handbook of reinforcement learning and control*, pp. 321–384.

Zhang, Lifeng et al. (May 2019). "A Game-theoretic Method based on Q-Learning to Invalidate Criminal Smart Contracts". In: *Information Sciences* 498.

Zhou, Ming et al. (2021). "Smarts: An open-source scalable multi-agent rl training school for autonomous driving". In: *Conference on robot learning*. PMLR, pp. 264–285.