Mitigating the impact of malicious nodes in distributed ledger networks with resource constrained nodes

Benčić, Federico Matteo

Sveučilište u Zagrebu

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Federico Matteo Benčić

MITIGATING THE IMPACT OF MALICIOUS NODES IN DISTRIBUTED LEDGER NETWORKS WITH RESOURCE CONSTRAINED NODES

DOCTORAL THESIS

Zagreb, 2023.

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Federico Matteo Benčić

MITIGATING THE IMPACT OF MALICIOUS NODES IN DISTRIBUTED LEDGER NETWORKS WITH RESOURCE CONSTRAINED NODES

DOCTORAL THESIS

Supervisor: Professor Ivana Podnar Žarko, PhD

Zagreb, 2023.

Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Federico Matteo Benčić

UBLAŽAVANJE UTJECAJA ZLONAMJERNIH ČVOROVA U MREŽAMA RASPODIJELJENIH GLAVNIH KNJIGA S ČVOROVIMA OGRANIČENIH RESURSA

DOKTORSKI RAD

Mentor: Prof. dr. sc. Ivana Podnar Žarko

Zagreb, 2023.

The doctoral thesis was completed at the University of Zagreb Faculty of Electrical Engineering and Computing, Department of Telecommunications.

Mentor: Ivana Podnar Žarko, PhD

The thesis has: 121 pages.

Thesis number: _____

About the Supervisor

Ivana Podnar Žarko is Full Professor at the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia (FER) where she teaches courses on distributed systems and Internet of Things. She received her B.Sc., M.Sc. and Ph.D. degrees in electrical engineering from FER, in 1996, 1999 and 2004, respectively.

She is affiliated with the Department of Telecommunications at FER from 1997. She was a guest researcher and research associate at the Technical University of Vienna, Austria, and a postdoctoral researcher at the Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland. She was promoted to Full Professor in December 2017 and currently leads FER's Internet of Things Laboratory, and from 01.07.2019 to 09.03.2022 she held the position of the Head of FER Research Support Centre.

Prof. Žarko has participated in 9 research projects in the last 5 years funded by national and EU funds and was the Technical Manager of the H2020 project "symbIoTe: Symbiosis of smart objects across IoT environments" (2016-2018). She is currently leading two national projects: IoT-Field, funded by the European Structural and Investment Funds, and IoT4us, funded by the Croatian Science Foundation. She is also participating as a researcher in the Centre of Research Excellence for Data Science and Advanced Cooperative Systems, which is the first national center of excellence in the field of technical sciences in Croatia.

She has co-authored more than 80 scientific journal and conference papers in the area of large-scale distributed systems, IoT, and Big data processing. Recently she focuses on research problems related to IoT interoperability and Distributed Ledger Technology (DLT). She has served as a program committee member for many international conferences and workshops (e.g., IEEE Globecom, IEEE 5G World Forum, Global IoT Summit, IEEE ICC) and is co-organizing a series of research workshops Int. Workshop on Interoperability and Open Source Solutions for the IoT (InterOSS-IoT) from 2014. She was a track chair of the 19th Annual IEEE/ACM Int. Symposium in Cluster, Cloud, and Grid Computing (CCGrid 2019) and member of the editorial board of Automatika: Journal for Control, Measurement, Electronics, Computing and Communications since 2016.

Prof. Ivana Podnar Žarko is a member of IEEE and was the Chapter Chair of IEEE Communications Society, Croatia Chapter (2011—2014). She has received an award for engineering excellence from the IEEE Croatia Section in 2013 and the FER Science Award in 2020 for outstanding achievements in research and innovation in the last 5 years.

O mentoru

Ivana Podnar Žarko je redoviti profesor na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva gdje predaje kolegije u području raspodijeljenih sustava i Interneta stvari. Diplomirala je, magistrirala i doktorirala u polju elektrotehnike na FER-u, 1996., 1999. odnosno 2004. godine.

Zaposlena je na Zavodu za telekomunikacije FER-a od 1997. godine. Usavršavala se na Tehničkom sveučilištu u Beču, Austrija i Ecole Polytechnique Fédérale de Lausanne (EPFL), Švicarska. U prosincu 2017. godine izabrana je u zvanje redovitog profesora. Voditelj je FER-ovog Laboratorija za Internet stvari, a od 1.7.2019. do 9.3.2022. obnašala je funkciju voditelja Centra za potporu istraživanju FER-a.

Sudjelovala je u nizu istraživačkih projekata financiranih iz domaćih izvora i fondova EU te je bila tehnički koordinator H2020 projekta "symbIoTe: Symbiosis of smart objects across IoT environments" (2016.-2018.). Trenutno vodi dva nacionalna projekta: IoT-polje, projekt financiran iz Europskih strukturnih i investicijskih fondova i IoT4us, projekt financiran od strane Hrvatske zaklade za znanost. Sudjeluje kao istraživač u radu Znanstvenog centra izvrsnosti za znanost o podatcima i kooperativne sustave koji je prvi je nacionalni centar izvrsnosti u području tehničkih znanosti u Hrvatskoj.

Autor je više od 80 znanstvenih radova u području raspodijeljenih sustava, IoT-a i obrade velikih skupova podataka. Nedavno se usredotočila na istraživačke probleme povezane s interoperabilnošću za IoT i tehnologijama vezanim uz raspodijeljene digitalne knjige (Digital Ledger Technologies, DLT). Bila je član programskog odbora na mnogim međunarodne znanstvenim skupovima i radionicama (npr. IEEE Globecom, IEEE 5G World Forum, Global IoT Summit, IEEE ICC) te suorganizator niza istraživačkih radionica pod nazivom Int. Workshop on Interoperability and Open Source Solutions for the IoT (InterOSS-IoT) od 2014. Bila je član organizacijskog odbora skupa 19th Annual IEEE/ACM Int. Symposium in Cluster, Cloud, and Grid Computing (CCGrid 2019) te je urednik u znanstvenom časopisu Automatika: Journal for Control, Measurement, Electronics, Computing and Communications od 2016. godine.

Prof. dr. sc. Ivana Podnar Žarko je član strukovnih udruga IEEE, IEEE Communications Society i IEEE Computer Society, a od 2011. do 2014. godine je bila predsjednica Odjela za komunikacije Hrvatske sekcije IEEE. Dobila je nagradu za inženjersku izvrsnost Hrvatska sekcije IEEE 2013. godine i nagradu za znanost FER-a u 2020. godini za izvanredna postignuća u istraživanju i inovacijama u posljednjih 5 godina.

Preface

I would like to express my sincere gratitude to my supervisor, Professor Ivana Podnar Žarko, PhD, at the Faculty of Electrical Engineering and Computing, University of Zagreb, for her continuous support during my work on this thesis, as well as for her help and contribution to the scientific work contained in this thesis. Through her guidance, I was able to discover my passion for research and the written word.

I would also like to express my gratitude to Professor Krešimir Pripužić, PhD, and Pavle Skočir, PhD, for their time and assistance extended to me during the creation of this thesis. To Denis Salopek, PhD, I extend my sincerest appreciation for his exceptional knowledge and expertise in open source and Linux.

I would like to thank my family and friends for their essential support and encouragement during my PhD studies and years of education. I would like to thank my girlfriend Andrea for taking care of me more than I took care for myself during my PhD journey.

Finally, I would like to acknowledge my sister Aurora. I am excited to see where her own journey will take her, and wish her the best of luck in the years to come.

Abstract

The immutable nature of Distributed Ledger Technology leads to continuous growth of the ledger over time. A Distributed Ledger node takes a non-negligible amount of time to synchronize with the network and consumes a lot of resources when running on current consumer hardware. New nodes, i.e., nodes that have never synchronized with the network before, require significant computational power and memory to download the ledger and verify all entries since the ledger was created. For this reason, resource constrained devices cannot participate in ledger maintenance without affecting important features of the technology, namely immunity to censorship, immutability and the elimination of a trusted third party.

In response to the continued ledger growth, light clients have been developed for Distributed Ledger networks. Light clients can verify the integrity of a ledger by downloading only a subset of the ledger. They are specifically designed to solve the efficiency problem, i.e., the high bandwidth and memory requirements that full nodes must meet — requirements that are inappropriate for consumer hardware and even some resource constrained devices.

In this thesis, we identify and investigate solutions to two problems. First, contrary to the ethos of the technology, the process of initial ledger synchronization can be considered relatively centralized, as it depends on a set of known nodes (i.e., bootnodes) that are both available and honest. Second, since light clients rely on and implicitly trust full nodes, security is an issue for current light clients, as they are vulnerable to various types of attacks, ranging from accepting maliciously forged data to Eclipse attacks. As a consequence, users typically opt for centralized solutions or rely entirely on trusted third parties.

We present Aurora — a set of three stochastic algorithms that is our solution to the above problems. The solution adds another layer of trustlessness into Distributed Ledger networks by analyzing the network structure that peers expose to a new peer joining the network or attempting to check the presence of a transaction. In a network where there are malicious actors, the solution discovers sets containing honest nodes that can be used for future interactions. If such sets can not be discovered, the solution signals to a Distributed Ledger client to cease operation. The solution allows a new node joining the network to initiate ledger synchronization with an honest node or to check that a transaction is present in the ledger without downloading the entire ledger or even a subset of it.

We provide a comprehensive overview of all features of Distributed Ledger Technology relevant to the implementation of our solution. The benefits of our solution are justified in the context of a project developed as part of this work, which enables privacy-preserving, verifiable and decentralized management of a product throughout its lifecycle in the supply chain. We provide the pseudocode of the solution, define the necessary parameters and their default values. The solution is evaluated on a Bitcoin-like network topology using a modified open source

blockchain simulator. We study the performance of the solution and analyze its time, communication and space complexity. As an implementation of our solution, we extend Trinity, an existing Python-based DL client for the Ethereum network, to run our solution and evaluated its execution on the Ethereum production network and show that the implementation of our solution consumes only about 0.31 MB of RAM and 1 MB of storage at runtime.

The experimental results of the proposed solution are presented and discussed. The proposed solution addresses the identified problems in a fundamentally different way than other state-of-the-art solutions. The proposed probabilistic variants of our solution outperform the analogous deterministic variants by up to two orders of magnitude in terms of communication complexity in a realistic scenario. Moreover, the solution has been shown to be hardware friendly and can be deployed on resource constrained devices. Thus, our solution significantly lowers the barrier to entry that Distributed Ledger Technology imposes on consumer hardware. It strengthens the trustless environment by incentivizing users who are not willing to run a full node or even a lightweight node to interact with the ledger in a trustless manner. The solution is also more robust and decentralized than other existing solutions, as it does not rely on bootnodes for initial network discovery and initial ledger synchronization. Finally, it can be integrated into existing solutions without making intrusive and backward incompatible changes.

Keywords: scalability, decentralization, resistance, light client, distributed ledgers, trustless

Prošireni sažetak

Ublažavanje utjecaja zlonamjernih čvorova u mrežama raspodijeljenih glavnih knjiga s čvorovima ograničenih resursa

Tehnologija raspodijeljenih glavnih knjiga omogućuje održavanje digitalne knjige u raspodijeljenom i decentraliziranom okruženju. U glavnu knjigu se zapisi mogu samo dodati, ali se ne mogu modificirati nakon umetanja. Knjigu održava grupa uzajamno nepovjerljivih sudionika, tj. čvorova, od kojih se očekuje da knjigu pohrane i verificiraju u cijelosti. Ključna svojstva raspodijeljenih knjiga su otpornost na cenzuru, decentraliziranost, nepromjenjivost i najvažnije, nepovjerenje (eng. *trustlessness*). Nepovjerenje je izraz koji označava da ne postoji treća strana od povjerenja odgovorna za rješavanje sukoba i održavanje globalnog stanja knjige. Raspodijeljene glavne knjige se pretežito primjenjuju za praćenje vlasništva nad digitalnom imovinom (npr. kriptovalutama). Međutim, tehnologija također omogućava izvršavanje proizvoljnog koda i pohranu proizvoljnih podataka, što znači da je tehnologija primjenjiva na šira područja od kriptovaluta.

Priroda tehnologije raspodijeljene knjige dozvoljava samo dodavanje novih zapisa te dovodi do stalnog rasta knjige s vremenom. Prosječan potrošački hardver nema dovoljno računalnih resursa za održavanje raspodijeljene glavne knjige. Za sinkronizaciju s mrežom je potrebno vrijeme, memorija i računalna snaga. Novi čvorovi, tj. čvorovi koji se nikada nisu sinkronizirali s mrežom troše značajnu količinu resursa prilikom preuzimanja knjige i provjere svih unosa od trenutka kada je knjiga stvorena. Iz tog razloga uređaji s ograničenim resursima ne mogu sudjelovati u održavanju knjige bez narušavanja nekih značajnih svojstava tehnologije. Nadalje, suprotno etosu tehnologije, početni proces sinkronizacije knjige može se smatrati relativno centraliziranim jer ovisi o skupu poznatih čvorova za pokretanje (eng. *bootnodes*) koji trebaju biti dostupni i iskreni.

Kao odgovor na kontinuirani rast knjige, razvijeni su klijenti prikladni za izvođenje na uređajima s ograničenim resursima (tzv. laki klijenti). Laki klijenti mogu provjeriti integritet glavne knjige preuzimanjem i verifikacijom samo podskupa knjige. Posebno su osmišljeni tako da budu učinkoviti i izbjegavaju zahtjeve koje čvorovi koji pohranjuju i verificiraju cijelu knjigu (tzv. puni čvorovi) moraju zadovoljiti, no ne bez narušavanja značajnih svojstava tehnologije. Najvažnije svojstvo koje se često narušava je nepovjerenje jer laki klijenti implicitno vjeruju punim čvorovima. Puni čvorovi dostavljaju lakim klijentima podskup glavne knjige koji im je potreban za verifikaciju integriteta knjige. Zbog toga su laki klijenti osjetljivi na različite vrste napada, kao npr. preuzimanje zlonamjernih ili nepotpunih podataka. Bez obzira na ovu ranjivost krajnji korisnici se često, umjesto pokretanja vlastitog čvora, odlučuju za korištenje lakih klijenata, tj. za centralizirana rješenja ili rješenja koja se oslanjaju na treću stranu od povjerenja.

U literaturi su predložene različite izmjene za lake klijente ili pune čvorove koje ublažavaju činjenicu da laki klijenti inherentno ovise o punim čvorovima ili da je proces početne sinkronizacije glavne knjige centraliziran. Neka rješenja ne narušavaju niti jedno značajno svojstvo tehnologije. Međutim, njihova implementacija je često invazivna za postojeće mreže raspodijeljenih glavnih knjiga jer zahtijeva značajne promjene postojećih čvorova koji često nisu kompatibilni unatrag. Takva vrsta promjena zahtijeva od svakog krajnjeg korisnika da ažurira čvor, odnosno mora doći do društvenog sporazuma na nivou cjele mreže, što je izuzetno teško u raspodijeljenom okruženju.

U sklopu disertacije razvijena je *Aurora*, rješenje u obliku skupa tri stohastička algoritma koji rješavaju prethodno navedene probleme bez narušavanja značajnih svojstava tehnologije. Rješenje se može implementirati u postojeće čvorove raspodijeljene knjige bez značajnih i unatrag nekompatibilnih promjena. Implementaciju algoritama Aurora nazivamo Aurora modulom. Rješenje je stohastičke prirode i generira skupove koji sadrže iskrene čvorove koje se može koristiti za tranzijentnu ili perzistentnu komunikaciju u prisutnosti zlonamjernih čvorova u mreži raspodijeljene knjige. Ako se željeni skupovi ne mogu generirati, rješenje signalizira klijentu da obustavi rad jer ne može pronaći iskreni čvor za ulazak u mrežu. Rješenje omogućuje novom čvoru koji se pridružuje mreži raspodijeljene knjige da na siguran način sinkronizira povijest knjige kroz komunikaciju s iskrenim čvorom ili da provjeri transakciju u knjizi bez preuzimanja cijele knjige, pa čak i podskupa knjige.

Skup osnovnih pretpostavki koje su preduvjet za razvoj rješenja bio je da pouzdani čvorovi za interakciju s mrežom potencijalno nisu dostupni kada se novi čvor pokušava pridružiti mreži. Novi čvor kontaktira jedan čvor koji je ujedno i prvi kontakt (dobiven npr., putem društvenog dogovora itd.), te nije svjestan niti jednog drugog mrežnog čvora. Prvi kontaktni čvor može biti zlonamjeran i dio tzv. klike jer surađuje s drugim zlonamjernim čvorovima kako bi prevario novi čvor. Nadalje, novi čvor može otkriti podskup mrežnih čvorova putem prvog kontaktnog čvora.

Nakon jasnog definiranja pretpostavki i iskazivanja problema, disertacija daje pregled svih značajki tehnologije raspodijeljene knjige koje su relevantne za implementaciju predloženog rješenja, naime prezentira strukture podataka koje se koriste za održavanje knjige i način postizanja konsenzusa te objašnjava pojam povjerenja pomoću kojega možemo ustvrditi da je transakcija uključena u glavnu knjigu i pojam pametnih ugovora koji se mogu koristiti za izvršavanje koda u okruženju raspodijeljene knjige.

Motivacija za provedeno istraživanje proizašla je iz razvoja rješenja za upravljanje opskrbnim lancem korištenjem Interneta stvari i tehnologije raspodijeljenih glavnih knjiga — DL-Tags. DL-Tags proširuje postojeći sustav TagItSmart koji integrira posebne oznake (npr. RFID, NFC i QR kodove) u proizvode za stvaranje takozvanih pametnih oznaka. Prije razvoja rješenja DL-Tags, potrošač proizvoda u sustavu TagItSmart morao je implicitno vjerovati tvorcu

pametne oznake i drugim dionicima u opskrbnom lancu da će pružiti autentične podatke u oznaci proizvoda. DL-Tags rješava ova pitanja pomoću decentralizirang upravljanja pametnim oznakama uz očuvanje privatnosti dionika lanca opskrbe i koje se mogu provjeriti tijekom cijelog životnog ciklusa proizvoda. Rješenje koristi platformu Ethereum za zapis interakcije među dionicima tijekom procesa razmjene proizvoda. Postizanjem konsenzusa o opisu proizvoda i stanju prijavljenom u raspodijeljenu glavnu knjigu, svi uključeni dionici i potrošači proizvoda mogu provjeriti autentičnost proizvoda bez otkrivanja svog identiteta. U sklopu disertacije, DL-Tags je detaljno opisan te je uključena analiza troškova svih transakcija na Ethereumu. Predloženo rješenje je osmišljeno da bude neovisno o konkretnoj implementaciji raspodijeljene knjige.

U sustavu DL-Tags jasno se identificiraju scenariji u kojima dionici mogu pokušati preuzeti glavnu knjigu i naći se u situaciji u kojoj nijedan čvor za pokretanje nije iskren ili dostupan, ili gdje dionici trebaju učinkovito provjeriti status transakcije koristeći uređaj ograničenih resursa, npr. pametni telefon. U tim scenarijima razvijeno rješenje pruža značajnu vrijednost.

Predloženo rješenje se u suštini oslanja se na niz hipergeometrijskih eksperimenata. Hipergeometrijski eksperiment sastoji se od slučajnog odabira uzorka s određenim brojem elemenata bez zamjene iz konačne populacije. Svaki element iz populacije može se klasificirati kao uspjeh ili neuspjeh. Pokus se može povezati s vjerojatnosti o broju uspjeha u uzorku.

Drugim riječima, Aurora otkriva čvorove u mreži raspodijeljene knjige istražujući mrežu u procesu koji se naziva prikupljanje. Prikupljanje se izvodi kao niz koraka koji se nazivaju izvlačenje. Izvlačenje se sastoji od zahtjeva novog čvora udaljenom čvoru kojim se traži popis poznatih čvorova udaljenog čvora i odgovora tog čvora koji sadrži podskup njegovih poznatih čvorova. Prikupljanje se nastavlja sve dok se ne zadovolji uvjet koji upućuje čvoru koji izvršava algoritam da prekine daljnje prikupljanje. Prikupljanje u suštini konstruira usmjereni aciklički graf nad topologijom apstraktne mreže čiji su vrhovi čvorovi, a rubovi između vrhova imaju dodijeljen smjer kada jedan čvor poznaje drugi čvor. Iz skupa jedinstvenih čvorova pronađenih tijekom prikupljanja (iz populacije), iz kojeg su neki čvorovi iskreni (tj., predstavljaju uspjeh u populaciji), a neki od njih nisu iskreni (tj., predstavljaju neuspjeh u populaciji), čvor Aurora odabire podskup čvorova bez zamjene (odabire uzorak) i provjerava je li vjerojatno, s unaprijed određenom vjerojatnošću, da ti podskupovi sadrže određeni broj iskrenih čvorova (uspjesi u uzorku). Tako generirane podskupove čvorova nazivamo iskreni vjerojatnosni skupovi.

Potrebno se je usredotočiti na dvije posebne vrste iskrenih vjerojatnosnih skupova relevantnih za domenu raspodijeljenih glavnih knjiga. Prvo, razmatraju se oni skupovi koji sadrže barem jedan iskren čvor koji jamči da novi čvor može saznati najnovije stanje knjige. Drugo, razmatraju se skupovi koji sadrže većinu iskrenih čvorova. To jamči da se iskren odgovor može saznati iz većine glasova članova iskrenog vjerojatnosnog skupa, bez potrebe za resursno intenzivnim procesom analize knjige.

Kako bismo ocijenili učinkovitost predloženog rješenja, uspoređuju se iskreni vjerojatnosni skupovi s njihovim determinističkim pandanom, koji se nazivaju iskreni deterministički skupovi. Kvantificira se broj jedinstvenih čvorova koje novi čvor mora otkriti tijekom prikupljanja prije nego što se iskren skup može konstruirati. Usporedba pokazuje da iskreni vjerojatnosni skupovi mogu biti do dva reda veličine manji u odnosu na odgovarajuće iskrene determinističke skupove, što znači da komunikacija s čvorovima iz vjerojatnog skupa generira do dva reda veličine manje poruka od komunikacije s čvorovima iz determinističkog skupa. Gornja granica broja poruka koje razmjenjuje čvor Aurora izražena je kao funkcija željene tolerancije na pretpostavljeni broj zlonamjernih čvorova u mreži. Nadalje, dokazano je da konstrukcija iskrenog skupa (vjerojatnosnog i determinisičkog) linearno ovisi o željenoj toleranciji na pretpostavljeni broj malicioznih čvorova, dok u realnom scenariju, veličina iskrenog vjerojatnosnog skupa može biti omeđena kvadratnim korijenom željene tolerancije na pretpostavljeni broj malicioznih čvorova. Nadalje, predstavljen je pseudokod rješenja, definirani su potrebni parametri za njegovo pokretanje i predložene su njihove zadane vrijednosti.

Sljedeći korak bio je definiranje holističkog postupka vrednovanja predloženog rješenja, u sklopu kojeg je razriješeno nekoliko problema i te je osmišljen postupak vrednovanja u dva koraka. Vrednovanje rješenja zahtijeva apsolutno poznavanje temeljne topologije mreže, ponovljivost eksperimenata te postojanje zlonamjernih čvorova u mreži, što je teško izvedivo u produkcijskom okruženju. Iz tih razloga, prvi dio postupka vrednovanja koji ocjenjuje učinkovitost rješenja proveden je u simuliranom okruženju. Radi boljeg razumijevanja potrošnje resursa resursa tijekom izvedbe na čvorovima ograničenih resursa, kao i provjere kompatibilnosti rješenja s postojećim rješenjima, drugi dio postupka vrednovanja sastoji se od implementacije rješenja i njegove integracije u klijenta otvorenog koda za mrežu Ethereum te izvođenja klijenta na produkcijskoj mreži Ethereuma.

Radi vrednovanja rješenja u simuliranom okruženju, definirana je realna topologija mreže na temelju otkrivene IPv4 mreže Bitcoina. Zbog nedostatka odgovarajućih alata za simulaciju, postojeći simulator otvorenog koda baziran na Javi, Simblock, izmijenjen je sa značajkama bitnim za vrednovanje rješenja. Dodatno, simulator je refaktoriran za korištenje Javinih streamova, dopuštajući paralelno izvršavanje i smanjenu potrošnju memorije prilikom izvršavanja simulacija. Empirijski rezultati izvedeni iz simulacija potvrđuju analitičke rezultate, odnosno rješenje radi očekivano s obzirom na unaprijed definiranu vjerojatnost ispravnog izvođenja. Kad se rješenje izvršava na uređaju bez značajno ograničenih resursa, rezultati pokazuju da uspijeva pronaći iskrene skupove za buduću komunikaciju s mrežom sve dok je broj malicioznih čvorova u mreži ispod 50%, ili će se izvršavanje klijenta obustaviti. Varijanta rješenja modificirana za uređaje s ograničenim resursima dosljedno obustavlja izvođenje kada broj malicioznih čvorova premaši oko četvrtinu.

Radi vrednovanja rješenja u produkcijskom okruženju, postojeći Ethereum klijent otvorenog

koda Trinity proširen je Aurora modulom. Pokretanje rješenja na produkcijskoj mreži Ethereuma pokazuje da Aurora modul integriran u klijenta troši približno $0.31MB$ radne memorije te $1MB$ podatkovne memorije, što ga čini primjenjivim na uređajima ograničenih resursa, kao npr. Raspberry Pi 2 model B. Nadalje, i dosljedno s postojećim istraživanjima, u mreži Ethereum još nema dovoljno punih čvorova koji su spremni posluživati lake klijente, odnosno udaljenim čvorovima nedostaju sposobnosti potrebne za korištenje rješenja za provjeru prisutnosti transakcije u knjizi. Nedostatak poticaja za posluživanje lakih klijenata na mreži širi je problem tehnologije raspodijeljenih glavnih knjiga i izlazi iz okvira ove disertacije.

U sklopu budućeg rada, performanse klijenta mogu se poboljšati paralelnim izvođenjem. Kako bi se skratilo vrijeme čekanja na izvršavanje prikupljanja, klijent može privremeno pohraniti rezultate prikupljanja, a zatim po potrebi ponovno koristiti postojeće iskrene vjerojatnosne skupove. Samo prikupljanje može se izvoditi dok je uređaj u stanju mirovanja ili tijekom punjenja. Ako krajnji korisnik poznaje osobu od povjerenja (npr. člana obitelji), iskreni vjerojatnosni skupovi mogu se podijeliti i ponovno koristiti putem društvenog ugovora. Konačno, korisniku bi koristilo intuitivno korisničko sučelje.

Zaključno, Aurora rješava identificirane probleme tehnologije raspodijeljene glavne knjige na bitno drugačiji način od ostalih suvremenih rješenja. Predložene vjerojatnosne varijante skupova koje koristi Aurora su manje od deterministčkih varijanti do dva reda veličine što značajno smanjuje komunikacijsku složenost u realnom scenariju. Nadalje, pokazano je da je Aurora modul prilagođen potrošačkom hardveru te da se može primijeniti na uređajima s ograničenim resursima. Stoga Aurora značajno snižava ulaznu barijeru koju tehnologija raspodijeljenih glavnih knjiga nameće takvim uređajima i omogućuje im interakciju s raspodijeljenom knjigom. Nadalje, rješenje potiče korisnike koji nisu voljni pokrenuti puni čvor, pa čak ni lakog klijenta, da vrše interakciju s raspodijeljenom glavnom knjigom bez narušavanja svojstva nepovjerenja. Rješenje je robusnije i decentraliziranije u odnosu na postojeća rješenja jer se ne oslanja na poznate čvorove za početno pokretanje kako bi se otkrila mreža te sinkronizirala glavna knjiga. Također se može koristiti za interakciju movih čvorova s postojećim mrežama raspodijeljene glavne knjige bez mijenjanja pravila konsenzusa ili struktura podataka korištenih za održavanje knjige, odnosno bez nekompatibilnih promjena.

Ključne riječi: blok-lanac, skalabilnost, decentralizacija, laki klijenti, raspodijeljene glavne knjige, nepovjerenje

Contents

Chapter 1

Introduction

Distributed Ledger Technology (DLT) enables the maintenance of a *Distributed Ledger (DL)*, which is a data structure replicated at *nodes* in a *Peer-to-Peer (P2P)* network. Distributed ledgers maintain the same data structure at each node (i.e., peer), where records can only be added and network nodes typically must verify each entry in the ledger, which must be fully replicated at each node. Any attempt to subsequently modify such a data structure is easily detected and extremely resource intensive.

A DL has its *state*, which is defined by the data contained in the ledger at a given point in time. Inputs called *transactions* can be written to or read from the ledger. When written to the ledger, transactions cause the state to change, so DLT can generally be viewed as a transaction-based state machine [1].

In terms of the ability to read from the ledger, DLT can be classified into two categories: *public* and *private*. In a public DL, any public entity can read from the ledger, as opposed to a private DL where only selected entities can read from the ledger [2].

In terms of the ability to write to the ledger, DLT can be further classified into two categories: *permissionless* and *permissioned*. In a permissionless DL, any public entity can write to the ledger. In practice, however, the ability to write is granted only after sufficient resources (e.g., hardware resources) have been extended (e.g.., enough computational power). In a permissioned DL, only selected entities are allowed to write to the ledger [2].

In the context of this thesis, we examine public and permissionless DLT solutions in more detail. We do so because of some relevant properties of such solutions, namely resistance to censorship, immutability, decentralized maintenance, and the elimination of the need for a central trusted third party responsible for resolving conflicts and maintaining a global state that is maintained by all parties that do not trust each other and some of these parties might be malicious or Byzantine. In other words, there is no mutual trust between peers, but the majority of peers in a DL network are assumed to be honest*. We call this feature *trustlessness*, and an

*More precisely, the majority of the voting power is assumed to be honest.

action respecting this feature *trustless*. Global truth is broadly achieved by a form of voting on the state of the ledger. It is assumed that the state is not compromised as long as the opponent holds the minority of the voting power.

Although public and permissionless solutions have their merits, they are fraught with a number of problems. For example, in a public and permissionless DL network, it is assumed that all nodes store and validate the ledger in its entirety. Ledger data replication and continuous verification provide secure technology, but at the cost of scalability. In addition, the append-only nature of the technology means that the ledger only grows over time. This property places significant storage requirements on nodes participating in the network. Nodes that store and validate the ledger in its entirety are referred to as *full nodes*. Participating in ledger maintenance is even more difficult when only consumer-grade hardware is used. This problem thus inadvertently leads to centralization, because if fewer nodes can store the ledger, then fewer nodes maintain the ledger.

Solutions that reduce the amount of data to be stored on a node in order for that node to operate within the network come in the form of *light clients*. By and large, these solutions need to store less data compared to full nodes, but in return sacrifice the trustlessness property of public DLs, as they largely depend on full nodes to be both honest (as opposed to Byzantine or explicitly malicious) and available, providing them with the metadata they need to operate. Even though the memory footprint of light clients is significantly smaller when compared to the full nodes, this still may not be enough to run a light client on a resource constrained device (e.g., a mobile phone). In turn, rather than running a full node or a light client, users tend to turn to centralized ledger explorers.

Furthermore, when a new node first joins the network, it must go through the process of *Transaction History Synchronization (THS)*. During this process, a new node downloads the ledger content. It does this by querying surrounding nodes about the latest ledger state. The surrounding nodes are determined by first contacting a group of known nodes called *bootnodes*. Bootnodes are typically hardcoded in the client and are usually assumed to be both honest and available, which may not always be the case [3, 4, 5]. If bootnodes are not honest and available, a new node entering the network may fall victim to the influence of malicious actors. Interaction with such actors can have various detrimental consequences and range from transactions being dropped to double spending.

In response to the above problems, the focus of this research is to identify mechanisms that positively impact the scalability of DLT networks while reducing the amount of resources that a node must allocate as its contribution to participate in the network. The goal of the dissertation is firts to create a mechanism that improves THS for a node entering an existing DLT network where malicious nodes are actively trying to subvert it, and second to design a mechanism that efficiently checks the presence of a transaction in a ledger without relying on a trusted

central point and without having to download the entire ledger or even a portion of the ledger (i.e., *Transaction Presence Checking (TPC)*). Both mechanisms should be suitable for resource constrained devices.

1.1 Motivation and background

We continue to elaborate the motivation which drove us to develop our solution. It is not trivial for a consumer-grade device to participate in a DL network and ledger maintenance. To better understand this assertion, we will take a closer look (without loss of generality) at a specialization of DLT, namely *blockchain* and its two prominent solutions, *Bitcoin (BTC)* [6] and *Ethereum (ETH)* [7]. In particular, we will discuss the amount of memory needed for ledger maintenance if a user is running a full node or a light client.

For now, and for the sake of motivation, we will introduce a number of basic terms related to the blockchain. A more detailed explanation of a blockchain and DL in general is given in the following sections (see Chapter 2). Hereinafter, it is assumed that a blockchain is a data structure used to maintain a DL. It consists of ordered units called *blocks*. Blocks contain *block headers* and transactions. Each block header contains, among other metadata, a reference to its predecessor in the form of the predecessor's hash. The initial state is hard-coded in the first block, the *genesis block*. Unlike other blocks, the genesis block has no predecessor.

When a full node enters a blockchain network, the node should download the entire ledger and verify all transactions in all blocks since the genesis block. In the case of BTC, the ledger size as of May 2021 is approximately 347 *GB* (as shown in Fig. 1.1[†]) and in the case of ETH, the ledger size over the same period is approximately 7334 *GB* (as shown in Fig. 1.2[‡]). Storing such a volume of data is not trivial.

Should a user decide to run a light client in BTC or ETH, it is sufficient to download the *header chain* from a trusted node, which is an ordered sequence of block headers instead of whole blocks. In April 2020, the size of the header chain in BTC was relatively small (50 *MB*), while the size of the header chain in ETH was, considering resource constrained devices such as mobile phones, a non-negligible 5 *GB* [8], growing by approximately 1 *GB* per year [9]. In this context, solutions that reduce the amount of data a user must download before interacting with the ledger, while maintaining the feature of trustlessness (fully or to some degree) are most welcome.

As already stated, apart from the issue of memory requirements, the process of THS depends largely on the presence of honest bootnodes. A recent *Defense Advanced Research Projects Agency (DARPA)* study[§] has shown that DLT solutions can still be considered relatively cen-

[†]Source: blockchain.com — https://www.blockchain.com/charts/blocks-size
[‡]Source: Etherscan.io — https://etherscan.io/chartsync/chainarchive
[§]`https://assets-global.website-files.com/5fd11235b3950c2c1a3b6df4/`

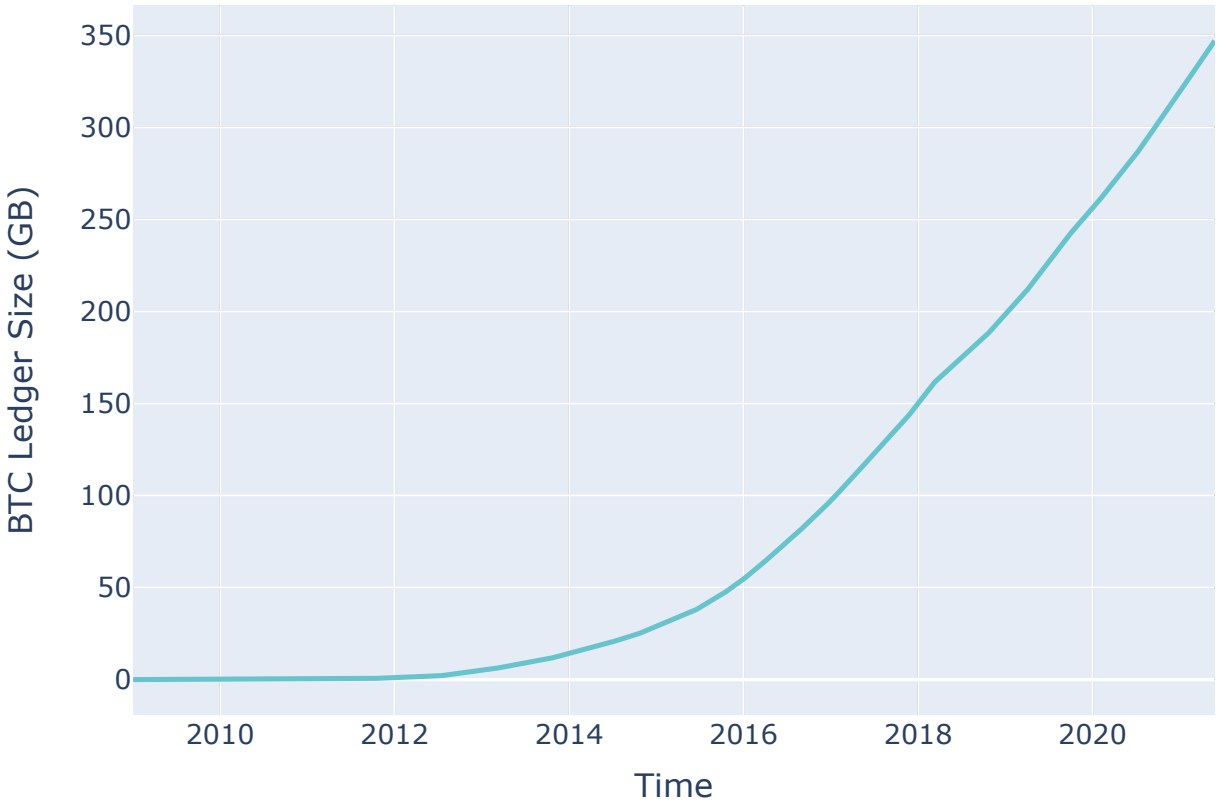

Figure 1.1: BTC ledger size as of May 2021

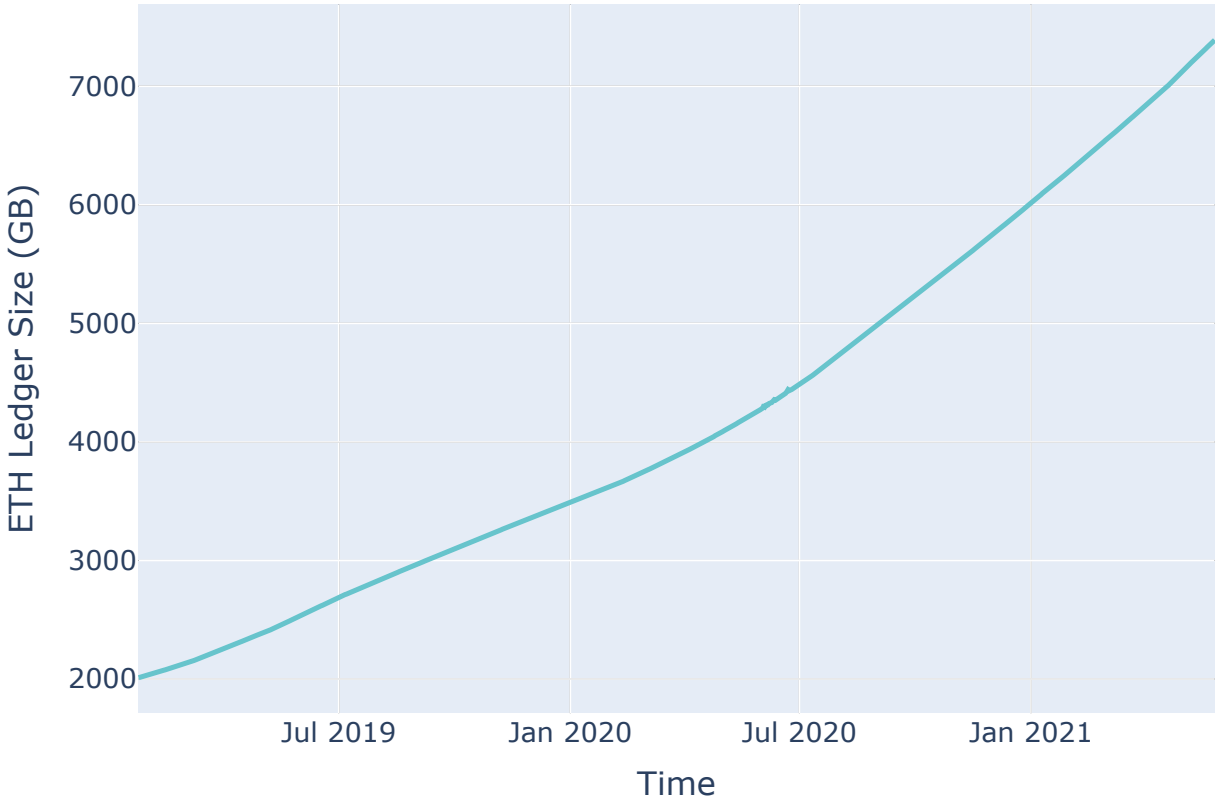

Figure 1.2: ETH ledger size as of May 2021

62af6c641a672b3329b9a480_Unintended_Centralities_in_Distributed_Ledgers.pdf — Online; accessed 03.08.2022

tralized. Unavailability of bootnodes or their Byzantine behavior makes a new node vulnerable to various types of attacks, for example Eclipse attacks [10, 11]. One study reports that centralization at the level of autonomous systems makes BTC vulnerable to routing attacks causing bootnodes to become unavailable [12]. Furthermore, bootnodes can become unavailable due to *Denial of Service (DoS)* attacks. To circumvent the problem, users may manually add bootnodes, but such nodes may exhibit malicious behavior. When a malicious node is a first contact for a node interacting with a blockchain network, the consequences can range from a waste of time and resources at best to a state where the victim is unaware of the existence of a longer chain at worst because no honest node is available to advertise it. Another study provides evidence that such an event has been observed: Bitcoin Cash was the victim of a sybil attack with up to 5000 malicious nodes [13]. In conclusion, it is safe to say that it is assumed that communication with an honest node is preferred over communication with a malicious node.

In light of the above, we can draw the following two very broad conclusions about the current state of DLT, which are the driving factors behind this research:

1. The maintenance of a DL as well as the interaction with a DL is not trivial due to high resource consumption, and the reduction of this barrier is of benefit to a DLT ecosystem.
2. The process of THS may be regarded as centralized to an extent, as it depends on a set of well known nodes which are available and honest.

1.2 Problem statement

We begin by formalizing the framework from which we will draw our terms and definitions. Let us consider a blockchain network S containing a subset of malicious nodes M, $M \subset S$. We assume the following:

Assumption 1. The majority of the network voting power is honest.

The existence of an honest majority is a standard assumption in DLT networks [7, 14, 15] and is interpreted in accordance with the consensus mechanism used by the network. For example, Proof of Stake assumes that an honest majority owns the majority of network tokens, and Proof of Work in Bitcoin assumes that an honest majority is able to construct a chain with the highest cumulative difficulty.

Assumption 2. Bootnodes may become unavailable, but a new node a joining the network can discover at least one first-contact node fc.

The discovery of potential bootnodes is a standard procedure in DLT and P2P networks. Here we assume that bootnodes may become unavailable due to malicious interference, as recognized by the prominent DLT solutions. For example, Bitcoin uses a pseudo-random method to obtain

a subset of potential bootstrapping peers, cached peers for future connections, DNS to identify bootstrap candidates, and, as a last resort, a hard-coded list of remote nodes. However, one study found that despite all of the above countermeasures, malicious influence is still possible [16]. Hence, first contact nodes may be either trusted or malicious. First-contact candidates can also be added manually to the set of nodes known to node a[¶].

Assumption 3. A subset of nodes from S, denoted by Γ, contains nodes discoverable by node a via fc.

By *discoverable nodes* we refer to those nodes that node a can encounter or become aware of their existence when its first-contact node is fc.

Assumption 4. The first-contact node fc may be malicious ($fc \in M$), and may collude with other malicious peers to subvert node a.

Malicious nodes have incentive to collude since they have the opportunity for financial gain. For example, a victim of such collusion can be a seller of products. The seller can be convinced that a transaction is present in a DL, and then sends goods under the assumption that payment was received, when in fact no payment has been made.

Assumption 5. An adversary has limited computational resources and can spawn up to κ nodes, where $|M| = \kappa$. The value of κ can be estimated and is denoted as $\hat{\kappa}$.

We call κ a *desired malicious node tolerance* to identify the number of malicious nodes which our solution can circumvent. The estimation of κ is discussed in the following sections, where we also explain how this assumption can be relaxed.

Assumption 6. The data contained in a block is part of an integrity-validating structure, and a node can verify that the transaction is indeed present in that structure. It is assumed that all necessary additional metadata is available to the node.[‖]

We justify and elaborate our assumptions further in the scope this work, and also explain under which conditions some of our assumptions can be relaxed. Given these assumptions, we identify two specific problems that node a faces and that arise from the assumption that malicious or Byzantine influences are possible:

Problem 1. Node a wants, with high probability, to start the THS process while communicating with an honest node, as opposed to starting the process while communicating with a malicious node. If unable to communicate with an honest node, node a wants to abort further operation and interaction with the ledger.

[¶]For example, Bitcoin offers the *addNode* procedure

[‖]Such structures are present in prominent DLT solutions in the form of Merkle trees and Merkle proofs.

Problem 2. Node a wants to check, with high probability, whether a particular transaction with an identifier tx_{id} is present in a DL without downloading the entire ledger or a significantly large portion of the ledger (e.g., the header chain). If unable to communicate with an honest node, node a wants to abort operation.

In the scope of this research and to the best of our knowledge, we offer a unique solution to both Problem 1 and Problem 2.

1.3 Scientific contribution

The scientific contribution of this thesis, which is the result of research conducted using a research methodology that follows the established practices in the field of computer science and distributed systems, can be summarized as follows:

A new probabilistic honest set creation algorithm is defined: In distributed ledger networks with malicious nodes which are actively trying to subvert a new node interacting with the network, the algorithm creates a subset of network nodes which contain a predefined amount of honest nodes with a predefined probability. This subset is called a *probabilistic honest set*. If unable to create a probabilistic honest set, the algorithm signals the new node to abort operation. The algorithm is presented in Section 4.3 (see Algorithm 2).

A new probabilistic transaction history synchronization algorithm is defined: In distributed ledger networks with malicious nodes which are actively trying to subvert a new node interacting with the network, the algorithm communicates with a set of remote network nodes created by Algorithm 2, where the set contains at least one honest node with a predefined probability. The algorithm ensures that the transaction history can eventually be synchronized from an honest node. The algorithm is presented in Section 4.3 (see Algorithm 4).

A new probabilistic transaction presence checking algorithm: In distributed ledger networks with malicious nodes which are actively trying to subvert a new node interacting with the network, the algorithm communicates with a set of remote network nodes created by Algorithm 2, where the set contains a majority of honest nodes with a predefined probability. The algorithm ensures that the presence of a transaction within a DL can be inferred by a majority vote. The algorithm is presented in Section 4.3 (see Algorithm 5).

A new procedure for evaluating the algorithms in a selected DL network with resource constrained nodes is specified: A procedure to evaluate the proposed algorithms is twofold: a simulation in a realistic BTC network, in the scope of which the complexity and efficiency of the

solution has been expressed and measured in the presence of malicious nodes, (see Section 4.4) and an implementation of the solution in an open source ETH client, in the scope of which the consumption of storage and *Random Access Memory (RAM)* is measured. Furthermore, the duration of the execution of our solution is documented. The and compatibility of our solution with the ETH mainnet is elaborated (see Section 4.5).

1.4 Thesis structure

The rest of the thesis is organized as follows: Chapter 2 surveys the features of DLT that are relevant to our solution. Chapter 3 presents a DLT solution that tracks the origin of a product as it moves through the supply chain, which we use as a motivating usage scenario to highlight the practical benefits of our solution. In Chapter 4 we present Aurora, introduce all relevant terms and definitions, present the pseudocode of the *Probabilistic honest set construction algorithm*, the *Transaction history synchronization algorithm* and the *Transaction presence checking algorithm*. The chapter also analyzes time, communication and space complexity of the algorithms, and verifies the efficiency and resource consumption of the solution in a simulated environment and in a production DL network. Chapter 5 highlights and summarizes the main scientific contribution of this thesis. Chapter 6 concludes the thesis and opens a discussion on future research directions. The nomenclature, list of acronyms and an index are included in the end of the document for the convenience of the reader.

Chapter 2

Overview of distributed ledger technology

Within this chapter, we take a closer look at DLT and its properties relevant to implementing our solution in an existing DL client, as well as the interaction of such a client with an existing DL network. We call a DL node updated to be able to execute our solution an *Aurora node* (a more formal definition is given in Chapter 4). We will systematically review a generic hierarchical architecture of a typical DLT system as defined in [17] and explain the impact of our solution on each layer. Such a system can be split into five distinct layers, which are the *Data, Network, Consensus, Contract* and *Application* layers. These layers, as well as a high-level architecture of our solution are displayed in Fig. 2.1. The role of the Aurora node in such an architecture is to observe messages containing neighboring peer information. Using neighboring peer information, the Aurora node is able to construct a probabilistic honest set. Members of the constructed probabilistic honest set are then queried for DL metadata relevant for the successful execution of THS and TPC.

2.1 Data layer

The Data layer describes how data is stored in a ledger. A fundamental prerequisite for DLT data structures is the existence of asymmetric cryptography. DLT solutions rely heavily on the use of *Public Key Infrastructure (PKI)* to generate public and private key pairs. Possession of a private key allows the owner to change the state of the ledger to create digital signatures or sign transactions (e.g., to transfer funds from one entity to another). In ETH, a private key consists of 256 random bits and should always remain private. The public key is associated with a private key. In the example of ETH and BTC, a public key is a point on an elliptic curve. Elliptic curve cryptography is based on the discrete logarithm problem and is expressed by mathematical operations (addition and multiplication) on the points of the curve. Note that both Ethereum and Bitcoin use the *secp256k1* curve [18, 19]. A ledger address is a unique identifier derived from a public key. For example, ETH uses the *Keccak-256* function on a public key to generate

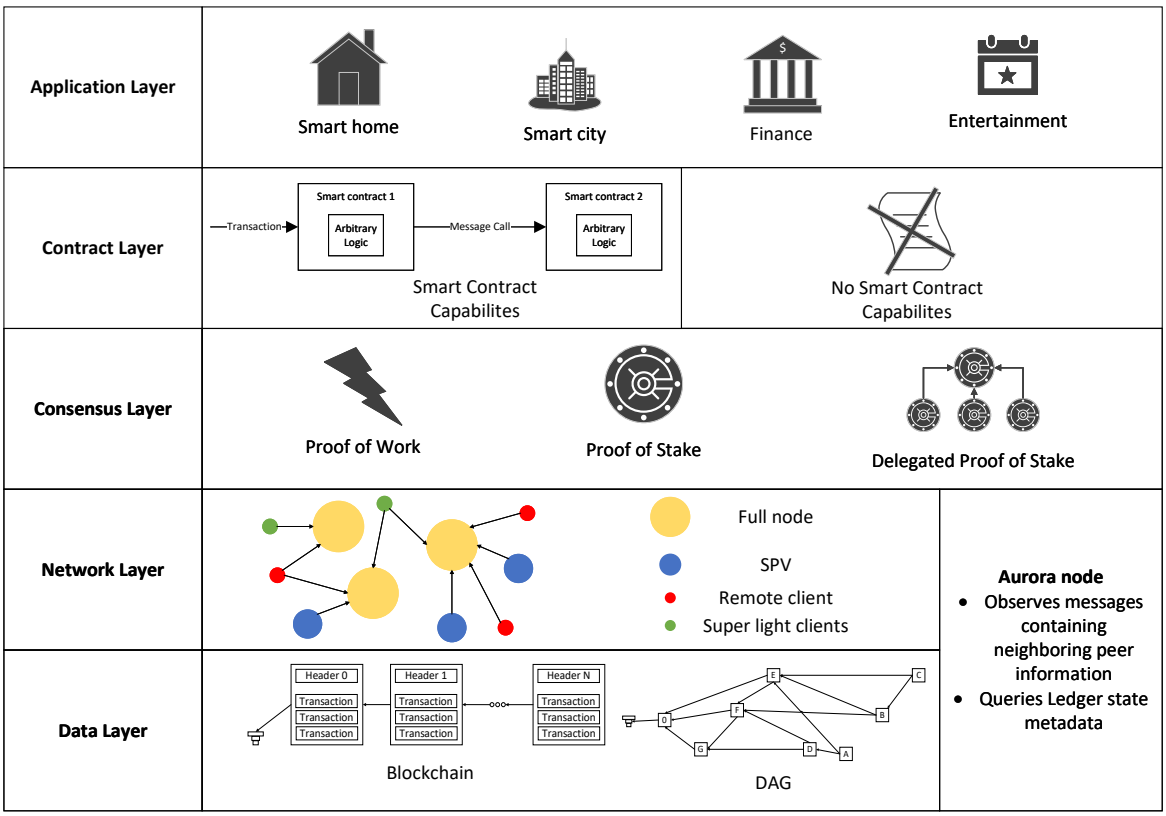

Figure 2.1: High-level generic DLT system architecture

a digest of the key. The last 20 bytes of the digest are a hexadecimal number representing an address.

2.1.1 Data structures

While the most prominent structure is a blockchain, there exists more than one data structure used to maintain a DL. The underlying data structure can also be a *Directed Acyclic Graph (DAG)*, or even a hybrid combining both data structures.

Blockchain

The notion and definition of a blockchain has already been rudimentarily discussed in Section 1.1 as an ordered sequence of blocks. A block consists of a block header and a set of transactions. An illustration of a blockchain structure is shown in Fig. 2.2.

In BTC, a block header contains a reference to a previous block and the protocol version. Furthermore, it contains the consensus related metadata [19], i.e., the *difficulty target*, the *timestamp*, and the *nonce* which we will cover in depth in Section 2.3. In addition, the block header contains the digest of an integrity check structure, i.e., it is the root of a *Merkle tree*, which is covered in depth in Section 2.1.2

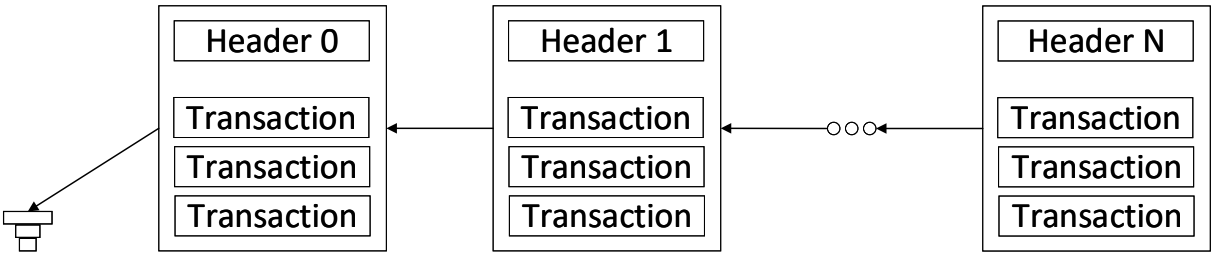

Figure 2.2: Blockchain data structure. Each block except the genesis block references its predecessor. Each block consists of transactions and a block header containing various metadata.

The ETH block header is more complex, because it contains more metadata compared to a BTC header. Regardless, all the metadata contained in a BTC block header is included in an ETH block header, which means that both solutions share common data structures that our solution relies on (e.g., the existence of an integrity validating structure like a Merkle tree).

Directed Acyclic Graph

As already stated, an underlying DL data structure can also be a DAG. An example DAG based data structure, the *IOTA Tangle* [20], is displayed in Fig. 2.3. IOTA stores transactions in nodes*, (as opposed to blocks), where each node holds a single transaction. In the IOTA Tangle, every transaction references two previous transactions, except the *genesis transaction*, which has no predecessor. Note that our solution is independent of the Data layer, which means it can be used in DAG based DLs, as long as the assumptions listed in Section 1.2 hold.

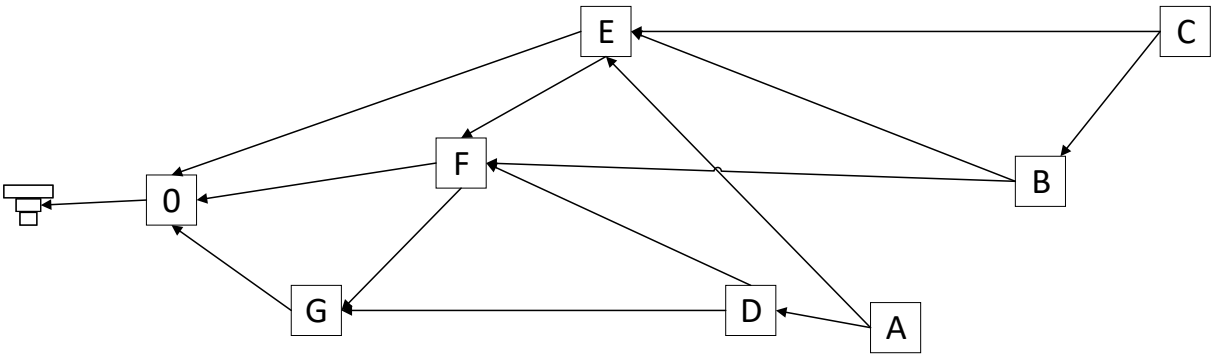

Figure 2.3: IOTA DAG data structure. Every transaction references two previous transactions, except the genesis transaction

Another example of a DAG data structure is the *block lattice* used by *Nano* [21]†. Much like IOTA, Nano stores transactions in nodes where each node holds a single transaction. In Nano, each account is associated with its own account chain corresponding to the account's transaction/balance history, as shown in Fig. 2.4.

*Not to be confused with a DL node.
†Former RaiBlocks.

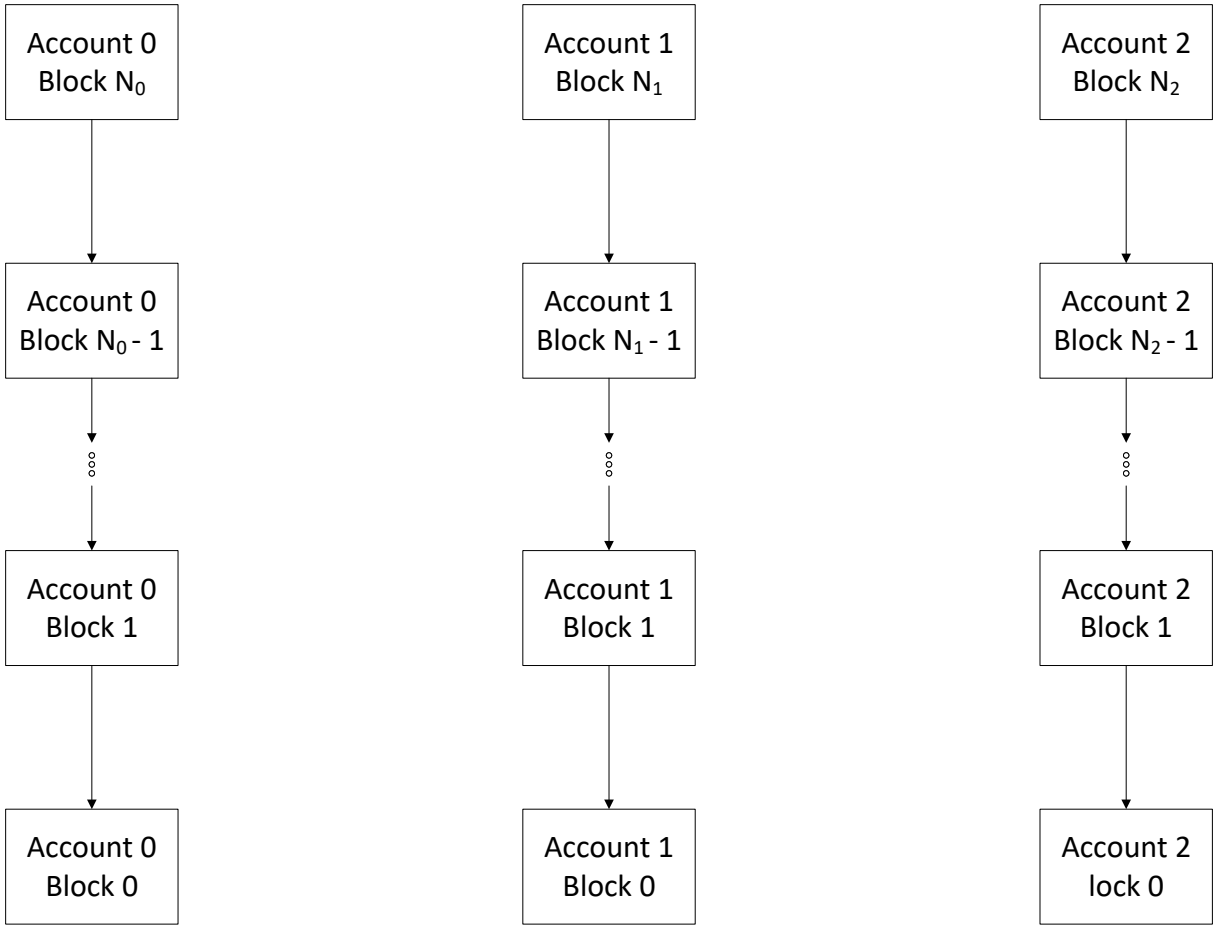

Figure 2.4: Nano DAG data structure. Every account is granted a dedicated account chain and all account chains form the block lattice.

Hybrid approaches

There are solutions that incorporate both blockchain and DAG data structures. An example is *Phantom* [22] and its heuristic variant *GHOSTDAG* [23], which is designed for improved performance compared to Phantom[‡].

The use of a DAG structure in the context of hybrid solutions is intended to increase the throughput of transactions in the network and effectively increase scalability by exploiting the *soft fork* phenomenon, as described in Section 2.3.

2.1.2 Merkle tree

Data stored in a DL is often contained in an integrity validating structure (as required by Assumption 6, which is a standard assumption in DLT networks). For example, in BTC and ETH, within each block, transactions are associated with a binary tree of hash pointers referred to as Merkle trees [24, 25]. Merkle trees are well-known tools in cryptography that allow efficient proof of a data item's membership in a set without revealing the entire set [26]. Both ETH and

[‡]Phantom requires the solution of an NP-hard problem while GHOSTDAG does not.

BTC store transactions within Merkle trees. Depending on the DL network, there may be more than one Merkle tree associated with a block. For example, ETH uses four such trees, while BTC uses a single tree.

An example Merkle tree is displayed in Fig. 2.5. Using a Merkle tree, one can generate a *Merkle proof*, which is a proof of the existence of certain data in a given dataset. The data corresponds to a *leaf* (a data item in a Merkle tree with no children) in a Merkle tree. The root of a Merkle tree is called the *Merkle root*. To generate a Merkle proof, both the leaf and the Merkle root should be known a priori. Segments of the tree in the path from the leaf to the Merkle root are either provided by the prover, or computed and serve to reconstruct and confirm the a priori known Merkle root [26].

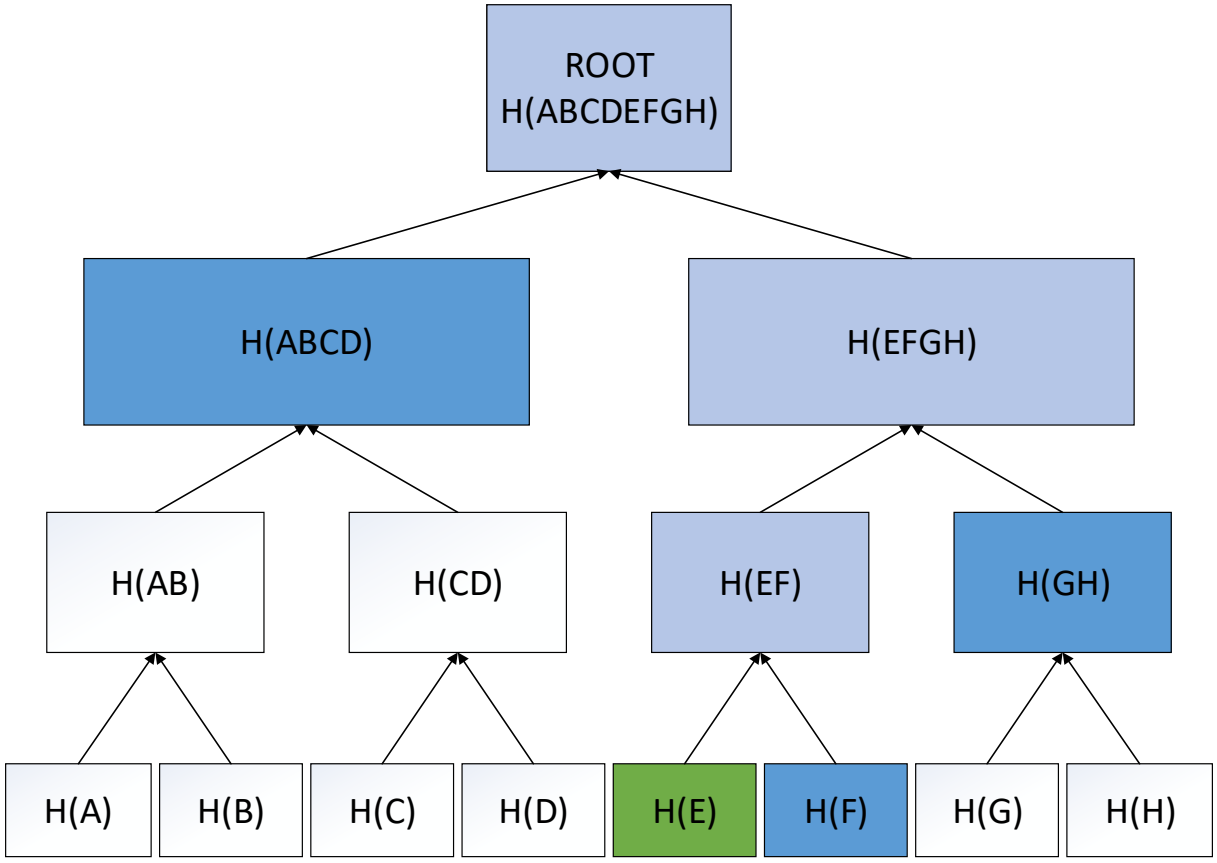

Figure 2.5: A Merkle tree and a Merkle proof. The hash of the transaction that needs to be proven to belong to the dataset is colored green. The Merkle root is labeled as ROOT. Segments of the tree that are provided by the prover are colored dark blue, and the intermediate hashes that are computed are colored light blue.

Since the reverse-engineering of a Merkle proof is highly unlikely, the existence of a proof guarantees the inclusion of an element in a dataset (e.g., a transaction in a block). Our work depends on usage of such structures to prove that a transaction is present in a ledger, which we will discuss in detail in Chapter 4. Note that our solution can also be applied in DL solutions that do not rely on integrity validating structures such as Merkle trees, however this makes our solution less secure.

The existence (or absence) of a Merkle proof has no meaning without a broader context of how consensus is achieved in a DL network. In addition to the fact that a transaction in question is actually present in a block, it is important that the network recognizes the entire block (and therefore any Merkle proof generated for transactions in such a block) is part of the ledger. We address this topic in Section 2.3.

2.2 Network layer

The Network layer describes the network structure used by DLT solutions, which are in essence P2P networks, and allows network nodes to exchange peer information and ledger metadata, which is a standard assumption in DLT networks [7, 15, 25].

Nodes that follow a DLT protocol can query other remote nodes for their neighboring peers using an active-peers request message (*ping*). The queried nodes respond with an active-peers response messages (*pong*). Furthermore, nodes can ask for the canonical ledger head by sending a *status request*. A remote node responds to a status request message containing the remote's ledger *status*. Finally, nodes can request a proof that a transaction is included in a ledger block by sending a *proof request* messages. A remote node responds to a proof request message containing a specific Merkle proof. Our solution relies on the existence of the listed protocols messages, which we map to the actual messages in protocols supported by ETH in Chapter 4. Furthermore, before node a can begin exchanging these messages with other remote peers, it must go through a process called *bootstrapping*. Using bootnodes as the initial contact nodes, node a will discover the rest of the network.

However, in a realistic setting, the possible presence of malicious nodes must be accounted for. A malicious or Byzantine peer may choose to respond with fake pong, *status response*, or *proof response* messages, or not to respond at all. Bootstrapping is very sensitive for node a, as a malicious bootnode can severely compromise the view of the network for node a [5, 25]. To shed more light on these types of attacks, we will take a closer look at the influence of attackers at the network layer and present some existing attacks on prominent DLT solutions.

2.2.1 Adversarial influence in the Network layer

The influence of malicious actors in P2P networks has been researched in [27, 28, 29], with recent work focusing on DLT solutions such as BTC [5, 10, 12, 30] and ETH [5, 30, 31, 32, 33, 34], which justify Assumption 4. The identified problems are diverse and include, but are not limited to the following:

•Eclipse Attacks, where node a is completely surrounded by malicious actors.

•DoS and *Distributed Denial of Service (DDoS)* attacks, where bootnodes can become

unavailable or overloaded, as was the case with Skype login nodes [35].

- •Manipulation of routing advertisements (i.e., BGP hijacks), where transactions can get dropped or the network can become partitioned. For example, 80% of BTC traffic is routed through autonomous systems belonging to a single entity [13].
- •Sybil attacks where node a can become surrounded by an attacker using multiple identities. A recent study documented a Sybil attack on the Bitcoin Cash network with up to 5000 Sybil nodes [13].
- *Fake bootstrapping*, where node a initiates the process of bootstrapping with a malicious peer.

Countermeasures to these attacks have been addressed in the existing literature, and we provide a detailed overview of them in Section 4.1 in the scope of the related work. For now, we can confidently state that malicious influence in the Network layer is possible and has been documented. This malicious influence may cause bootnodes to become unavailable or behave in Byzantine or malicious ways, as per Assumption 2.

The unavailability of bootnodes is not a problem in itself. A candidate bootnode can be discovered in many ways. For example, existing approaches to counter Fake bootstrapping can be used, like the Random Address Probing method [30]. A bootnode can also be found via social contract, *Instant Messaging (IM)* channels or similar. Prominent DLT solutions have mechanisms that allow node a to connect to other remote peers manually, for example BTC offers the *addNode* procedure[§] or the bootnodes flag for *geth*, an implementation of the ETH protocol in the *go* programming language[¶]. When a node discovers a first contact node fc, the rest of the network can be discovered, as per Assumption 3. Nonetheless, if fc is Byzantine or malicious, node a still may fall victim to malicious influence.

2.2.2 Ledger size reduction methods and ILD optimization

Ledger size reduction: The fact that participation in ledger maintenance requires significant amounts of storage is recognized by leading DLT solutions. BTC clients offer a *pruning* mechanism that deletes blocks after the entire ledger has been downloaded and validated, and only a small amount of data is retained. The data is retained to forward the most recent blocks to peers and compensate for the fact that reaching consensus in BTC is a stochastic process, as described in Section 2.3.2. The advantage of this approach is that storage space is saved, while the disadvantage is that other participants are no longer able to download the entire history from a pruned node[‖]. Similar to BTC, ETH offers a pruning mechanism. ETH keeps track of the deltas in the global state maintained by a Merkle tree.

[§] `https://developer.bitcoin.org/reference/rpc/addnode.html`

[¶] `https://geth.ethereum.org/docs/interface/peer-to-peer`

[‖] `https://github.com/bitcoin/bitcoin/blob/master/doc/release-notes/release-notes-0.11.0.md`

THS optimization: The fact that THS requires significant resources has also been recognized by leading DLT solutions. For example, ETH implements a *fast sync* to trade processing power for bandwidth usage[**]. Instead of processing all transactions since genesis, fast sync downloads the header chain and checks its consistency. By using hashes from the header chain, additional metadata (e.g., Merkle trees as defined in Section 2.1.2) can be downloaded from another remote node in a trustless manner. When the chain reaches recent state (*chain head* − 1024 blocks), the remaining blocks are fully processed. Prominent solutions also recognize the possibility of downloading a malicious chain. BTC[††] uses a headers-first strategy [36] in which it first downloads the chain head from a bootnode, partially validates the headers, and downloads blocks in parallel. Similarly, ETH listens for the latest status response message from a neighboring node[‡‡]and forms a *header skeleton*, meaning that each 200th block header is downloaded from the remote node, and the remaining block headers are downloaded in parallel from neighboring peers identified using ping and pong messages [25].

The ledger size reduction methods and THS optimization mechanisms share a common problem: they all require the presence of at least one honest node, meaning that Problem 1 is still present regardless of the mechanisms mentioned above.

2.3 Consensus layer

The Consensus layer ensures that each node agrees about the state of the ledger. One of the remarkable features of public and permissionless DLT solutions is their ability to achieve consensus in an environment where the number of participants is unknown, participants can enter and leave the network at will, and participants may or may not be malicious and do not trust each other, as long as an adversary always holds a minority of the voting power [15], as per Assumption 1.

2.3.1 Consensus mechanisms

Nakamoto consensuses revolves around the selection of a participant who is allowed to attach a block to a DL through a sort of lottery function [15]. BTC (amongst others) achieves consensus using a method called *Proof of Work (PoW)*, but there exist many other methods, for example *Proof of Stake (PoS)* or *Open Representative Voting (ORV)*.

Proof of Work: In PoW, the first participant that successfully solves a cryptographic puzzle wins the lottery and gets selected as the leader. The winner is allowed to append data to

[**]`https://github.com/ethereum/go-ethereum/pull/1889`
[††]Bitcoin Core 0.10.0
[‡‡]`https://github.com/ethereumproject/go-ethereum/wiki/Blockchain-Synchronisation`

the ledger. For example, BTC uses partial hash inversion as its cryptographic puzzle function, which requires the hash of a block, along with a free variable in the function called nonce to start with a predefined number of zero bits. More precisely, the PoW puzzle must produce a block hash that is equal to or less than the difficulty target, which is a numerical representation of the maximum number of a block hash when interpreted as a hexadecimal number. A higher difficulty target decreases the difficulty of the PoW puzzle while a lower difficulty target increases the difficulty of the PoW puzzle. The function is intentionally hard to solve for security reasons — to manipulate the ledger, an attacker would need to have a supermajority of computational power on the network, making it expensive to perform an attack. Nodes that generate blocks in a PoW network are called *miners*. The process of block generation is called *mining*. Miners have an economic incentive to mine. In return for mining, miners receive tokens in the network.

Proof of Stake: In PoS, nodes stake tokens in the network as a guarantee that they will follow protocol, as opposed to offering computational resources. In the ETH network, the native token is called *Ether*, while in BTC the token is a homonym with the name of the network, also called Bitcoin.

Open Representative Voting: In ORV, as defined by *Nano (XNO)*, each node must choose another remote node to act as its representative. A representative may be changed over time. The election is done by delegating network tokens XNO to the elected representative. Representatives vote to resolve conflicts, and their vote is weighted according to the delegated stake. In the event of a conflict, the transaction that received the most votes [21] wins. No voting overhead is required for a transaction with no issues.

Without loss of generality, we will discuss only PoW based solution for implementing our solution — the principles remain the same, regardless of the consensus method.

2.3.2 Global truth as a Stochastic process

Assuming that there is a supermajority of honest nodes (more precisely, that the honest supermajority of nodes holds the majority of voting power), the global truth in a DL is found in the *canonical chain*. Any chain that diverges from the canonical chain is called a *chain fork*. A chain fork constructed by malicious actors will be shorter and can be safely ignored by honest nodes.

The presence of a chain fork in blockchain networks does not occur exclusively as a result of malicious actors. Forks are an integral part of network operations, resulting from the fact that multiple nodes are trying to solve the PoW puzzle in parallel, and two different miners can solve the puzzle at roughly the same time, resulting in two different blocks which are referencing the same predecessor. This phenomenon is called a soft fork. All miners track both

forks, but commit to a single fork. After one of the two forks becomes longer, all miners on the shorter chain will abandon it in favour of the *longest chain*. An example of soft forks is shown in Fig. 2.6.

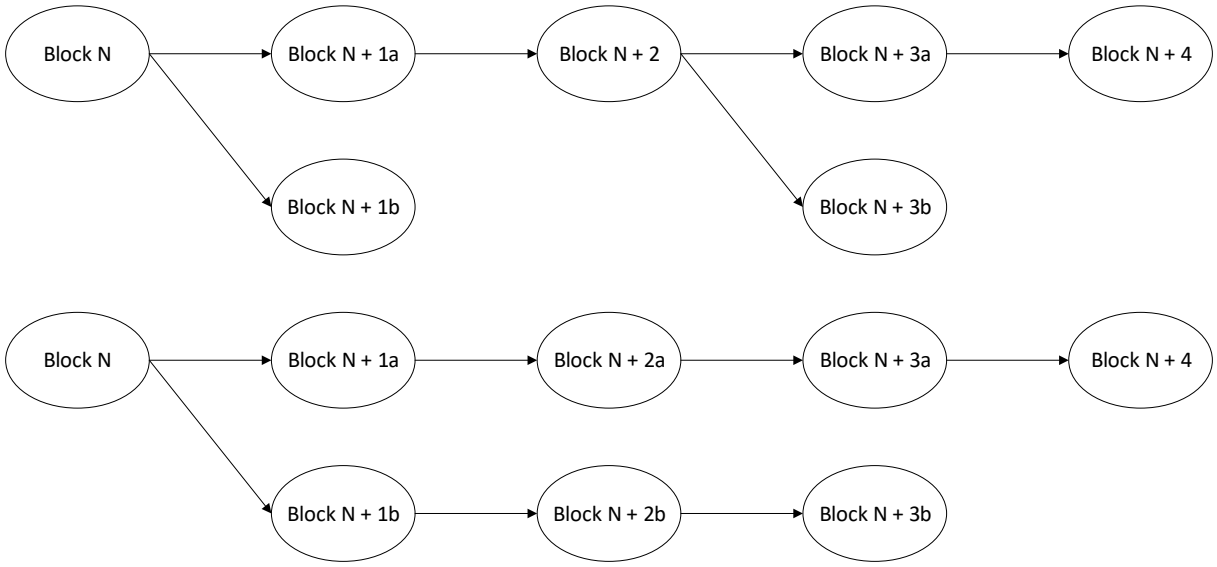

Figure 2.6: Diagram demonstrating temporary blockchain forks. The top chain depicts a typical fork, while the bottom chain depicts an atypical fork.© 2018 IEEE

The fact that a fork is abandoned means that all transactions in that fork are not included in the canonical chain. Given that a soft fork can occur at any time, it is uncertain to assume that the transactions present in the last block in the chain (i.e., the chain head) will remain part of the longest chain in the future. As the length of the chain increases, the probability that a block will remain part of the longest chain increases. The number of blocks that must be appended above the observed block before it can be said with a high degree of certainty that it remains part of the longest chain depends on the implementation (e.g., 6 for BTC and $5-11$ for ETH [15]).

How the canonical chain (more precisely, the global truth) is determined when taking into account the existence of forks depends on the implementation of a DL. In BTC, for example, the canonical chain was originally determined using the longest chain (as shown in Fig. 2.7), which was the chain with the most blocks. However, this rule was later replaced in favour of the *strongest chain* rule (as shown on Fig. 2.8), which takes into account the cumulative difficulty of the PoW puzzle.

In summary, in the context of our solution, there exist DL solutions where reaching consensus is a stochastic process. In such solutions, one has to wait until a transaction is settled. More precisely, one has to wait so long that it is highly unlikely that the block containing the transaction will be orphaned. Furthermore, this means that each block cannot be considered a single unit — it may be valid with respect to the transactions present in the block, but it can only be considered part of the ledger if it is present in the longest chain. Given the fact that Nakamoto consensus is stochastic affects the design of our client. This means that our solu-

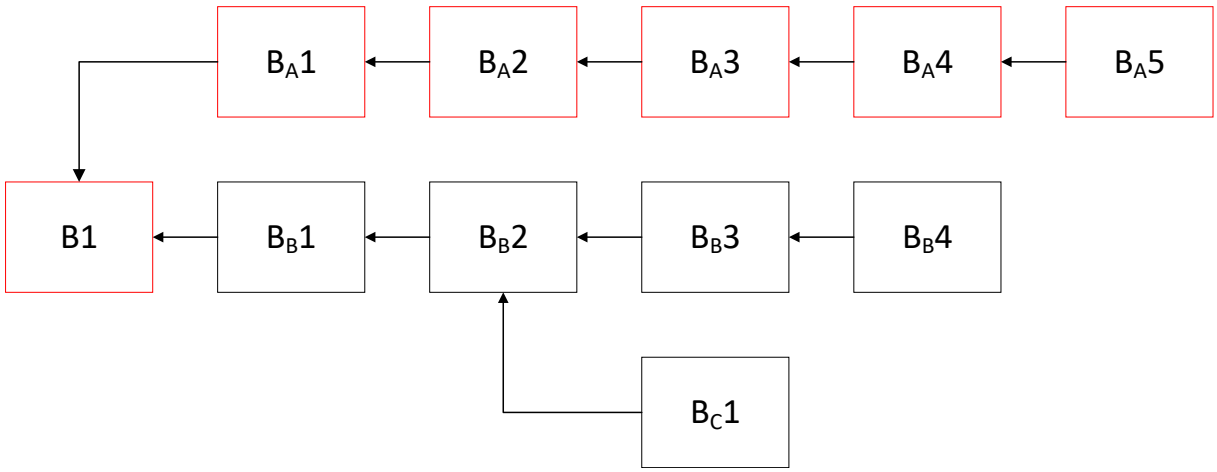

Figure 2.7: In the presence of soft forks, the longest chain is determined as the chain that contains the most blocks and is marked red.

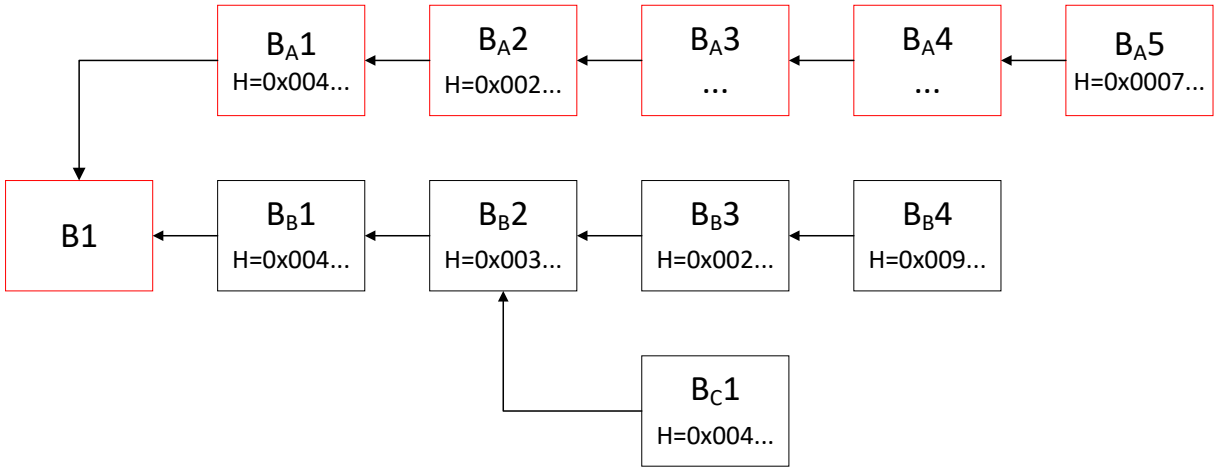

Figure 2.8: In the presence of soft forks, the strongest chain is determined to be the chain containing the most work. *H* is the hash of the block, where hashes that contain more zeros as part of their prefix are harder to create, so more work is spent solving the PoW puzzle. The strongest chain is highlighted in red.

tion must take into account the possible presence of soft forks. A solution to this problem is discussed in Chapter 6.

2.4 Contract layer

A *Smart Contract* is code executed in a distributed DL environment that enables support for custom data storage and arbitrary business logic based on user requirements. Applications developed in this manner are commonly referred to as *Distributed Applications (DApps)*. Code running on DL retains all the DLT properties listed in Chapter 1*.

For example, ETH uses a domain specific, high level, Turing complete language called

*Resistance to censorship, immutability, decentralized maintenance, and the elimination of a trusted third party

Solidity, which is later compiled into a low level, stack based bytecode and executed on top of a virtual machine called *Ethereum Virtual Machine (EVM)* [7, 37]. BTC does not provide a Turing complete language, but instead provides a restricted stack based language called *Script* [37]. There are attempts to integrate Turing complete languages on BTC, for example in the form of a solution called Rootstock [38].

Smart Contracts in the ETH network are assigned a unique identifier in the network. In the case of ETH, it is an address deterministically computed from the address of its creator and the number of transactions the creator's address has previously created. The creation of a Smart Contract is, in essence, a transaction. Users in ETH interact with Smart Contracts by sending transactions to the address of the contract. Smart Contracts can interact with each other on ETH by sending a *Message Call* [7]. Message Calls are a consequence of issuing a transaction. In other words, a single unit of activity in ETH is a transaction (see Fig. 2.9).

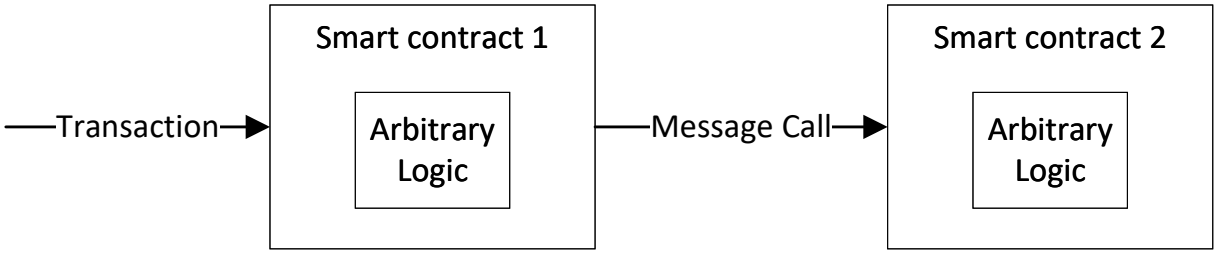

Figure 2.9: Interaction between Smart Contracts.

Smart Contract capability is not a requirement for the implementation of Aurora algorithms, i.e., the Probabilistic honest set construction algorithm, the Transaction history synchronization algorithm and the Transaction presence checking algorithm. Furthermore, since interaction with Smart Contracts is, in essence, done by sending a transaction to the address of the contract, our solution does not have to explicitly account for the existence of Smart Contracts — proof that a Smart Contracts function has been executed is that the corresponding transaction is included in the ledger. However, in this research, they are used to implement a real-world use case of DLT in which our client can be used, which is discussed in Chapter 3.

2.5 Application layer

Perhaps the most popular and well-known application of DLT is the decentralized P2P digital currency BTC, which takes advantage of DLT's key features of security, immutability, transparency, and the ability to cut out the middleman. After the rise of BTC, solutions were developed that allow arbitrary code to be executed over a DL while maintaining the same relevant features. As a result, the technology has the potential to be used in fields other than currency transfer. To showcase the applicability of the technology, we provide an overview of some these fields, focusing on solutions relevant to the *Internet of Things (IoT)* domain within this section.

Public Key Infrastructure. IoT devices often exchange sensitive information. It is critical that such data remain confidential when needed. In addition, it is important that this data is not altered during transmission and that only those with the proper credentials can access it [17]. While PKI that relies on a *Certificate Authority (CA)* is a well-known approach, this approach fundamentally relies on the existence of a trusted third party. As a result, solutions have been developed to remove the CA by using DLT. For example, as part of [39], a solution has been developed where the records of an IoT device are stored in a DL. Such a device can later be associated with a public key.

Domain Name Resolving. The most common purpose of *Domain Name System (DNS)* is to associate human-friendly names with the numeric *Internet Protocol (IP)* addresses that computers need to find services and devices. To this end, a hierarchical naming system is established that relies on the authoritative name servers that serve the DNS root zone[†]. Each time an IoT device resolves an address through these servers, implicit trust is placed in these servers. Solutions have been developed to break such a chain of trust. This involves storing the link between a human-friendly name and the numeric address in a DL [40, 41].

Healthcare. The creators of Vegvisir [42] have developed a solution that creates a tamper-proof log that can be used to log access to sensitive medical records. Access requests to sensitive records are granted on the condition that the request has been recorded in a tamper-proof log. The solution does not proactively prevent access to the data, but by retroactively reviewing the log, unauthorized access can be detected and sanctioned. The solution is particularly useful in situations where the communication infrastructure, such as cell towers, is inoperable, such as during natural disasters. It works in an unreliable network consisting of resource-constrained IoT devices such as smartphones used to build an ad hoc mobile network, and is energy efficient.

Entertainment industry. Admission to an event (e.g., concert, exhibition) is often granted only after proof of appropriate access right, usually in the form of a ticket. Event organizers rely on third-party services [43] to issue and charge tickets, which can be avoided by using DLT. A ticket can be represented as an *Non-Fungible Token (NFT)*, a token that is unique and cannot be divided or merged. A guest at an event can then present a *Quick Response (QR)* code with their smartphone, which is linked to a DL record that proves the guest has a valid ticket.

Data and resource sharing. An IoT device such as a sensor is often used to collect data. This data is then to be shared or sold. By combining DLT and zero-knowledge proofs, the authors of ChainAnchor [44] have developed a blockchain-based architecture to deploy IoT devices and

[†]`https://www.iana.org/domains/root/servers`

sell IoT data. In addition, DLT can be used to incentivize device owners to share resources by rewarding the owner of a device with tokens of value. For example, Helium[‡] is a solution that incentivizes resource-constrained devices to provide network coverage for a reward.

Energy sector. Sensors deployed in a smart home are the cornerstone of smart metering. Automated billing, prepaid solutions, and consumer micro payments can be implemented through the use of DLT [45] to log energy consumption in a DL and to make payments to the service provider through the use of network tokens.

[‡]https://www.helium.com/

Chapter 3

Use-case: supply chain management

The motivation for the proposed solution has already been briefly presented in Section 1.1 from a higher-level perspective. In this chapter, we extend this perspective by presenting a solution developed as part of our research. The solution provides a decentralized, privacy-preserving and verifiable management of a product during its lifecycle through the supply chain: *DL-Tags (DL-T)*[*] [1], in the context of which our solution is very applicable.

For completeness, we first give a brief overview of existing DLT solutions for supply chain management and compare them with DL-T. DL-T focuses on the use of DLT in supply chain management in a trustless environment. It enables tracking of products and allows all stakeholders in a product lifecycle to validate a product they are handling and exchanging. A solution proposed by Tian [46] is similar to DL-T in that it recognizes the problem of centralized traceability systems. It implements a real-time traceability system for the food supply chain based on critical control points and hazard analysis using blockchain and IoT. The solution relies on IoT technologies such as *Radio Frequency Identification (RFID)* for data collection and stores product-related data in BigchainDB [47], a private decentralized data store. This distinguishes it from DL-T, which is explicitly designed for a public, permissionless, and trustless ledger. An innovative solution that proposes and implements a blockchain-based protocol for supply chains is presented by OriginTrail [48]. DL-T solves a subset of the problems covered by OriginTrail. However, OriginTrail relies on a custom token economy that introduces further complexity into the developed system, while DL-T does not rely on a custom token and therefore consists of simpler procedures compared to OriginTrail. Another relevant solution that tracks medical products and their ambient temperature is presented in [49]. Unlike DL-T, the proposed architecture is not DLT-agnostic and does not focus on tracking product exchange, but tracks temperature variations in the products' environment.

[*]Not to be confused with DLT

3.1 DL-Tags in the context of Aurora

DL-T is designed to extend an existing system for tracking digital assets (products) through their supply chain lifecycle, called *TagItSmart (TIS)*[†]. TIS is an IoT solution for supply chain management based on the use of *Smart Tags*. Smart Tags are markers used to track digital products in the TIS ecosystem. Typically, they are issued in the form of QR codes that are printed with special ink and as such have enhanced capabilities compared to regular QR tags. For example, a Smart Tag can change its content depending on environmental conditions such as temperature. A high level overview of the architecture is displayed in Fig. 3.1. The product is exchanged between the stakeholders in the system. Every time the owner of product changes, the state of the product (as it is written on the Smart Tag) is stored on the ledger, while the identity of the stakeholders is never revealed.

Entities: DL-T is designed to scale to any number of potential entities, but in what follows we assume three particular entities. First, TIS, the creator of Smart Tags. Second, the *producer*, the creator of a product to which a Smart Tag is assigned. Third, the *E-commerce store* of a product who distributes the product. There is also a special entity, namely the *consumer*. A consumer consumes a product and is not required to write to the DL or even have an DL account. This was a deliberate design decision to remove the barrier to entry for end consumers, i.e., product buyers. Since a consumer does not change the state of a DL associated with a bought product, this entity was excluded from the list of stakeholders.

Case study: The case study implemented in [1] revolves around the creation, distribution and consumption of wine bottles. A producer (a winery) interacts with the TIS system to create a digital product (a bottle of wine) that is assigned to a unique Smart Tag. The product is passed to a E-commerce store (a web store). The E-commerce store sells the product to an end user (a consumer of the bottle of wine). The state of the product is stored and updated via an ETH DApp, and is publicly readable. The end consumer should be able to interact with the ledger to read its state (read the state of a wine bottle from a Smart Tag QR code and compare it with the digital representation of the wine bottle stored in an ETH Smart Contract) using a resource constrained device (e.g, a mobile phone).[‡]

DL-T improved the TIS ecosystem by making the system less dependent on a central point of failure, namely the TIS platform [1]. More precisely, through the use of Smart Contracts DL-T solves the following relevant issues identified in the TIS ecosystem:

1.Modification of data contained within existing Smart Tags.

[†]https://tagitsmart.eu/

[‡]A video description of the case study which was awarded the IEEE Access Best Multimedia Award is available online: `https://www.youtube.com/watch?v=JCC98iMCPOs`

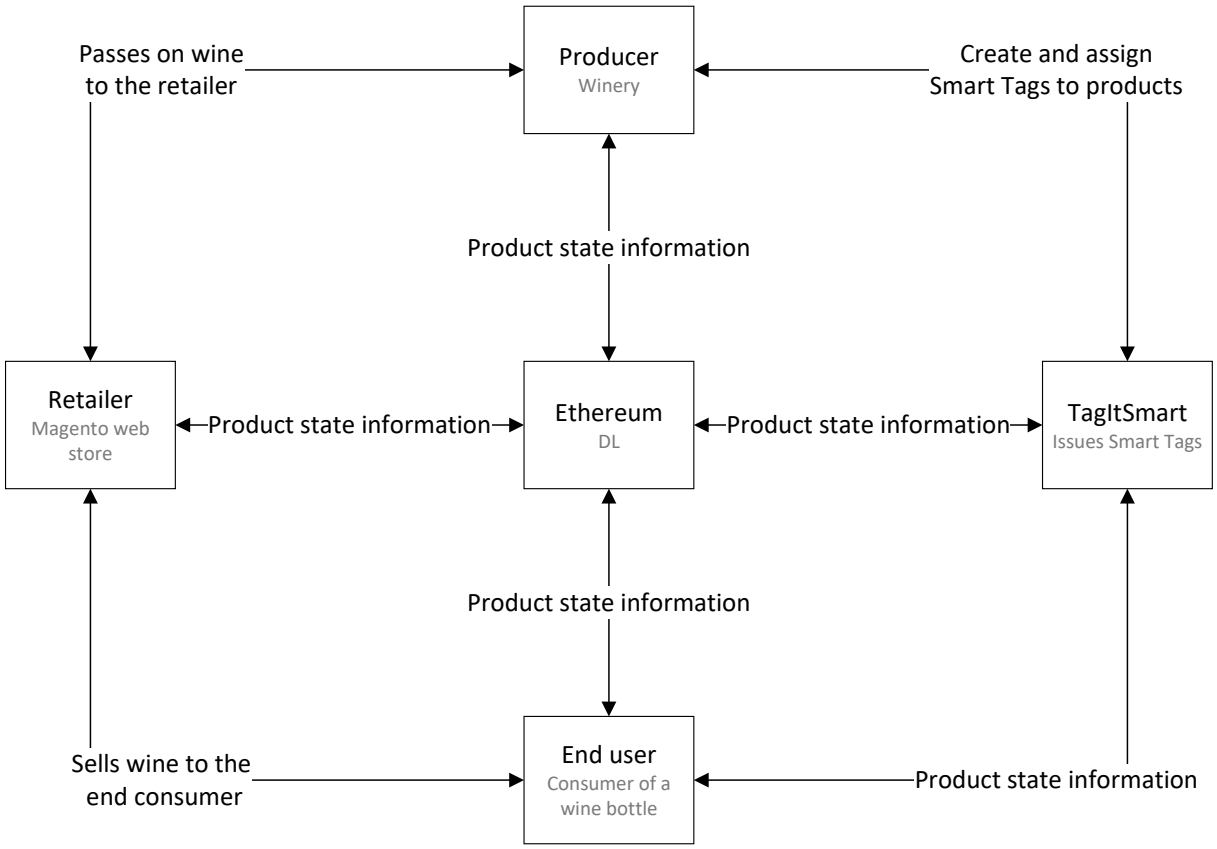

Figure 3.1: Overview of the DL-T architecture.

2.Multiple uses of existing Smart Tags in ways not intended by the ecosystem, or duplication of those tags.

3.Unclear chain of responsibility in case of change of ownership of a product (e.g., if a product was never delivered to the customer, but the customer paid for the product).

4.Circumventing the entire system by creating Smart Tags which were not issued by TIS and contain fraudulent data.

System Requirements: The creation of Smart Tags by TIS must be logged on the DL and such entries must be immutable. Even though ETH is used in the case study, the solution must be able to work with other DL implementations. In other words: ETH should be easily interchangeable, requiring only minor changes to the solution. Each stakeholder should disown a product after passing it to the next stakeholder. Each time ownership of a product passes from one stakeholder to another, the new stakeholder must confirm its authenticity and record this assertion in the DL. Each stakeholder must have the ability to decline to receive information about a product after that product has been passed down the supply chain. All DL records related to a product owned by a stakeholder must be available and transparent to that stakeholder. The end state of a product must be available to all stakeholders and the consumer.

To showcase the utility of Aurora, we review relevant features of DL-T[§] to answer the following two research questions:

Research Question 1. Where in the DL-T ecosystem does a stakeholder initiate ledger synchronization (THS) and could face Problem 1?

For every stakeholder other than the consumer, DL-T has provided a *DL-T proxy* for DL agnostic communication, as shown in Fig. 3.2. It is assumed that the DL DApp is accessed via a full node.

Research Question Answer 1. [Answers RQ. 1] When using a proxy, a stakeholder should run a full node and could face Problem 1.

Research Question 2. Where in the DL-T ecosystem does a stakeholder need to check the presence of a transaction and could face Problem 2?

The possible occurrence of Problem 2 is more nuanced. It can happen mainly because there is no way to enforce any stakeholder to run a full DL node, and a stakeholder might decide to run a light client instead. As a consequence, the use of DL-T proxy becomes centralized since it relies on a light client. The red colored arrows (i.e., *web3js*) in Fig. 3.2 mark the communication flow going through a central entity if a retailer decides not to run a full DL node.

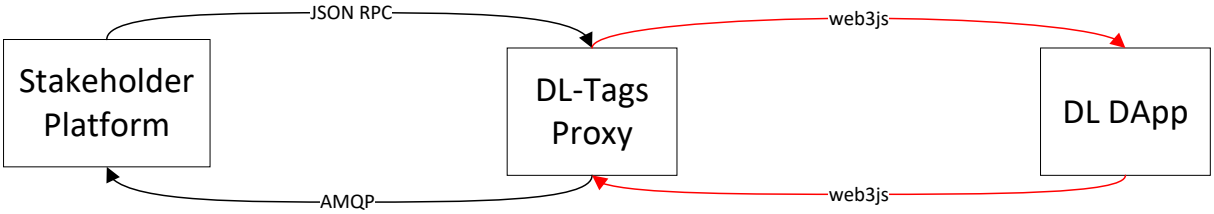

Figure 3.2: The DL-T proxy abstracting a concrete DL implementation from a platform run by a stakeholder.

We continue our analysis of DL-T and the presence of Problem 2 using a bottom-up approach. First, we define a set a possible actions that an entity can preform to interact with the DL-T system, which are:

1.Creation of a product.

2.Creation of a stakeholder.

3.Handover of a product.

4.Voting about the state of a product.

5.Checking the state of a product.

[§]Cost evaluation has intentionally been left out as it does not relate to the focus of this work. For a fuller account of the solution we refer the reader to [1]

3.1.1 Action 1: Creation of a product

We define the action of creating a product as a sequence of messages displayed in Fig. 3.3. A stakeholder (e.g., a producer) sends to the DL-T proxy arbitrary text containing the product description (*productDescription*) and the identification of the stakeholder creating the product (*stakeholder*), which in this case is the producer (messages 1 and 2). The request is forwarded to the DL-T proxy, which belongs to TIS, and then to the TIS platform (message 3). TIS assigns an ID to the product (*productItemID*) and returns it to the DL-T proxy (message 4). At this point, the TIS DL-T proxy has all the metadata necessary to create a product representation on the DL. On the DL side, TIS (*owner*), the producer (*stakeholder*) and the digest of the product identifier (*hash(productItemID)*) are recorded (messages 5, 6 and 7). Since a considerable amount of time may pass before a transaction has been recorded in the ledger, such a request is first registered (messages 5, 6) and the response with the transaction metadata is returned at a later time (message 7).

Once a stakeholder has been assigned to a particular product, an event is triggered to all stakeholders subscribed to the *stakeholderAdded* topic (messages 8 and 9), which is this case only the producer. The event contains the identifier of the new stakeholder added to the product and the digest of the product identifier (*hash(productItemID)*). In this way, the producer learns that a new product has been created. Finally, ownership of the product is passed from TIS to the producer and an event is fired to all stakeholders subscribed to the *ownershipTransferred* topic (messages 11 and 12) containing the identifier of the new owner added to the product and the digest of the product identifier (*hash(productItemID)*). In this way, the producer learns that a new product has changed ownership. If a malicious node replies with these messages, it might reply, for example, that a stakeholder was not added when in fact a stakeholder was added, or that ownership was transferred when it was not transferred.

Research Question Answer 2. [Answers RQ. 2] Messages colored red, namely 7, 8 and 11, and as a consequence 9 and 12 are essentially read operations which confirm that a certain transaction has been included in a ledger and mark the possible manifestation of Problem 2.

3.1.2 Action 2: Creation of a stakeholder

We define the action of assigning a stakeholder to an existing product as a sequence of messages displayed in Fig. 3.4. In this example, an E-commerce store has been added to a particular product by a producer. Similar to Section 3.1.1, an event is triggered to all stakeholders subscribed to the *stakeholderAdded* topic (messages 3, 4, 6 and 7) containing the identifier of the new stakeholder added to the product and the digest of the product identifier (*hash(productItemID)*).

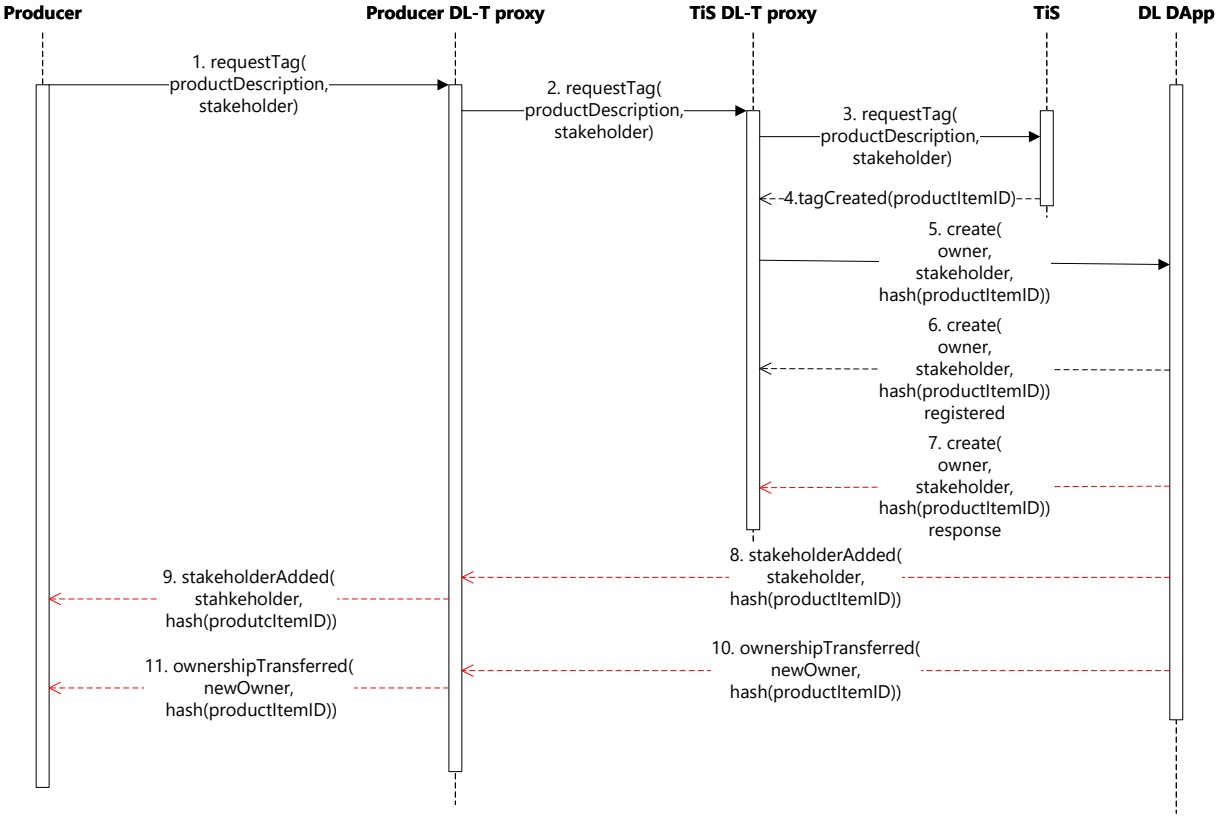

Figure 3.3: Sequence diagram depicting the creation of a product in DL-T.

Research Question Answer 3. [Answers RQ. 2] Messages colored red, namely 3 and 6, and as a consequence 4 and 7 mark the possible manifestation of Problem 2.

3.1.3 Action 3: Handover of a product

We define the action of handing over a product from one stakeholder to another as a sequence of messages displayed in Fig. 3.5. In this example, a producer has transferred ownership to a new stakeholder (*newOwner*). Similar to Section 3.1.1, an event is fired to all stakeholders subscribed to the *ownershipTransferred* topic, which contains the identifier of the new owner of the product and the digest of the product identifier (*hash(productItemID)*). Note that the new owner and event emission have been omitted for simplicity in Fig. 3.5. These messages are analogous to messages 11 and 12 in Fig. 3.3.

Research Question Answer 4. [Answers RQ. 2] The messages colored in red in Fig. 3.5, which are 4 and as a consequence 5, mark the possible manifestation of Problem 2.

3.1.4 Action 4: Voting about the state of a product

We define the action of voting on the state of a product as a sequence of messages displayed in Fig. 3.6. In this example, a producer and an E-commerce store cast their votes on the state of

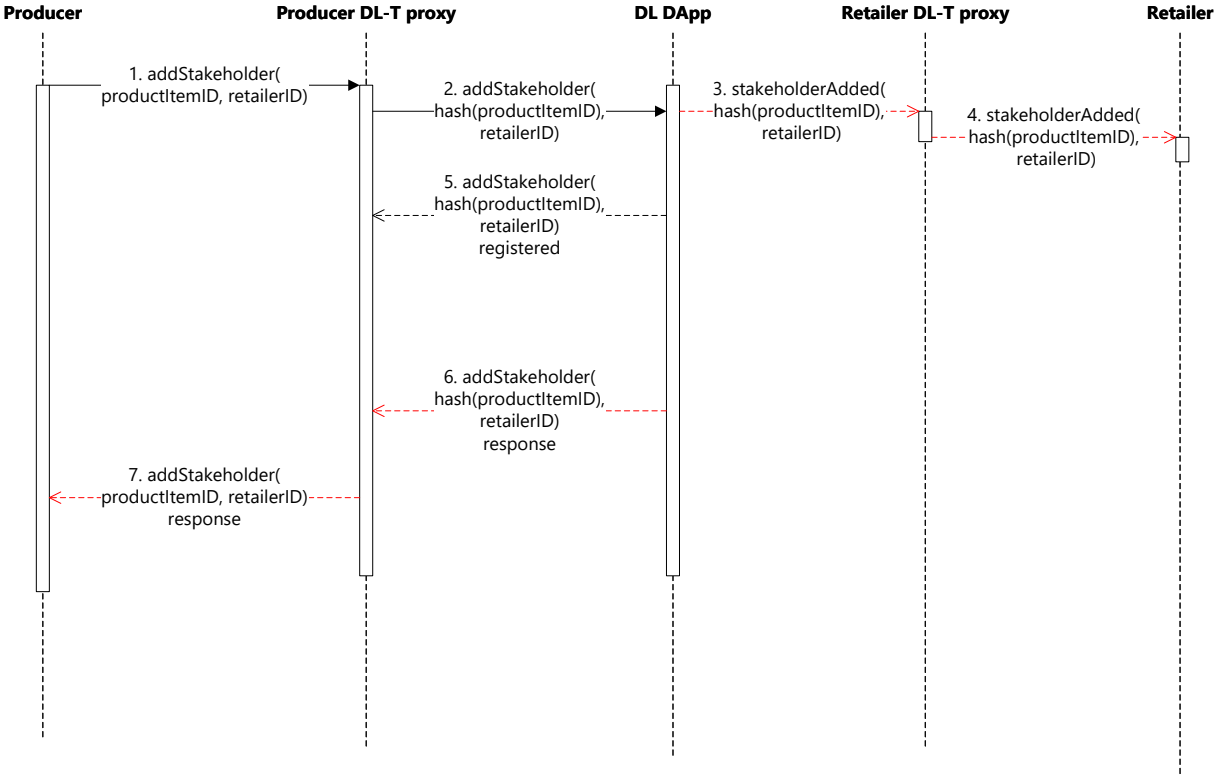

Figure 3.4: Sequence diagram depicting the creation of a stakeholder in DL-T.

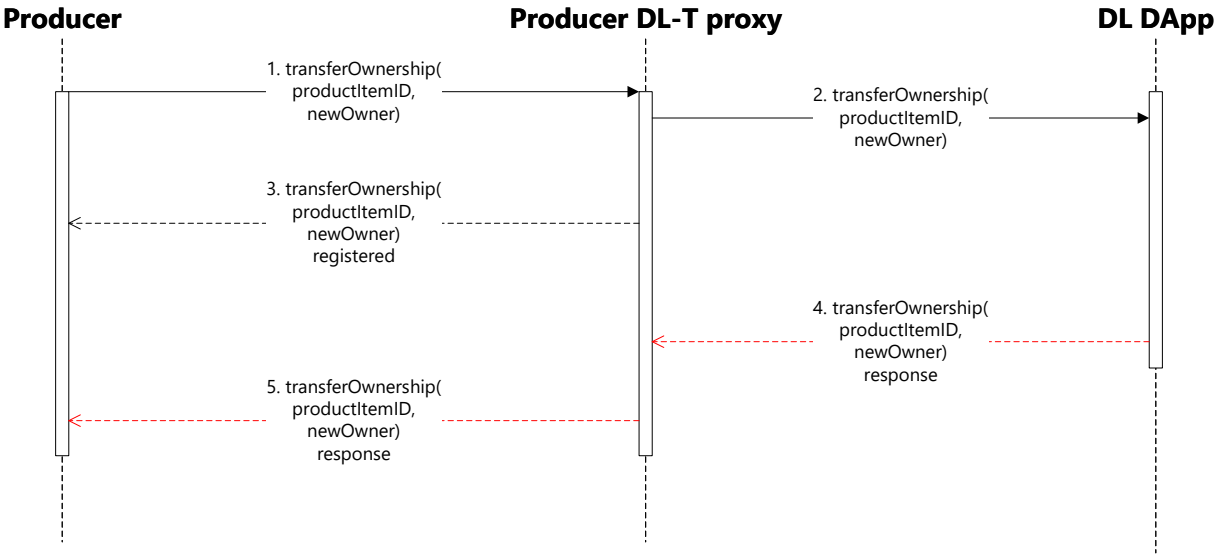

Figure 3.5: Sequence diagram depicting the handover of a product in DL-T.

a product. A vote by the E-commerce store in this example is displayed by messages 1, 4, 5, 10, 11 and 12. A vote by the producer is displayed by messages 2, 3, 6, 7, 8 and 9. A vote contains a product identifier (*hash(productItemID)*) and arbitrary text containing the product description (*productDescription*) (messages 1 and 2). Each stakeholder is assumed to have obtained the arbitrary text containing product description by inspecting the product during procurement in the supply chain. In the case of DL-T, the product description is obtained by scanning a Smart Tag contained on the product itself. The *productDescription* is then hashed by the respective

DL-T proxies and the request is forwarded to the DL (messages 3 and 4). Notification of the registration of the request is sent back to the respective stakeholders (messages 5 and 6). Once a vote is cast on the DL, the response is forwarded to the stakeholders (messages 7 and 9 and 10 and 11). Each time a vote is cast, the Smart Contract that tracks the consensus state of a product updates its internal state (messages 8 and 12). The consensus on a product is designed to be unilateral, i.e., a product is considered valid only if all parties have cast their votes and all parties have submitted the exact same *productDescription*. Any other state of a product is considered not valid. Checking the state and validity of a product is discussed in more detail in Section 3.1.5.

Research Question Answer 5. [Answers RQ. 2] The red colored messages in Fig. 3.6, which are 7 and 10, and as a consequence 9 and 11, mark the possible manifestation of Problem 2.

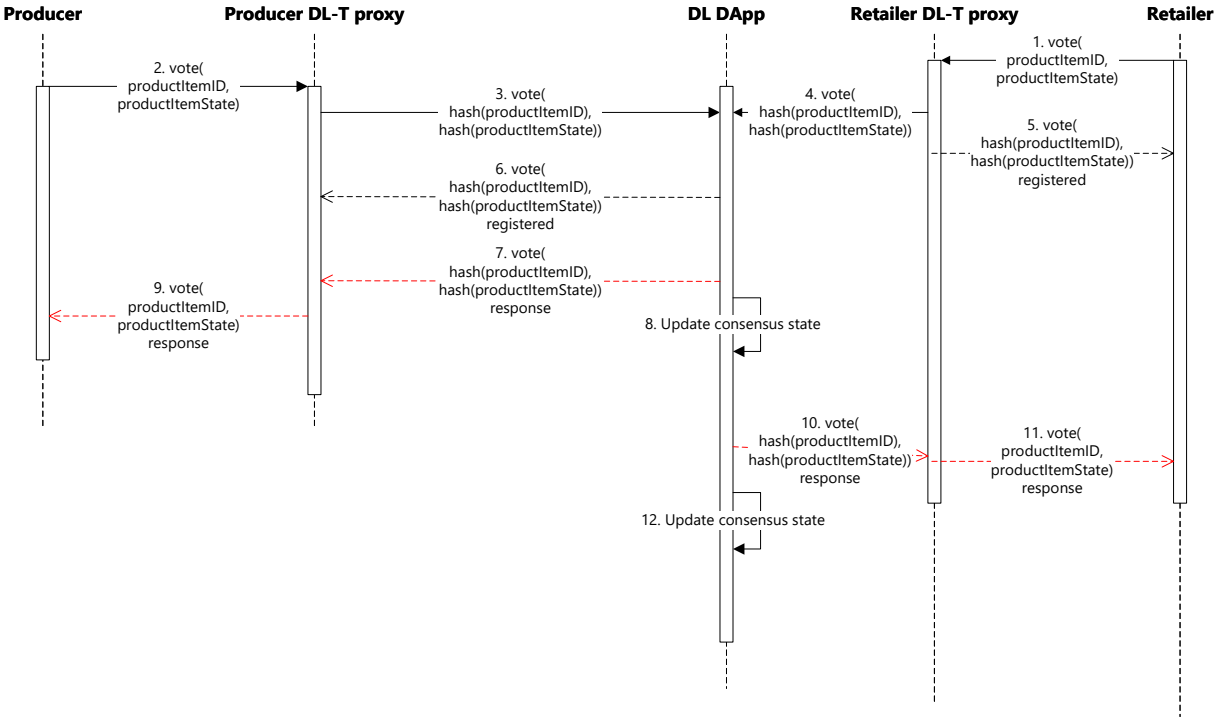

Figure 3.6: Sequence diagram depicting the voting about the state of a product in DL-T.

3.1.5 Action 5: Checking the state of a product

We define the action that checks the state of a product as a sequence of messages displayed in Fig. 3.7. In this example, a producer checks the validity of a particular product. The producer queries the DL-T proxy for the state of a product (message 1). The DL-T proxy hashes the product identifier (*hash(productItemID)*) and arbitrary text containing the product description (*productDescription*) (message 2) and the registration of the request is sent to the producer. The response is sent back to the producer at a later time (messages 4 and 5). Note that these

messages involve only read operations on the DL, which are much faster compared to write operations and do not incur transaction charges.

Research Question Answer 6. [Answers RQ. 2] The red colored messages in Fig. 3.7, which are 4 and as a consequence 5, mark the possible manifestation of Problem 2.

As mentioned in Section 3.1.4, a product is valid only if all parties have cast their votes and all parties have submitted exactly the same *productDescription*. However, this means that the state of the product has not changed at the time of the vote, and says nothing about the current state of a product. For this reason, any validation must include the product identifier (*hash(productItemID)*) and arbitrary text containing the product description (*productDescription*) to verify that the current state of the product has not changed since consensus was reached. For example, this may be the case if a product (a bottle of wine) was stored in an unfavorable condition (the storage room was too warm). Since a Smart Tag is environmentally aware, it will change its contents and the product will be classified as invalid.

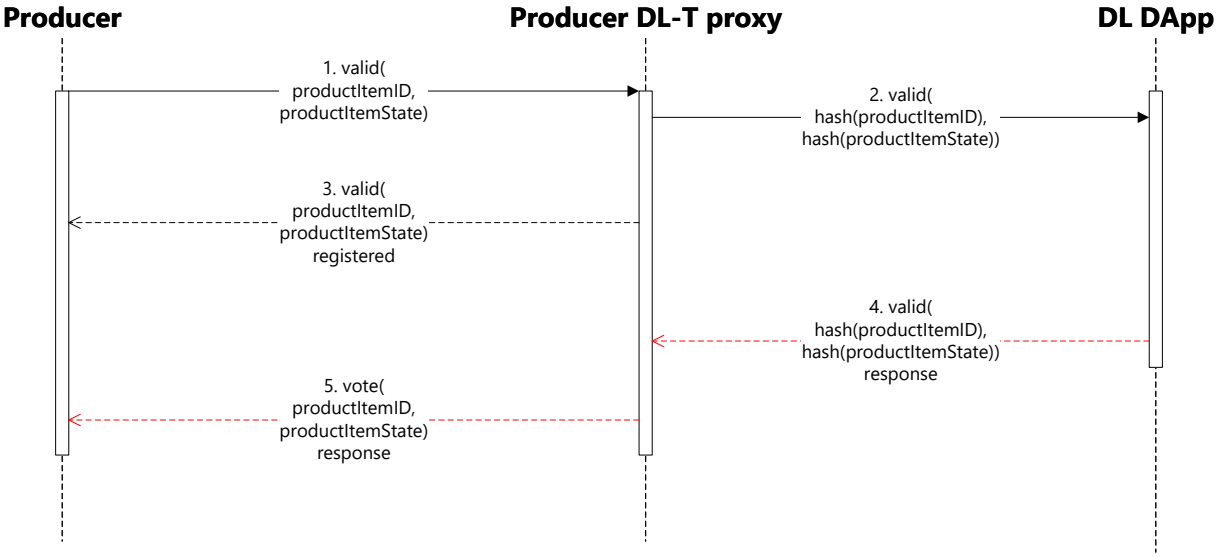

Figure 3.7: Sequence diagram depicting the validity check of a product in DL-T.

After reviewing all possible actions a stakeholder can take, we can conclude that a stakeholder faces either Problem 1 or Problem 2. There is one entity that has not been reviewed yet, which is the consumer. The consumer does not have a DL-T proxy and it is assumed that the consumer does not run a full node and as a result will never face Problem 1. Instead, an average user checks the state of a product with a resource-constrained device (e.g., a smartphone) while inspecting a product in a shop or during home delivery, meaning the consumer can face Problem 2. We continue by looking at DL-T from a higher level, taking a closer look at the issues solved by DL-T, focusing on the consumer.

3.1.6 Issue 1: Smart Tag tampering

A fraudulent party may attempt to modify the contents of a Smart Tag. Such a modified tag may lead a user to a phishing site where fraudulent information about a product may be displayed to the consumer. A fake and cheaper product can be presented as original and more expensive. Fig. 3.8 shows how DL-T solves the problem.

Research Question Answer 7. [Answers RQ. 2] The consumer must compare the data on the Smart Tag with the data stored on the DL to detect data tampering. This means that the consumer needs to confirm that a transaction that has written the data exists in the ledger, and therefore the consumer could face Problem 2.

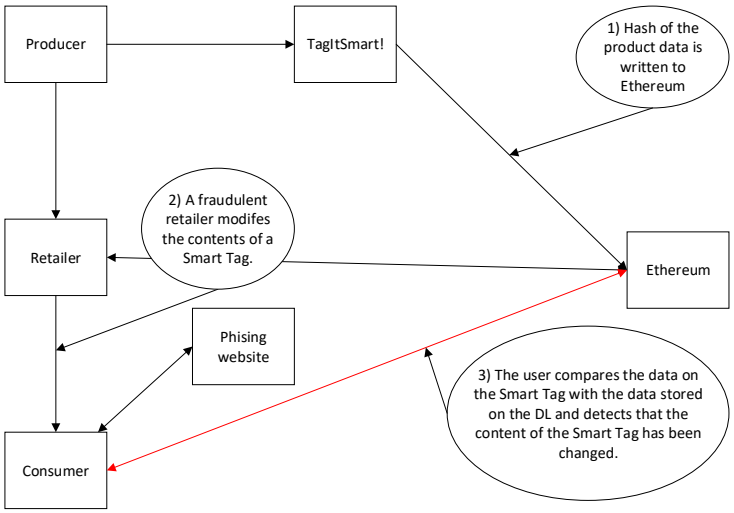

Figure 3.8: Communication flow in the DL-T system when a user detects that a Smart Tag has been tampered with. The red colored arrow marks the communication flow going through a central entity.

3.1.7 Issue 2: Smart Tag duplication

A fraudulent party may attempt to duplicate the contents of a Smart Tag. Unlike the tampering of a Smart Tag, a duplicated Smart Tag contains the data of an authentic tag issued by TIS. In this way, and similarly to the situation where a Smart Tag has been tampered, fake and cheaper products can be presented as original and more expensive ones. Fig. 3.9 shows how DL-T solves the problem.

Research Question Answer 8. [Answers RQ. 2] The consumer must confirm that a product has been disowned by the retailer, which means that the consumer needs to confirm that a transaction that has disowned that product exists in the ledger, and therefore the consumer could face Problem 2.

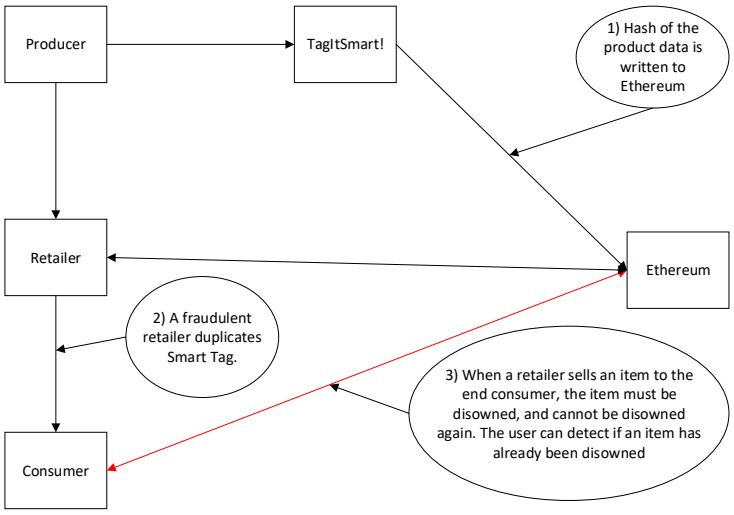

Figure 3.9: Communication flow in the DL-T system when a user detects that a Smart Tag has been duplicated. The red colored arrow marks the communication flow going through a central entity.

3.1.8 Issue 3: Chain of responsibility

Aside from the consumer of a product, there are situations where a E-commerce store may want to verify the validity of a product. For example, there may be multiple E-commerce stores in the product life cycle as shown in Fig. 3.10.

Research Question Answer 9. [Answers RQ. 2] The E-commerce store can decide to check the state of a product (see Section 3.1.5). This action requires that the state of a product has been recorded in the ledger in the scope of a transaction, therefore, a E-commerce store could face Problem 2.

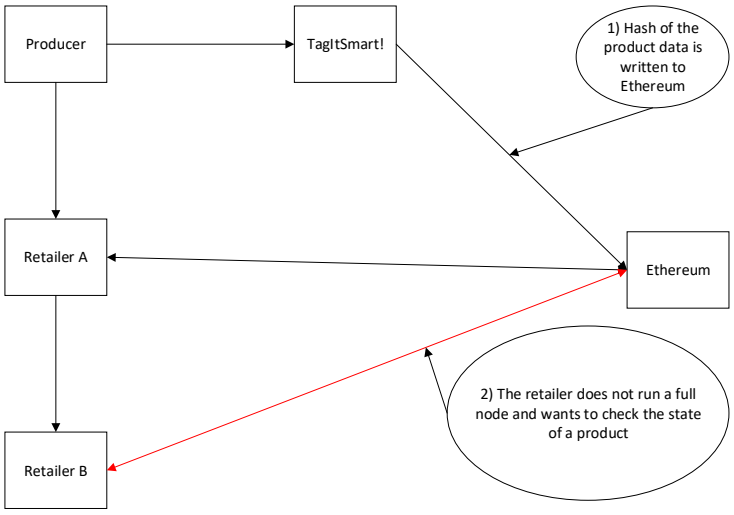

Figure 3.10: Communication flow in the DL-T system when a E-commerce store wants to check a product's state. The red colored arrow marks the communication flow going through a central entity should a E-commerce store decide not to run a full node.

3.1.9 Issue 4: Circumvention of the system

A fraudulent party may attempt to create Smart Tags that direct a user to a phishing site, similar to the scenario displayed in Section 3.1.6. In addition, the fraudulent party creates a dedicated DApp that is intended to mimic the behavior of DL-T but falsely certifies a product state. Such a fraudulent DApp can be easily identified because DLT largely depends on PKI, so the user can tell when an DApp is not authentic (i.e., issued by TIS), as displayed in Fig. 3.11.

Research Question Answer 10. [Answers RQ. 2] By the use of PKI, the consumer can detect that a transaction is included in the leger but has been issued by a fraudulent DApp and could face Problem 2.

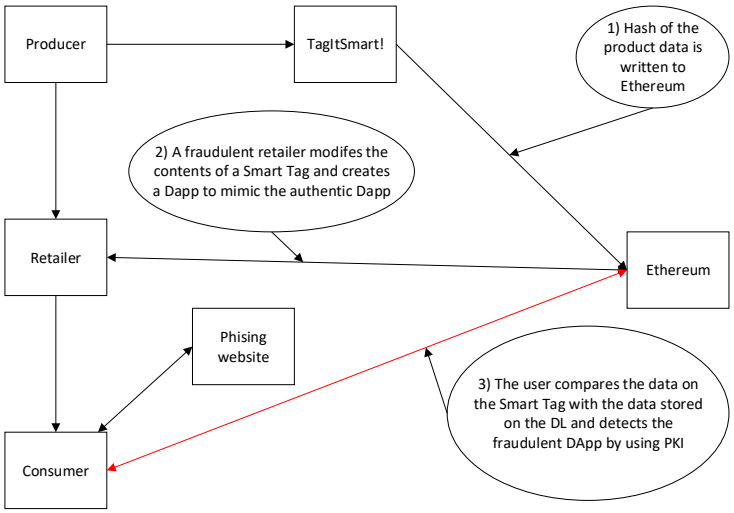

Figure 3.11: Communication flow in the DL-T system when a producer creates a fraudulent DApp and a consumer wants to check the state of a product. The red colored arrow marks the communication flow going through a central entity.

In summary, the aftermath of the implementation of DL-T clearly demonstrated that DLT in its current state is in dire need of solutions that lower the barrier to entry for new users, which do not have hardware capable of running a full node, or want to use a light client, in a way that retains the property of trustlessness. Note that DL-T is by no means an isolated case study where Problem 1 or Problem 2 may occur. Another rudimentary example would be a user checking their token balance after participating in an *Initial Coin Offering (ICO)* or an NFT sale. Both ICOs and NFTs on ETH are essentially Smart Contracts with a map structure that has user addresses as keys and token balances as values [50]. The users, who verify their balances essentially perform a TPC, so this transaction confirms that tokens have been transferred from the creator of the tokens to the benefactor, and could face Problem 2. All these facts motivated the development of our solution.

Chapter 4

Aurora

Having laid out the exact problems we solve (Problem 1 and Problem 2 in Chapter 1) and highlighted the possible occurrences of these problems in an existing DL solution in Chapter 3, we proceed to explain how our solution works. Specifically, we give an overview of related work, a detailed theoretical analysis of our solution, an evaluation of our solution using open source simulation tools and the evaluation of our solution on an open source ETH client. We have named our solution Aurora, which is a set of three stochastic algorithms: the Probabilistic honest set construction algorithm (see Algorithm 2), the Transaction history synchronization algorithm (see Algorithm 4) and the Transaction presence checking algorithm (see Algorithm 5). A runtime able to execute Algorithm 2, Algorithm 4 and Algorithm 5 is called an *Aurora module*, and a DLT node enhanced with the Aurora module is called an Aurora node. We start by comparing our solution with other existing work.

4.1 Related work

4.1.1 DL clients

Existing DL clients can broadly be classified in three distinct categories, based on their trust model (i.e., the amount of trust assumptions that have to be made when using such clients) and te amount of required storage and RAM, which are *Remote Clients*, *Simplified Payment Verification clients (SPVs)* and *Trustless clients*:

Remote clients rely on a centralized solution for communication with a DLT network. An example is MetaMask, an Ethereum wallet that communicates with the Ethereum ledger via the Infura Gateway System [51]. Remote clients which are not completely centralized also exist, for example the SlockIt INCUBED client*. This client uses multiple gateways to communicate with a ledger network. The gateways have a stake in the network that can be reduced if their

*https://consensys.net/diligence/audits/2019/09/slock.it-incubed3/

misbehavior is discovered. Remote clients sacrifice the integrity of the data by trusting a third party in exchange for a very small computational resource requirement to interact with a DLT network. They differ greatly from our solution because our solution is completely decentralized.

Simplified Payment Verification clients (SPVs) are standard distributed ledger clients that synchronize with the network's header chain and request the rest of the block information as needed. Examples are Electrum[†] for Bitcoin, or Geth (in light mode) for Ethereum. There are SPVs that, similar to our solution, recognize the added value of multiple sources of truth and can even be used in conjunction with our solution. For example, Electrum maintains connections to approximately 10 servers and subscribes to block header notifications to all of them to detect forks and partitions. Here we see an emerging synergy: The Probabilistic honest set construction algorithm can be used to identify 10 servers which are honest with high probability. The Tendermint [52] light client acquires prerequisites for connecting to the network by means outside of Tendermint[‡] (e.g., social consensus), and thus it can apply our solution to identify these prerequisites. SPVs differ from remote clients since they maintain their trustlessness to some extent by validating metadata against the header chain in their possession. However, since they are served by full nodes, they rely on full nodes which are assumed to be available and honest. Moreover, the applicability of SPVs to resource-constrained devices is debatable — even the header chain may be too large for some devices. Our solution differs greatly from SPVs since it does not require the header chain to work.

Trustless clients attempt to maintain the trustless mechanisms of SPVs while keeping their size sub-linear or constant compared to ledger size or even header chain size. As such, they are sometimes referred to as *super light clients*. A client enhanced with the Aurora module would partially fall into this category — when used for TPC, the client does not need to download the header chain (or part of the header chain). A proposal from [53] uses a cryptography accumulator and generates a chain summary in the form of a block attribute. Then, the client must randomly choose a slice of the network nodes, and if the majority of nodes is compromised by the attacker, the protocol is not secure. Consequently, our algorithm can be used in conjunction with the above — the slice can be the output of our algorithm. Vault [54] is a solution built on a proof-of-stake consensus protocol proposed by Algorand [55] and introduces a fast bootstrapping method relying on the presence of stamping certificates, while our solution does not require any additional data structures within a DL to operate.

We continue by highlighting two solutions: FlyClient [56] as an example of a solution that would benefit significantly from Aurora, and BlockQuick [57] as an example of a solution that addresses the potential existence of Eclipse attacks, but does so in a fundamentally different way than Aurora.

[†]https://electrum.readthedocs.io/en/latest/faq.html
[‡]https://docs.tendermint.com/master/spec/p2p/node.html

FlyClient is a light client proposal for Proof of Work blockchains [56]. The client only needs to store a logarithmic number of block headers to provide strong mechanisms for ensuring the validity of those block headers by using techniques such as probabilistic sampling, MRRs[§], and the Fiat-Shamir heuristic. As FlyClient requires nodes to maintain MRRs, FlyClient cannot be used on the currently running Bitcoin and Ethereum networks without forks. Also, the client must be connected to at least one honest node, which means that FlyClient cannot protect a node against Eclipse attacks. The one honest node that FlyClient needs to operate can be found with our solution. Thus, our client does not compete with this solution, but can work in synergy with it.

BlockQuick is based on a consensus-based reputation scheme [57]. A BlockQuick client maintains the Consensus Reputation Table, which contains miner addresses with the highest reputation shares generated based on the latest 100 block headers. The reputation share of a miner is equal to the number of blocks mined in the last 100 blocks. When new blocks are broadcasted, the client validates the miner's cryptographic signature against the data from the Consensus Reputation Table. A new block is considered valid only if the block receives a consensus share score $> 50\%$. Thus, it is not just a matter of choosing the longest chain with the highest difficulty, but accepting the block from miners with a high consensus share. The client is resistant to both MITM and Eclipse attacks. Similar to FlyClient, the requirements for running BlockQuick clients with the current Ethereum and Bitcoin networks are not met. According to [57], each miner should be reachable at the address specified in the block, each block header must contain an address and the public ID of the block miner and each block must contain a proof of inclusion of all previous block headers. The result is that, unlike our solution, a BlockQuick client cannot be deployed on currently running Ethereum or Bitcoin networks without a hard fork.

4.1.2 Countermeasures to adversarial influence

Adversarial influence in P2P networks: Relevant solutions in the P2P domain are anomaly detection methods and sibyl group inference solutions (e.g., SybilGuard [58]). However, since reputation measurements are not available in existing blockchain solutions, such approaches are not applicable in this context. Anomaly detection techniques [59], flow-based and graph-based methods [60, 61] as well as network traffic reliant solutions (e.g., [62, 63]) differ from our solution in two key aspects: The first and most obvious is the application domain — our work is DLT-specific. Second, we do not deal with or infer the identity of malicious clusters, but focus on identifying honest nodes for persistent or transient communication. Furthermore, the notion of Fake bootstrapping is also a widely explored topic and the countermeasures can be classified

[§]Merkle Mountain Ranges

into 6 categories [30]. First, *Random address probing*, where node a may attempt to discover bootnodes by trying to connect to a random IP address and a known port within the address space. This problem is a subset of Problem 1 and a justification for Assumption 2 (since a random first contact node fc can be found using this approach). Second, *Peer-Caches*, where if node a has already discovered other remote node, their addresses are cached and can be reused. This approach is not sufficient to solve Problem 1, since a completely new node has no access to such a cache. Third, *Centralized Bootstrapping*, where dedicated bootnodes are hardcoded in the client. This approach is not sufficient to solve Problem 1 as it is contrary to Assumption 2 (since it assumes that the global bootstrapping network is available). Fourth, *Global Bootstrap Service*, where the bootstrapping services of multiple P2P networks can be merged in a single global bootstrapping network. This solution is also not sufficient to solve Problem 1, as it is contrary to Assumption 2. Fifth, *Out of band mechanisms*, where the address of bootnodes are periodically stored on a third party web cache. Unlike our solution, this approach is not trustlessness (since it depends on a third party web cache). Sixth, *Network Layer Mechanism* allow for the creation of multicast groups for bootstrapping. This approach is contrary to Assumption 2 (since it assumes that the multicast groups are available). Network-level solutions that defend against Eclipse attacks have also been suggested. Proposals by [27, 64, 65] suggest adding more structure to the P2P network, which is not applicable to unstructured networks like BTC. There also exist proposals that are designed for unstructured networks. For example, a misbehaving peer can be blacklisted [66], or a centralized service based on PKI can be used [67]. This is contrary to Assumption 2. A proposal by [68] limits the rate at which remote peers exchange network metadata. Our solutions differs from all of these approaches as it is DLT specific, i.e., these solutions can not directly be applied to resolve Problem 1 or Problem 2.

Adversarial influence in DLT networks: Adversarial influence in DLT networks has already been covered in Section 2.2.1. Prominent DLT solutions recognize these threats. For example, BTC uses cached peers for subsequent connections, employs a pseudo-random protocol to obtain bootnode, uses DNS to discover potential bootnodes, and uses a list of hardcoded bootnodes when the DNS bootnodes are unavailable. Another group of solutions relevant to our problem are solutions that generate fraud proofs [69], but they require adding a fraud proof node to the network, while our solution does not require any kind of specialized node. Solutions that prevent low resource eclipse attacks in DLT networks are also proposed. A proposal for ETH [32] recognizes that it is beneficial to the network to make it difficult for the attacker to generate Sybil nodes from a single IP address, and in response geth limits the number of nodes in the same $/24$ subnet to 10. Such proposals justify Assumption 5, which in turn means that an adversary can spawn a finite number of Sybil nodes (i.e., up to κ nodes). Note that this means that our solution is vulnerable to an attacker that can create an infinite amount (or a sufficiently

large amount) of geographically distributed, coordinated Sybil nodes. In the scope of this thesis, we do not present mechanisms to defend against such an attacker, but note that solutions to this scenario exist. In particular, in [16] the authors propose two distinct countermeasures that can be used to solve Problem 1. First, using of external services to exchange ledger metadata between nodes. Second, analyzing of the header chain and the detection of suspicious block timestamps. The former approach differs from our solution as it relies on third party services for the storage of ledger metadata, while our solution is trustless. The latter approach can be used in a strong synergy with our solution in response to Problem 1, which enables the elimination of Assumption 5 but introduces additional complexity as a consequence (since it requires the analysis of suspicious block timestamps).

4.2 Core functionality — Honest sets

The core purpose of our solution is to discover sets of network nodes containing honest nodes that can be used for transient or persistent communication. We continue by providing definitions for these sets.

Definition 1 (Deterministic Honest Set, Δ_h)**.** For a given Γ, a set Δ_h is a subset of Γ which contains at least h honest nodes.

Definition 2 (Probabilistic Honest Set, Π_h)**.** For a given Γ, a set Π_h is a subset of Γ which contains at least h honest nodes with at least probability ρ. The probability ρ is derived from an underlying hypergeometric distribution.

4.2.1 Honest sets and ledger synchronization

We proceed to explain how does the use of Δ_h and Π_h offer a streamlined THS in the presence of κ malicious nodes in the network (i.e, the solution to Problem 1). Let $h = 1$, and let us define the following pair of sets:

Definition 3 (*Deterministic safe set*, $\Delta_s \equiv \Delta_1$)**.** For a given Γ containing κ malicious nodes, where $|\Gamma| > \kappa$, Δ_s is a Δ_h which contains at least one honest node and its size is at least $\kappa + 1$.

Definition 4 (*Probabilistic safe set*, $\Pi_s \equiv \Pi_1$)**.** For a given Γ containing κ malicious nodes, where $|\Gamma| > \kappa$, Π_s is a Π_h which contains at least one honest node with at least probability ρ.

Nodes in the network advertise the status of the ledger (e.g., the highest total difficulty, the longest chain, etc.). We call a request from the Aurora node to the remote node for the remote's ledger status a status request message. The remote answers with a status response, containing the latest ledger status. The Aurora node will try to synchronize its ledger with the remote node

that advertises the latest status. Here we make a reasonable assumption that an honest node will respond with information that is part of the longest chain (as in Assumption 1). In turn, malicious nodes can advertise any status. However, also per Assumption 1, the following must be true:

Corollary 6.1. If a malicious node advertises a status that is newer (e.g., higher total difficulty, longer chain, etc.) than the one advertised by honest nodes, that status must be fake.

It is worth noting that a node that follows protocol (i.e., is honest) may know nothing about the longest chain and can behave in all respects like a malicious node, which could happen if the node never comes into contact with a node that knows the longest chain. Without loss of generality, we do not distinguish such nodes from malicious nodes.

When an honest node from a Δ_s or Π_s advertises the longest chain, the Aurora node will initiate the THS with that node. In turn, if a malicious node from a Δ_s and Π_s advertises the longest chain, the Aurora node will commence the THS with a malicious node. However, since that node offers a fake state (Corollary 6.1) and the Aurora node will attempt to download and verify the ledger, the Aurora node will eventually detect that the provided status is fake, abort the THS, blacklist the remote node and choose a next remote node that offers the latest status. Eventually, the Aurora node will contact an honest node and will complete the THS. A more formal account of this approach is given in Section 4.3.

Π_s and transaction relay: Note that a Π_s has other potential use cases. For example, it can be used as a guarantee that a transaction submitted by the Aurora node will reach the rest of the network, since the honest node will relay the transaction to the rest of the network whereas a malicious node may decide to drop the transaction.

4.2.2 Honest sets and transaction presence checking

Next, we describe how to use Δ_h and Π_h to check if a transaction is present in a ledger in the presence of κ malicious nodes in the network (i.e, the solution to Problem 2). Let $h \in [\lfloor|\Delta|/2\rfloor+1, \lfloor|\Pi|/2\rfloor+1]$ (i.e., the majority of Δ or the majority of Π), and let us define the following pair of sets:

Definition 5 (*Deterministic progress set*, $\Delta_p \equiv \Delta_{\lfloor|\Delta|/2\rfloor+1}$)**.** For a given Γ containing κ malicious nodes, where $|\Gamma| > 2\kappa$, Δ_p is a Δ_h which contains a majority of honest nodes and its size is at least $2\kappa+1$.

Definition 6 (*Probabilistic progress set*, $\Pi_p \equiv \Pi_{\lfloor|\Pi|/2\rfloor+1}$)**.** For a given Γ containing κ malicious nodes, where $|\Gamma| > 2\kappa$, Π_p is a Π_h which contains a majority of honest nodes with at least probability ρ.

Having a Δ_p or Π_p at its disposal, the Aurora node can ask members of these sets for a Merkle proof that a transaction of interest is present in a block by sending a proof request. A remote node responds to such a request with a proof response. The state of the transaction can then be inferred by majority vote since the majority of nodes in Δ_p and Π_p are honest. A more formal account of this approach is given in Section 4.3.

4.2.3 Probabilistic honest set construction

Probabilistic sets are generated during an iterative process when the Aurora node in each step first contacts a network node, second asks for a set of its neighbors, and third selects a next node from the set of neighbors to repeat the process. For this reason, we define the following:

Definition 7 (Set of discoverable network nodes in step i, Γ_i)**.** Γ_i is a subset of network nodes which the Aurora node has learned about while performing network discovery up to step i, where $|\Gamma_i| \leq |\Gamma_{i+1}|$.

In other words, Γ_i is filled with nodes appearing in neighbor sets (i.e., peer lists) reported by contacted nodes, and this set contains candidates for probabilistic honest sets. Π_h is generated from a finite population Γ_i by sampling random nodes without replacement. The sampling process can be modeled by the hypergeometric distribution [70], as each element selected from Γ_i can be classified as a failure or a success. In our context, a success denotes the selection of an honest node, while a failure occurs when a malicious node is selected. In summary, the distribution models the probabilities associated with the number of successes in a hypergeometric experiment, and is defined by

$$X \sim Hypergeometric(N, K, n), \tag{4.1}$$

where N is the population size, K is the number of successes in the population, and n is the sample size. An example of a hypergeometric distribution is displayed in Fig. 4.1, where: $N = 100$, $K = 70$, $n = 5$. For example, the probability that a single success will be contained in the sample of size 5 is 0.02548.
The underlying probability mass function is then given by

$$P(X = x) = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}} = f(N, K, n, k), \tag{4.2}$$

where k is the number of successes in the sample.

We begin the construction of a probabilistic set by performing network discovery initiated at node fc, as can be seen on Fig. 4.2, where honest members are colored white and malicious members are colored red. In the example shown on Fig. 4.2, the first contact node fc is malicious and exposes only malicious nodes. In another example, the first contact node fc

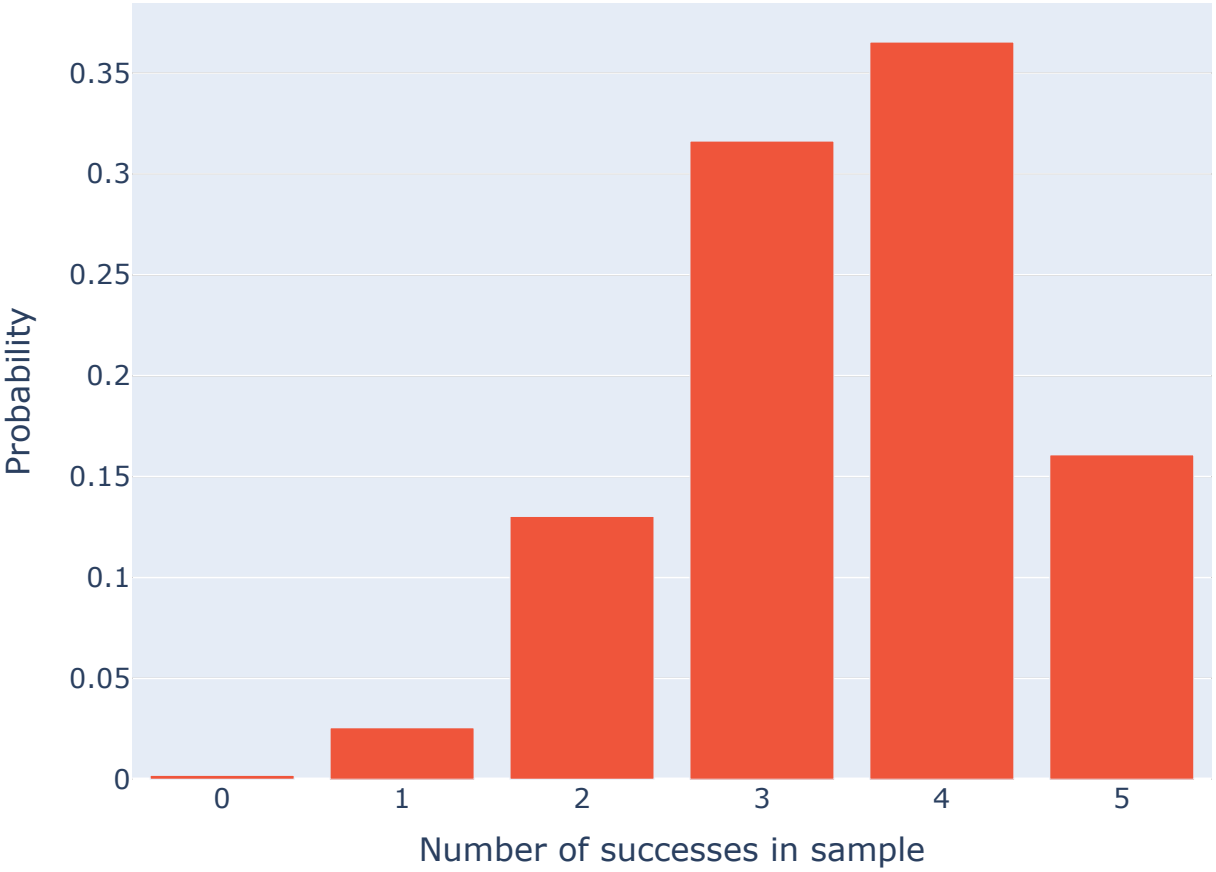

Figure 4.1: An example hypergeometric distribution: $N = 100$, $K = 70$, $n = 5$

could have been honest and would have exposed both malicious and honest nodes (since honest members do not know which members are malicious).

The Aurora node then explores the network until $|\Gamma_i| > \kappa$. As per Assumption 5, an adversary has limited computational resources and can spawn at most κ malicious nodes.

We then start performing hypergeometric experiments for a given Γ_i (the population). It contains $|\Gamma_i| - \kappa$ honest nodes (successes), and there exists a subset of size $|\Pi_h|$ (the sample) which contains h honest nodes with probability ρ or more. More formally, the hypergeometric distribution can be stated as:

$$X \sim Hypergeometric(|\Gamma_i|, |\Gamma_i| - \kappa, |\Pi_h|), \tag{4.3}$$

where we evaluate the following predicate

$$p(X \geq h) \geq \rho. \tag{4.4}$$

If the predicate is evaluated as true, we have constructed a probabilistic set of size $|\Pi_h|$ containing at least h honest nodes with at least probability ρ. If the predicate is evaluated as false, we continue exploring the network.

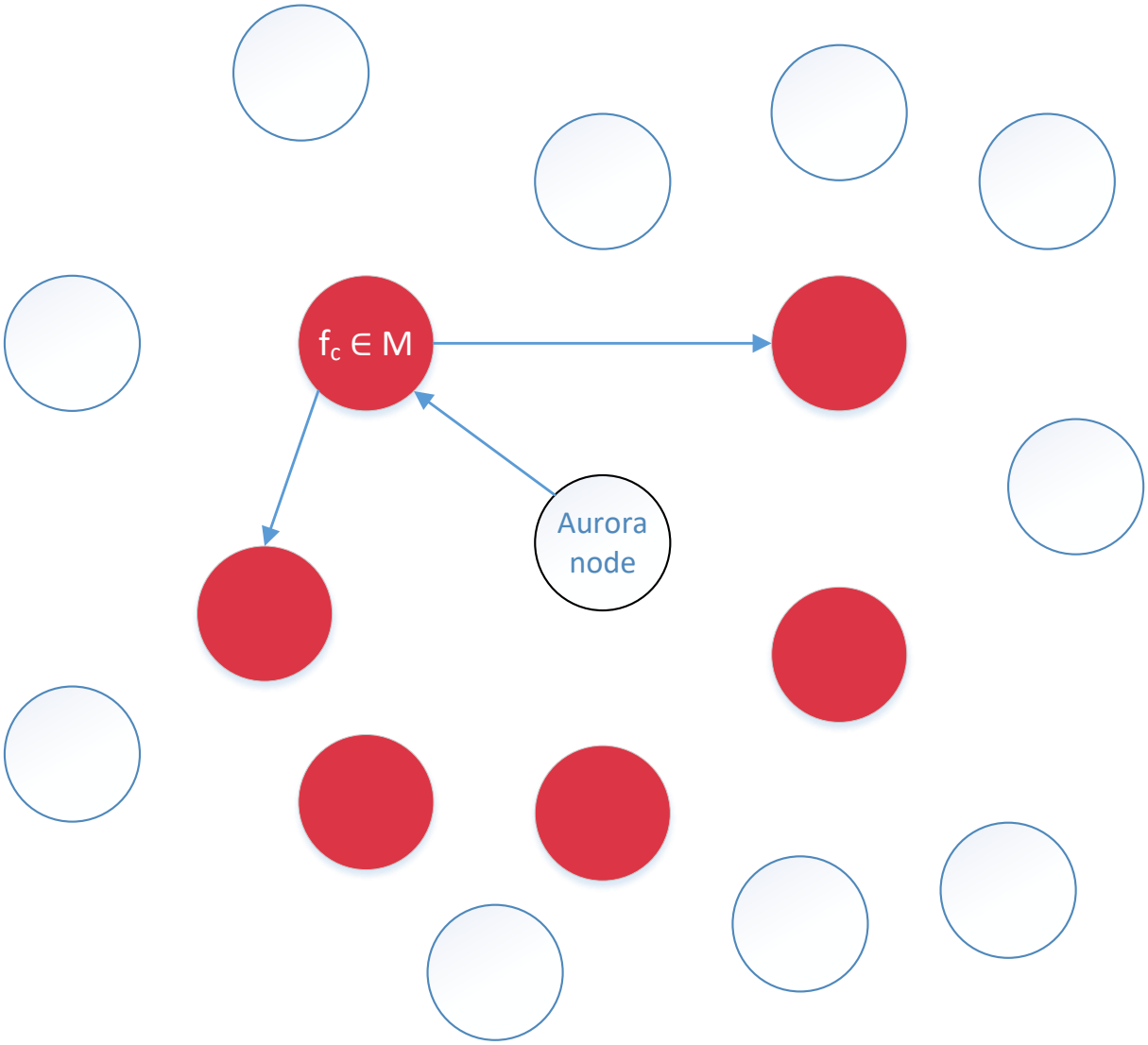

Figure 4.2: The Aurora node enters the network by contacting a malicious fc that exposes only malicious members.

The core pseudocode defining the construction of a probabilistic set is given in Algorithm 1 which requires four input parameters: the first contact node fc, the correctness probability ρ, the desired probabilistic set size $|\Pi_h|$, and desired malicious node tolerance (the maximum number of malicious nodes that can exist (i.e. be tolerated) in a given DL network such that an Δ_h or Π_h can be constructed.) κ, the initialization of which is explained in Section 4.3.

Now that we understand how Π_h is constructed, we can analyze why using probabilistic sets is advantageous compared to using their deterministic counterparts.

We perform an experiment to compare the deterministic and probabilistic honest sets which are the most relevant to be used in the DLT domain, namely safe sets and progress sets. For four different population sizes $|\Gamma|$, we gradually increase the number of malicious nodes κ to compare Δ_s with Π_s, and Δ_p with Π_p. The results presented in Fig. 4.3 lead us to the following conclusions:

Algorithm 1: Core probabilistic honest set construction algorithm

Input : fc — First contact node
Input : ρ — Probability guarantee
Input : $|\Pi_h|$ — Desired probabilistic set size
Input : κ — Desired malicious node tolerance

```
Discover at least κ + 1 nodes in Γ starting from fc;
constructed ← false;
do
    X ~ Hypergeometric(|Γ|, |Γ| − κ, |Π_h|);
    if p(X ≥ h) ≥ ρ then
        constructed ← true;
    else
        Add more nodes to Γ;
    end
while !constructed;
```

- The sizes of probabilistic sets Π_s and Π_p are much smaller compared to the sizes of Δ_s and Δ_p;
- The size of deterministic sets is, as expected, linearly dependant on the number of malicious network nodes, while the size of probabilistic sets shows sub-linear growth. For certain parameters (e.g., when κ is reasonably small), the probabilistic set sizes are significantly smaller compared to their deterministic counterparts. The observed difference is significant for larger populations, when a probabilistic set size can be several orders of magnitude smaller compared to its deterministic counterpart.

Thus, we can conclude that probabilistic honest sets bring significant benefits for solving Problem 1 and Problem 2, especially for large networks.

4.2.4 Probabilistic honest set size reduction method

The conclusions drawn from Fig. 4.3 open a new research question:

Research Question 3. Is there any method by which the size of a given Π_h can be reduced in comparison with the size of the corresponding Δ_h?

Since by Assumption 5, an adversary can spawn up to κ malicious nodes, we propose the following:

Proposition 1. By increasing the number of discoverable nodes $|\Gamma|$, $|\Pi_h|$ will converge to 1.

Proof. Let the population S (a set of nodes in a DL network) contain K successes (honest nodes) and C failures (malicious nodes) where C is constant, and let us increase the population infinitely by adding more successes. As the number of successes approaches infinity, the probability of sampling a success can be approximated by:

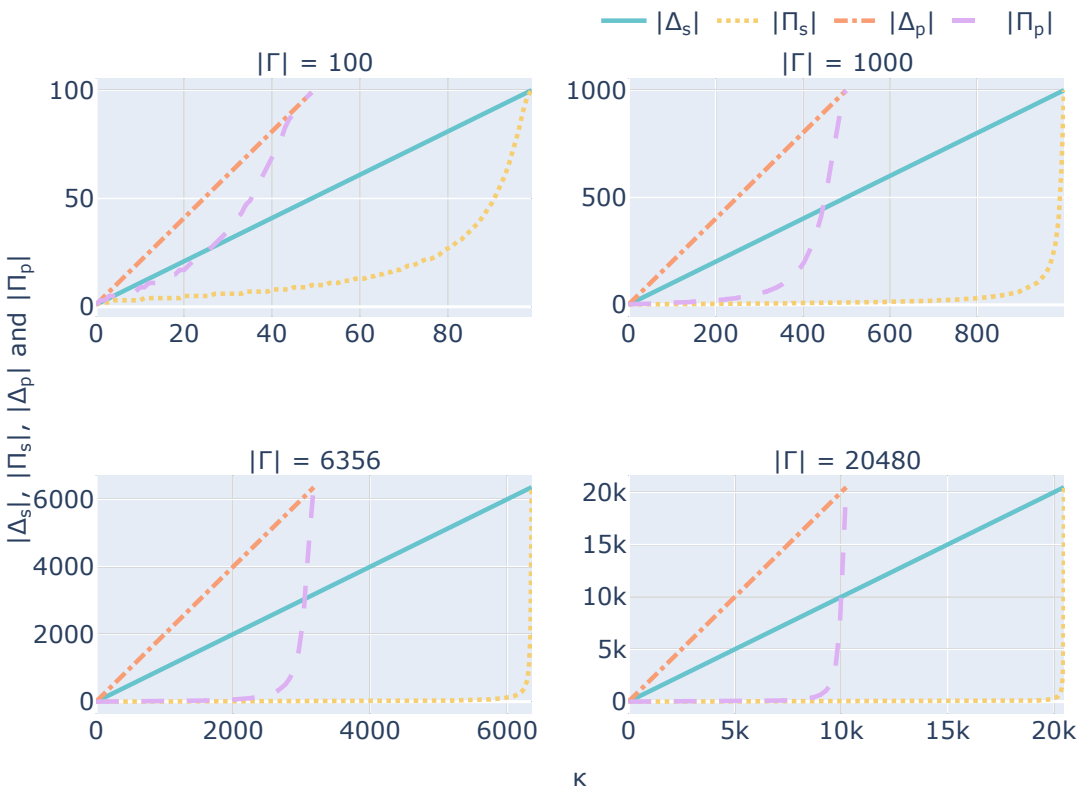

Figure 4.3: Deterministic vs. probabilistic set sizes with an increase of κ for four different population sizes. When population size increases, we observe a greater difference between a probabilistic set size and its deterministic counterpart.

$$P(X = x) = \frac{K}{K + C,} \tag{4.5}$$

and solving the corresponding limit gives us:

$$\lim_{K \to \infty} \frac{K}{K + C} \overset{H}{=} 1 \tag{4.6}$$

Simply put, the probability of sampling an honest node from a network of infinite size with a constant number of malicious nodes is 1, and the probability of sampling a malicious node is 0. Consequently, the size of a set containing at least one honest node is 1. Similarly, the size of a set containing the majority of honest nodes is also 1. This proposition is very beneficial in terms of communication complexity (especially for large networks), since it implies that querying Π_h could have a complexity of $O(1)$.

An experiment was performed to verify whether this is the case, and the result is shown in Fig. 4.4. The number of malicious nodes in the network was kept constant, i.e. $|M| = 100$, while the network S was increased incrementally by adding more honest nodes (i.e., successes). The experiment shows that both $|\Pi_s|$ and $|\Pi_p|$ converge towards 1 quite quickly as the number

of discovered nodes increases.

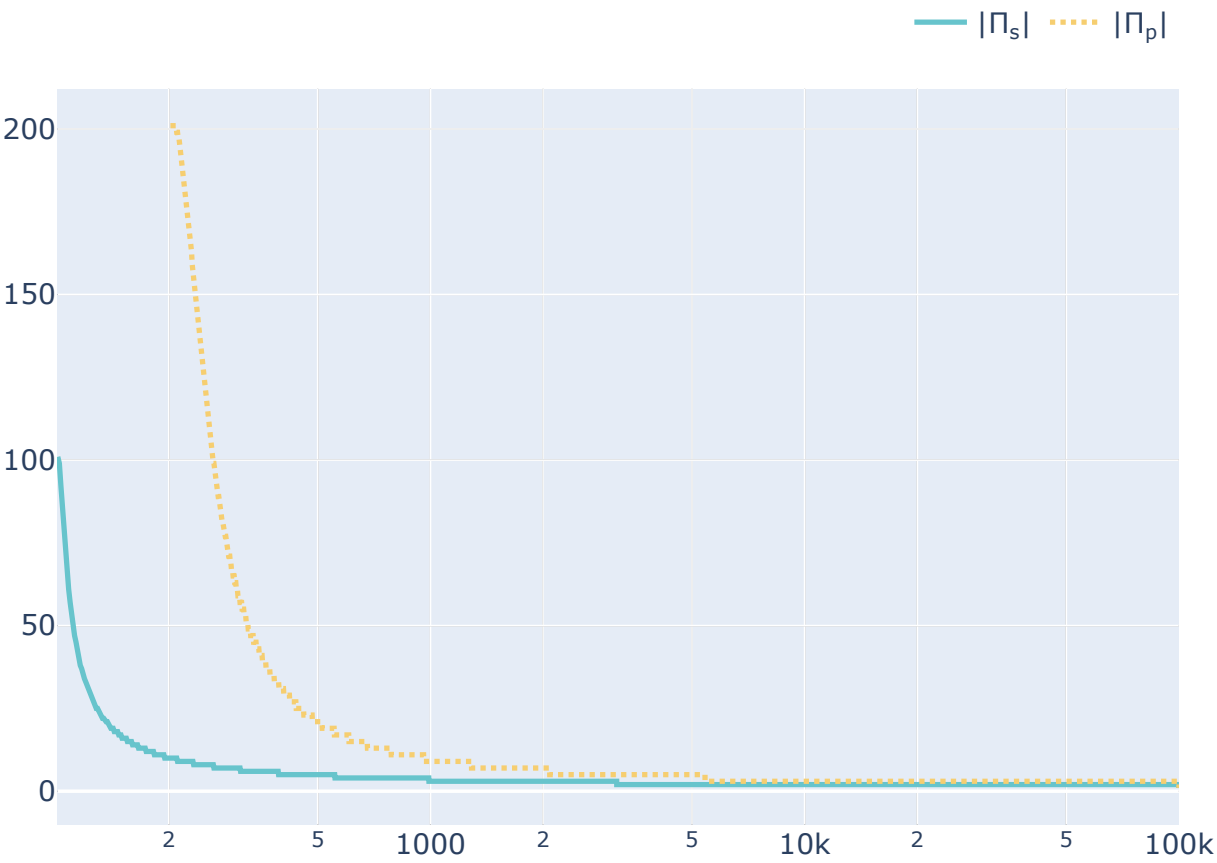

Figure 4.4: If the number of malicious nodes in a population is constant (in this example, $\kappa = 100$), while the number of honest nodes increases, the corresponding probabilistic set sizes decrease (in this example, Π_s and Π_p were generated when $\rho = 0.999$) and eventually their size is reduced to 1.

Research Question Answer 11. [Answers RQ. 3] $|\Pi_h|$ can be reduced by adding more successes (honest nodes) to Γ. If sufficient successes are added to Γ, Π_h will contain a single member.

Here, we open a new research question, which we will answer in the upcoming section:

Research Question 4. In a realistic scenario, how much can $|\Pi_h|$ be reduced by adding more successes to Γ?

4.2.5 Probabilistic set size reduction feasibility

Since the Aurora node uses members of Π_h (i.e., Π_s or Π_p) for transient or persistent communication, reducing $|\Pi_h|$ will diminish the communication complexity of Algorithm 4 and Algorithm 5. As shown in the previous section, the size of Π_h can be reduced by increasing the number of honest nodes (i.e., successes) in the population. In this section, we examine the rate at which $|\Pi_h|$ is reduced with respect to the growth of $|\Gamma|$, and compare $|\Pi_h|$ to sub-linear

functions of κ in the context of the current size of the Bitcoin IPV4 network. The results of the conducted experiment show that the following constraint is pragmatic in a realistic scenario, and will be of great use in the upcoming complexity analysis:

Constraint 1. $|\Pi_h|$ is bounded by $\sqrt{\kappa}$, i.e., $|\Pi_h| \leq \sqrt{\kappa}$.

The experiment was conducted to mimic algorithm execution that creates an increasing set of discoverable network nodes: We varied the number of discoverable nodes $|\Gamma|$ from 2 to 20480, which is the maximum number of total nodes that a Bitcoin client can store [71]. For each $|\Gamma|$, κ was set to $|\Gamma| - 1$ and gradually decreased by 1. During each experimental run, the ratio between $|\Gamma|$ and κ was marked when the following predicates (i.e., conditions) were met:

1. $|\Pi_s| \leq \sqrt{\kappa}$.2. $|\Pi_s| \leq \ln \kappa$.3. $|\Pi_p| \leq \sqrt{\kappa}$.4. $|\Pi_p| \leq \ln \kappa$.

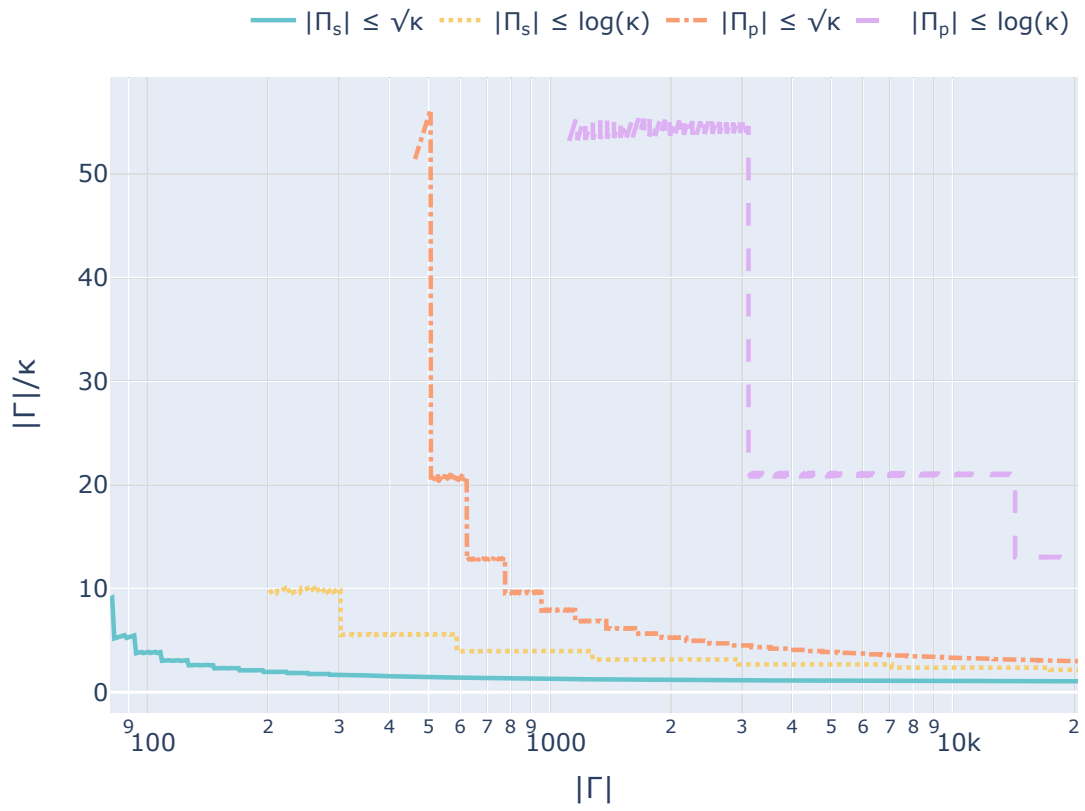

Figure 4.5: Ratio $\frac{|\Gamma|}{\kappa}$ when the four predicates limiting the probabilistic set sizes to $\sqrt{\kappa}$ and $\ln \kappa$ have been satisfied with an increase of $|\Gamma|$.

Results are depicted in Fig. 4.5. By examining the ratios indicating potential reduction of probabilistic set sizes with respect to malicious node tolerance, we can conclude that as the size of the population increases, the ratio between $|\Gamma|$ and κ required to reduce the probabilistic set sizes decreases. Thus, it is easier to satisfy the four predicates limiting $|\Pi_h|$ for larger populations, i.e., for larger sets of discoverable nodes.

Table 4.1 shows the values extracted from Fig. 4.5 for $|\Gamma| = 6356$, which is the number of active Bitcoin IPV4 nodes discovered and reported in [72]. This number is used as the default network size in all the following experiments.

Table 4.1: *ps*: probabilistic set type, f: upper bound for probabilistic set size as a function of κ, $f(\kappa)$: f evaluated at respective κ, P: predicate, $|\Pi|$: respective probabilistic set size, $|\Delta|$: respective deterministic set size

$\|\Gamma\|$	6356			
ps	Π_s		Π_p	
f	$sqrt(\kappa)$	$\ln(\kappa)$	$sqrt(\kappa)$	$\ln(\kappa)$
P	$\|ps\| \leq f(\kappa)$	$\|ps\| \leq f(\kappa)$	$\|ps\| \leq f(\kappa)$	$\|ps\| \leq f(\kappa)$
$\|\Gamma\| - \kappa$	549	3985	4615	6053
κ	5807	2371	1741	303
$\|\Gamma\|/\kappa$	1.09454	2.68073	3.65078	20.9769
$f(\kappa)$	76.20367	7.77107	41.72529	5.71373
$\|\Delta\|$	5808	2372	3483	607
$\|\Pi\|$	76	7	41	5
$\frac{\|\Delta\|}{\|\Pi\|}$	**76.421052**	**338.85714**	**84.951219**	**121.4**
p	0.9990005	0.9990004	0.9990073	0.9990014

For example, by examining the first column in Table 4.1, we see that to satisfy the predicate $|\Pi_s| \leq sqrt(\kappa)$, even up to 5807 of 6356 network nodes can be malicious, while the size of Π_s is less than or equal to $\sqrt{5807} = 76.20367$, which corresponds to 76, and the correct execution is guaranteed with $p = 0.9990005$. If we would want to deterministically find at least one honest node within the same network, $|\Delta_s| = \kappa + 1 = 5808$. In other words, 76.421052 times more nodes need to be queried for a deterministic answer in comparison to a probabilistic answer.

Research Question Answer 12. [Answers RQ. 4] From a pragmatic point of view, the first three predicates limiting probabilistic set sizes with respect to κ (results displayed in the first three columns of Table 4.1) can be satisfied relatively easily, while the last condition has little value for a real-world scenario due to its extremely high $|\Gamma|/\kappa$ ratio. Further reduction of the probabilistic set size requires an even larger $|\Gamma|/\kappa$ ratio and is considered impractical. Therefore, the square root as a sub-linear function that bounds the probabilistic set size with respect

to κ is practical in a realistic scenario.

4.2.6 Probabilistic sets and network topology

So far we have relied on the existence of the set Γ without elaborating on how this set is constructed, as our analysis has been independent of any particular DLT solution or underlying network topology. However, both the topology and specifics of DLT solutions need to be considered when defining the Probabilistic honest set construction algorithm. In this section, we take a more formal approach by providing definitions of terms related to the network topology.

The construction of probabilistic honest sets is a core function of the Probabilistic honest set construction algorithm. We gave a brief overview of the construction and use of these sets within an DL client in Section 4.2. As previously stated, starting from node fc, the Aurora node explores the network in a series of steps, where a single step is defined as:

Definition 8 (*Draw*)**.** A draw is a step performed by the Aurora node during network exploration which consists of two messages: a ping message from the Aurora node to a network node requesting its peer list, and the corresponding pong message containing a response denoted as $\aleph$. A draw at step i expands Γ_{i-1} with the unique nodes found in $\aleph_i$, i.e., $\Gamma_i \leftarrow \Gamma_{i-1} \cup \aleph_i$

A network node can receive multiple ping queries and is assumed to be stateful [73], which means it retains state about which addresses of neighboring peers are revealed to the Aurora node together with a timestamp. Consequently, the network node will not reveal nodes that have already been revealed to the Aurora node in a previous pong message. A network node is no longer queried by the Aurora node if it responds with an empty peer list or fails to respond. Once a draw is successful, the Aurora node terminates a connection to the network node since resources should be released as soon as possible. Draws are executed in steps until a *halting condition* is satisfied.

Definition 9 (Halting condition)**.** Halting condition is a predicate that defines specific conditions which indicate to the Aurora node to terminate further draws in the network.

The Aurora node checks, after each draw, whether the halting condition is satisfied and consequently terminates further draws. Hereafter, unless otherwise specified, the *default halting condition* is defined as a step when all discoverable nodes stop responding to ping requests of node the Aurora node (i.e., there are no available nodes left to query).

We define a sequence of draw steps as follows:

Definition 10 (*Gathering*)**.** Gathering is the process of network exploration which consists of a sequence of draws executed in steps $i = 1 \dots d$, where a halting condition is met in step d. We assume it is executed in a *directed* and *acyclic* manner.

We assume that a network can be represented as a graph in which nodes are represented as vertices and connections between nodes are represented as edges, where edges have an associated direction. In such a graph, a gathering can be associated with a DAG consisting of nodes contacted during a gathering. After a node responds with an empty peer list or does not respond at all, a subsequent node is selected from Γ uniformly at random. As the Aurora node performs the gathering, it ensures that cycle formation is avoided. Note that building a DAG on a network with several thousand nodes would require significant resources, making the solution is not suitable for resource-constrained devices. Instead, it is sufficient to have two sets of nodes, one set containing the nodes contacted during a gathering, and the second set containing nodes not contacted during a gathering, where the already contacted nodes would not be contacted again.

Although at first glance a gathering is similar to the process of recursively scraping the entire network, it is something quite different. The purpose of a gathering is to collect a minimal number of nodes within a minimal period of time to construct probabilistic honest sets. The probabilistic sets are then used by the Aurora node to choose an optimal bootstrap candidate and honest nodes for TPC.

4.3 Pseudocode, parameters and initialization

Having established the influence of network topology to our solution, we are ready to present the pseudocode of the Probabilistic honest set construction algorithm, declare the necessary parameters and initialize them, as displayed in Algorithm 2.

Probabilistic honest set construction algorithm explanation: The lines 1 and 2 initialize local variables. The target of the first draw (*nxtDraw*) is set to fc and the set with unique node fond in a gathering (Γ) is set to an empty set. Next, a hypergeometric distribution is constructed in a loop where the population size N is set to the number of currently discovered nodes Γ, the number of successes in the population K is set to $|\Gamma| - \kappa$, and the sample size n is set to $|\Pi_h|$ (line 4). If it is true that the sample contains at least h honest nodes with at least probability ρ (line 5), then a Π_h can be constructed by sampling without replacement $|\Pi_h|$ nodes found in the gathering (line 6) and returned (7). If a Π_h cannot be constructed, the network must be explored further. A random available node from which no draw has yet been executed is assigned as the target of the next draw (line 9) and a ping message is sent to that node. The nodes from the pong message (line 11) are added to Γ (line 12). If a *halting condition* has been reached (e.g., there are no more unique nodes to ask for ping messages), a gathering is terminated and no Π_h is constructed (lines 13 and 14).

Algorithm 2: Probabilistic honest set construction algorithm

Input : ρ — Probability guarantee
Input : fc — First contact node identifier
Input : hc — Halting condition encapsulation
Input : $|\Pi_h|$ — Maximum Π_h size
Input : h — $[1, \lfloor |\Pi_h|/2 \rfloor + 1]$
Input : κ — Desired malicious node tolerance

```
nxtDraw ← fc;
Γ ← ∅;
do
    X ∼ Hypergeometric(|Γ|, |Γ| − κ, |Π_h|);
    if p(X ≥ h) ≥ ρ then
        Π_h ← Sample without replacement |Π_h| nodes from Γ;
        return Π_h
    else
        nxtDraw ← random responding node in Γ;
        Send ping to nxtDraw;
        ℵ ← peers in pong message from nxtDraw;
        Add ℵ to Γ;
        if hc.isHaltingConditionReached(nxtDraw, ℵ) then
            throw
        end
    end
while True;
```

Initialization of parameters: To use Algorithm 2, at least six parameters are required. As mentioned earlier, it is recommended to set the correctness probability ρ to 0.999, with a note that the impact of ρ on the size of probability sets is part of our future work. The first contact node fc can be found in any way a user sees fit, e.g., via chat, forums, etc. We define the default halting condition hc as the moment when there are no more nodes to ask for peer lists, although we strongly suggest changing this to a condition that better fits the underlying network topology and the resources of a ledger client (an example is given in Section 4.4.4). The maximum size of Π_h can be set in accordance with Const. 1, i.e., $\lfloor \sqrt{\kappa} \rfloor$. The number of honest nodes h in Π_h, i.e. choosing between Π_s and Π_p, depends on the use case, as discussed further in this section. Knowing κ a priori is nontrivial or even impossible in real P2P networks, and instead an estimate or expected number of malicious nodes $\hat{\kappa}$ is used. If a user makes a correct guess and sets $\hat{\kappa} = |M|$, Algorithm 2 executes optimally. If the user sets $\hat{\kappa} < |M|$, then correct algorithm execution cannot be guaranteed. In this case, our solution can be enhanced with a scheme based on the detection of suspicious block timestamps [16] (meaning Assumption 5 does not have to hold), at the cost of higher complexity. If $\hat{\kappa} > |M|$, the correct execution is guaranteed with probability ρ at the cost of higher communication complexity. Thus, this parameter should in general be an overestimation of the envisioned number of malicious nodes in the network, but

the size of Π_h, as per Const. 1 remains bound by $\sqrt{\hat{\kappa}}$.

Since it is excessive to expect that a number $\hat{\kappa}$ can be guessed correctly, an alternative can be offered instead of κ which uses another parameter, the maximum execution time in seconds t_{max} specifying how long a user is willing to wait for probabilistic honest set construction to complete. Such an algorithm is displayed in Algorithm 3, while a user interface running this algorithm is presented in Section 4.6. Within a period of t_{max}, the Aurora node collects nodes. After the specified timeout occurs, using the values from Fig. 4.5, the user is presented with a table such as Table 4.1 to choose an adequate probabilistic set based on the underlying use case.

Algorithm 3: Probabilistic honest set construction algorithm — time-based facade

Input : ρ — Probability guarantee
Input : fc — First contact node identifier
Input : hc — Halting condition encapsulation
Input : $|\Pi_h|$ — Maximum Π_h size
Input : h — $[1, \lfloor |\Pi_h|/2 \rfloor + 1]$
Input : t_{max} — Maximum execution time in seconds

```
nxtDraw ← fc;
Γ ← ∅;
while t_max has not elapsed do
    nxtDraw ← random responding node in Γ;
    Send ping to nxtDraw;
    ℵ ← peers in pong message from nxtDraw;
    Add ℵ to Γ;
    if hc.isHaltingConditionReached(nxtDraw, ℵ) then
        throw
    end
end
κ ← extract from Fig. 4.5;
Π_h ← Sample without replacement √κ nodes from Γ;
return Π_h
```

Note that a user cannot be certain that $\hat{\kappa}$ computed in this way actually reflects $|M|$, just as a user does not know whether the network contains the majority of honest voting power, although this is stated in Assumption 1 as a standard assumption in DLT networks. Nonetheless, as explained in Section 4.1, querying multiple sources in search of the truth enhances the trustlessness of the technology, adds another layer of safety, and is already used in some state-of-the-art solutions [74].

Our solution provides context that is lacking by specifying the conditions under which multiple sources respond with an honest answer. In the example of Electrum that maintains connections to approximately 10 nodes (see Section 4.1), the following assertion can be made if an Electrum client client were to be enhanced with our solution: If the client has discovered 6356 nodes and is connected to 10 remote nodes, the client can be 99.9% certain that 1 node out of

10 is honest even if 3187 ($\approx 50\%$) of discovered nodes are malicious, and that 6 out of 10 are honest if 572 ($\approx 9\%$) of discovered nodes are malicious.

Transaction history synchronization algorithm explanation: Once a Π_s has been constructed it can be used to synchronize with a node as displayed in Algorithm 4. The algorithm requires only one parameter, which is a Π_s constructed by Algorithm 2.

Algorithm 4: Transaction history synchronization algorithm

```
   Input      : Π_s — Probabilistic Safe Set
1  sync ← false;
2  while sync = false do
3  |  candidate ← null;
4  |  for node in Π_s do
5  |  |  candidate ← node offering latest ledger state;
6  |  |  Attempt to syncronize with candidate;
7  |  |  if Attempt successful then
8  |  |  |  sync ← true;
9  |  |  else
10 |  |  |  Remove candidate from Π_s;
11 |  |  end
12 |  end
13 end
```

First, a flag indicating whether an THS was successfully executed is set to $false$ (line 1). While the THS is unsuccessful (line 2), select a candidate that may offer canonical chain (lines $3-5$), and attempt an THS (line 6). If the attempt is successful, update the *sync* flag, which stops the loop (lines $7-9$). Otherwise, remove this candidate from Π_s (line 10) and try the THS again.

Transaction presence checking algorithm explanation: Once a Π_p has been constructed, it can be used for TPC, as indicated in Algorithm 5. The algorithm requires four parameters: the identifier of the transaction tx_{id} that is supposedly present in the ledger, the identifier of the block supposedly containing transaction tx_{id}, the Merkle root of the Merkle tree containing transaction tx_{id}, and a Π_p constructed by Algorithm 2. The first three parameters are given by the person who wants to prove (i.e., check) that a transaction was submitted (i.e., is present in a DL) (e.g., a buyer who wants to prove that a payment was made for goods). The last parameter is a Π_p created by Algorithm 2.

The local variables of the algorithm are initialized in lines $1-3$. First, the majority number of nodes in a Π_p is computed (line 1). Second, the number of nodes in a Π_p that asserted that a transaction is present in the ledger (variable *present*, line 2) is set to 0. Third, the number of nodes in a Π_p that asserted that a transaction is not present in the ledger (variable *notPresent*, line 2) is set to 0. Next, a Merkle proof is requested in a loop for each node in the Π_p (line 5) and

Algorithm 5: Transaction presence checking algorithm

```
   Input  : Merkle root — String, merkle root
   Input  : tx_id — String, transaction ID
   Input  : blk_id — String, block ID
   Input  : Π_p — Probabilistic Progress Set
1  majority ← ⌊|Π_p|/2⌋ + 1;
2  present ← 0;
3  notPresent ← 0;
4  for node in Π_p do
5  |  Request Merkle proof from node;
6  |  Verify Merkle proof for tx_id in blk_id using Merkle root;
7  |  if node offers valid Merkle proof then
8  |  |  present ← present + 1;
9  |  else
10 |  |  notPresent ← notPresent + 1;
11 |  end
12 |  if present = majority then
13 |  |  Declare transaction present;
14 |  |  return;
15 |  else if notPresent = majority then
16 |  |  Declare transaction not present;
17 |  |  return;
18 |  end
19 end
```

then checked as described in Section 2.1.2 (line 6). If the Merkle proof is valid, *present* is incremented (line 8), otherwise *notPresent* is incremented (line 9). If *present* ever reaches *majority*, then a transaction can be declared as present (lines $12-14$). Otherwise, the transaction can be declared as not present (lines $15-17$).

4.3.1 Complexity

In the scope of this section, we express the time, communication and space complexity of our solution. Although the comparison between Π_h and Δ_h has already been given in Section 4.2.5, to further justify the utility of our solution, we compare our solution to a simplified SPV model. We assume that an SPV client must download all chain headers since the genesis block to verify the integrity of the header chain. Furthermore, since an SPV depends on the existence of at least one honest node (unlike our solution), we must assume that the SPV has access to such a node.

We express time complexity r as the number of rounds executed by the Aurora node during a gathering plus the number of rounds executed when communicating with members of Π_h (i.e., before generating a response). Communication complexity m is expressed as the maximum number of messages the Aurora node will exchange before generating a response. We express

space complexity as the number of data objects s stored on a node and required for the execution of our solution.

Time complexity

The time complexity of our solution can be decomposed into two main parts. First, the execution of Algorithm 2, as the number of rounds executed during a gathering, where a round in a gathering is represented by the submission of a ping request and the corresponding pong response (i.e., the number of draws). Second, the number of rounds executed during communication with members of Π_h (i.e, the execution of Algorithm 4 or the execution of Algorithm 5). Thus, the time complexity is a sum of these values, which can be derived from the following three equations.

First, a single gathering can be described (i.e., approximated) as a linear function where the domain is the number of steps performed during a gathering d, the co-domain is the number of discoverable nodes discovered during the gathering $|\Gamma|$, and the slope Y is a random variable. Thus, we write:

$$|\Gamma| = f(d) = Y * d. \tag{4.7}$$

The average slope Z is a random variable and is the average of the sample means[¶] that can be calculated for any network or topology via simulation that approximates a normal distribution. By applying the previous statement to Eq. (4.7), we obtain the following:

$$|\Gamma| = f(d) = Z * d. \tag{4.8}$$

Second, as observed in Fig. 4.5, the ratio $\omega = \frac{|\Gamma|}{\kappa}$ when $|\Pi_s|$ and $|\Pi_p|$ begin behaving like $\sqrt{\kappa}$ is a function of κ and can be precomputed[‖]. Thus, the total unique number of nodes $|\Gamma|$ can also be expressed as

$$|\Gamma| = f(\kappa) = \omega * \kappa. \tag{4.9}$$

Using Eq. (4.8) and Eq. (4.9), we get the following:

$$\omega * \kappa = Z * d \implies d = \frac{\omega}{Z} * \kappa. \tag{4.10}$$

Since there are d steps in a gathering, the number of rounds executed during a gathering can be expressed as

[¶]Central Limit Theorem (CLT)

[‖]E.g, for Π_p, ω ranges from 55.22 to 1.99 for all relevant values of the domain (i.e., BTC).

$$f(\kappa, Z) = \left\lceil \frac{\omega * \kappa}{Z} \right\rceil. \tag{4.11}$$

Third, given Const. 1, the number of rounds executed when communicating with the members of the constructed Π_h can be expressed as follows:

$$f(\kappa) = |\Pi_h| = \lfloor \sqrt{\kappa} \rfloor. \tag{4.12}$$

Consequently, the total number of rounds executed before Aurora node generates a response can be expressed as the sum of Eq. (4.11) and Eq. (4.12):

$$r(\kappa, Z)_A = \left\lceil \frac{\omega * \kappa}{Z} \right\rceil + \lfloor \sqrt{\kappa} \rfloor, \tag{4.13}$$

where ω can be read from Fig. 4.5 and Z can be set equal to the threshold on the average number of newly discovered nodes per draw (see Section 4.4.4). Therefore, we conclude that the time complexity of our solution is $\mathcal{O}(\kappa)$.

Unlike our solution, the SPV needs to download and process the entire header chain, and then verify the state of a transaction. For a header chain containing h headers, where processing a single header or verifying the state of a transaction represents a round, the total number of rounds before the SPV generates a response is

$$r_{SPV}(h) = h + 1 \tag{4.14}$$

and has the complexity of $\mathcal{O}(h)$. In other words, the time complexity of our solution is independent of the size of the header chain, while the time complexity of an SPV is independent of the size of the network under the assumption that it synchronizes with an honest peer.

Communication complexity

Every step in a gathering involves the exchange of ping and pong messages. Furthermore, the maximum number of messages exchanged when communicating with members of Π_h is exchanged when all members of Π_h are queried. In total, $|\Pi_h|$ requests and $|\Pi_h|$ responses are exchanged. Using Eq. (4.13), we can express the maximum number of messages our solution will exchange before generating a response as follows:

$$m(\kappa, Z)_A = 2 * \left\lceil \frac{\omega * \kappa}{Z} \right\rceil + 2 * \lfloor \sqrt{\kappa} \rfloor \tag{4.15}$$

and conclude that it has the complexity of $\mathcal{O}(\kappa)$. Once a gathering has been executed and Π_h can be reused, our solution has communication complexity of $\mathcal{O}(\sqrt{\kappa})$.

An SPV client on the other hand, needs to download all the headers. Depending on a

DLT implementation, the number of exchanged messages to download the entire header chain varies. For example, in BTC headers can be requested from peer nodes in a bulk with up to 2000 headers at a time [75]. The maximum number of messages the SPV client will exchange before generating a response is

$$m_{SPV}(h) = 2 * \left\lceil \frac{h}{2000} \right\rceil + 2. \tag{4.16}$$

Thus, we can conclude that it has the complexity of $\mathcal{O}(h)$. Once the header chain has been downloaded, the complexity turns to $\mathcal{O}(1)$. In conclusion, given that the header chain has been downloaded, our solution generates more messages than a standard SPV, but it needs to trust the node from which the header chain has been downloaded.

Space complexity

The purpose of every gathering is to construct the set of discoverable nodes Γ. The size of Γ is a multiplier of κ, as expressed by Eq. (4.9). Thus we can conclude that the number of data objects (i.e., remote peer address) stored on the device by our solution is:

$$s(\kappa)_A = \omega * \kappa \tag{4.17}$$

and that it has the complexity of $\mathcal{O}(\kappa)$.

In contrast, an SPV client needs to process every header. If there are h headers, we can express the number of data objects (i.e., chain headers) stored on a device as

$$s_{SPV}(h) = h. \tag{4.18}$$

In conclusion, the number of data objects stored by the SPV client has a space complexity of $\mathcal{O}(h)$. In realistic use cases, this means that our solution consumes significantly less storage compared to a standard SPV client. For example, our *Proof of Concept (PoC)* Aurora module which was integrated into the ETH Trinity client consumes approximately 1.31 MB of additional memory at runtime [76], whereas a Trinity client running as an SPV consumes more than 5 GB [8] of space, i.e., our solution consumes three orders of magnitude less space.

4.4 Evaluation in a simulated environment

Now that we have a better understanding the Aurora algorithms not only in the context of set creation, but also in a DLT environment, we move on to the methodology used for the evaluation of the Aurora algorithms. Evaluating our solution in a DLT production network is problematic — including malicious actors in a production network is resource intensive, detrimental, and

mostly useless since we would need to know the absolute state of the network to interpret the results. For these reasons, we modeled the evaluation as a two step procedure. The first part of the evaluation procedure, which measured the efficiency of our solution, was conducted in a simulated environment. To better understand the resource consumption of our solution as well as its compatibility with existing solutions, the second part of the evaluation procedure consisted of implementing our solution in an open source ETH client and running it on the ETH production network (*mainnet*).

4.4.1 Modeling the network topology

The network used in simulations was designed to resemble the BTC network topology. The core of the network was modeled as strongly connected, while the edge of the network was lightly connected [72, 77]. An unidirectional connection between node A and node B was formed when node knew about the network identifier of node B.

Network topology is expressed as a mapping

$$T = \{t_1 \mapsto \overrightarrow{t_1},\ \ldots,\ t_i \mapsto \overrightarrow{t_i},\ \ldots,\ t_n \mapsto \overrightarrow{t_n}\ \}, \tag{4.19}$$

where the keys t_i represent network nodes, while the values $\vec{t_i}$ represent a vector of nodes known to t_i, i.e., possible candidates to be included in peer lists.

Network topology was derived from a vector named **node capacities**. We define **node capacities** $\hat{c}$ as a vector where each element under index i represents the number of nodes known to node i:

$$\hat{c} = [c_1, c_2, \ldots, c_n] \tag{4.20}$$

Node capacities are modelled by sampling with replacement from an **exponential distribution** with a mean 507.5, which is an extrapolation of the data measured by [72] used to model the number of active IP addresses known to a network peer for the Bitcoin IPv4 production network slice:

$$X \sim Exp(\lambda = 1/507.5) \tag{4.21}$$

The distribution was truncated to the interval $[2, 6356]$, meaning a peer can know at least 2 peers (itself and another remote peer), and at most 6356 peers (itself and 6355 remote peers), where 6356 is the number of active Bitcoin IPV4 nodes discovered and reported in [72]. This number is used as the default network size in all the following experiments. Each individual capacity was sampled with **inverse transform sampling** and rounded to the nearest integer.

Edges between nodes were iteratively modelled using weighted sampling without replace-

ment, implemented as **fitness proportionate selection**, i.e. **roulette wheel selection** [78], derived from $\hat{c}$, defined in iteration i as:

$$\hat{c}^i = [c_1^i, c_2^i, \ldots, c_n^i] \tag{4.22}$$

For every $\hat{c}^i$, a vector of weights $\hat{w}^i$ was calculated, which was used for weighted sampling, such that a higher weight assigned to a specific node translates into a higher probability for that node to be selected as a candidate for edge formation, as expressed by Eq. (4.26). Vector $\hat{w}$ in iteration i was expressed as:

$$\hat{w}^i = [w_1^i, w_2^i, \ldots, w_n^i] \tag{4.23}$$

where each individual element was calculated as:

$$w_n^i = \frac{c_n^i}{\max c^i} \tag{4.24}$$

For every network topology element t_z, a vector of weights in iteration i containing all weights of nodes unfamiliar to node t_z was calculated as:

$$\hat{U}_z = [w_1^i, \ldots, w_j^i, \ldots w_m^i] \text{ for every } j \text{ unfamiliar to } z \tag{4.25}$$

Finally, the probability for nodes t_z and t_j to form a bidirectional edge is expressed as:

$$P(E = \{t_z t_j\}) = \frac{w_j}{\sum_i \hat{U}_z^i} \tag{4.26}$$

After forming a bidirectional edge between t_z and t_j, $\hat{c}^{i+1}$ was given as:

$$\hat{c}^{i+1} = [c_1^i, \ldots, c_z^i - 1, \ldots, c_j^i - 1, \ldots, c_n^i] \tag{4.27}$$

When bidirectional edges could no longer be formed, unidirectional edges were formed between t_z and all nodes known to t_z's neighbors. Any disconnected network snapshot (i.e., partitioned network) was discarded.

4.4.2 Simulator architecture

The Simblock [79] simulator written in Java has been forked and contributed to[**]. In addition, capabilities relevant for running our experiments on a BTC-like network topology were added to the simulator, namely that nodes can send ping messages (*GETADDR* messages) and reply with pong messages (*ADDR* messages) as defined in [10, 71, 72, 73, 77]. Algorithms defined

[**]https://github.com/dsg-titech/simblock

in Section 4.3 have been rewritten to conform to the simulators underlying discrete event driven environment. Finally, the simulator has been rewritten to use the Java Streaming *Application Programming Interface (API)* and consequently allow for parallel execution.

The parameters for the simulator added on top of the existing Simblock codebase are displayed in Fig. 4.6. In particular, the parameters are:

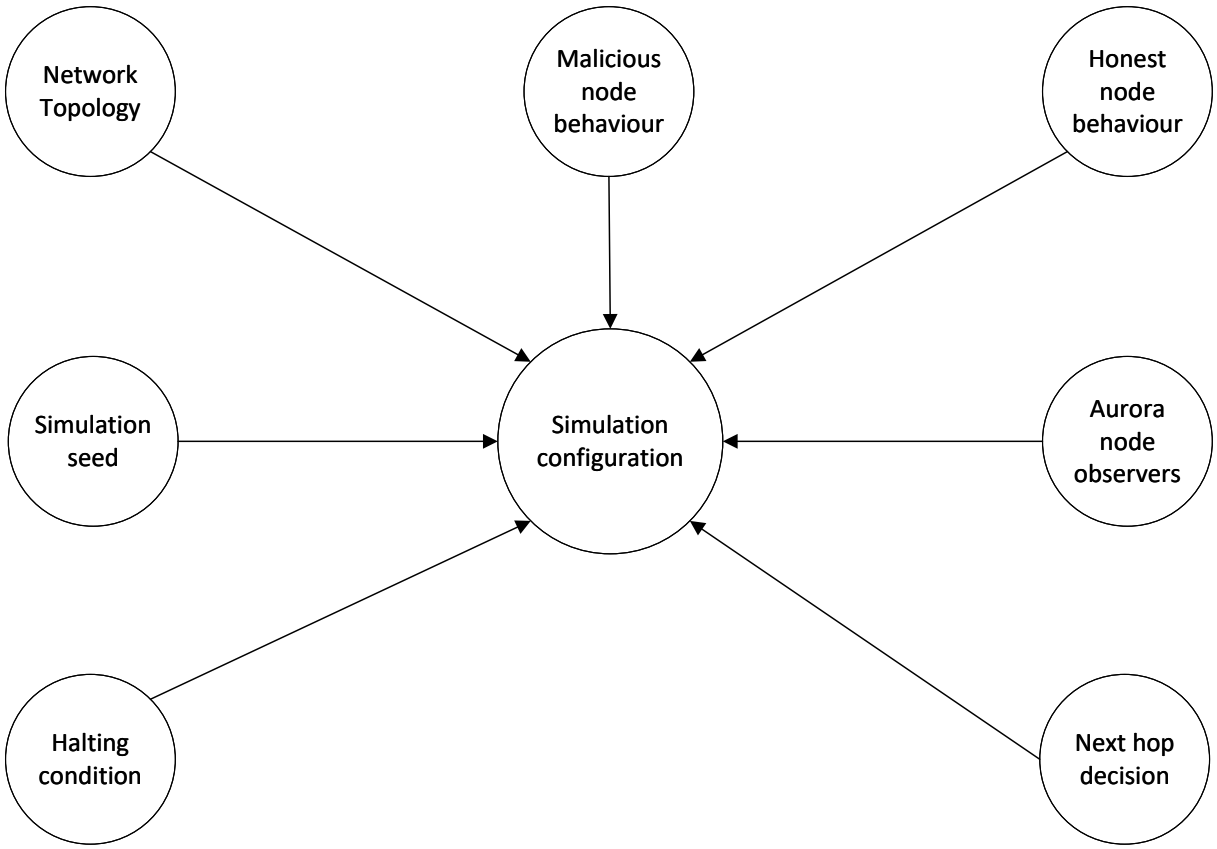

Figure 4.6: Additional parameter added on top of the Simblock simulator relevant for the execution of Aurora related experiments.

1.The network topology as defined in Section 4.4.1, except for malicious nodes, which are always fully connected [80].

2.The behavior of malicious nodes that always expose only malicious nodes, always offer a fake chain head with a higher difficulty compared to the canonical chain, and always lie about the state of a transaction.

3.The behavior of honest nodes that always follow protocol (in this case, the BTC protocol).

4. *Aurora node observers*, injected dependencies [81] and part of an observer design pattern [82] used to collect metadata from a simulation instance. The Aurora node is the subject, and maintains a list of Aurora node observers and notifies them automatically of any state changes. These dependencies are discussed in detail in Section 4.4.3.

5.Next draw decision, an injected dependency that determines how the Aurora node selects the candidate for the next draw.

6.A halting condition for the Aurora node to terminate further draws in the network.

7.A simulation seed for reproducibility.

Revisiting the high-level architecture of our solution displayed in Fig. 2.1, we conclude that our solution influences the Data layer, as the Aurora node exchanges status request, status response, proof request and proof request messages with members of Π_h. Furthermore, it influences the Network layer as the Aurora node exchanges ping and pong messages with members of Π_h. Our solution does not influence the Consensus layer, which means that it does not influence consensus related metrics, for example the number of transactions per second that a DLT network can process or the cost of the execution of a transaction, although the solution can be modified to support soft forks which are the consequence of a stochastic consensus mechanism, as is discussed in Chapter 6.

4.4.3 Simulations and the observer design pattern

To allow for monitoring of arbitrary data related to the Aurora node during a simulation instance, the observer pattern was used, as displayed in Fig. 4.7.

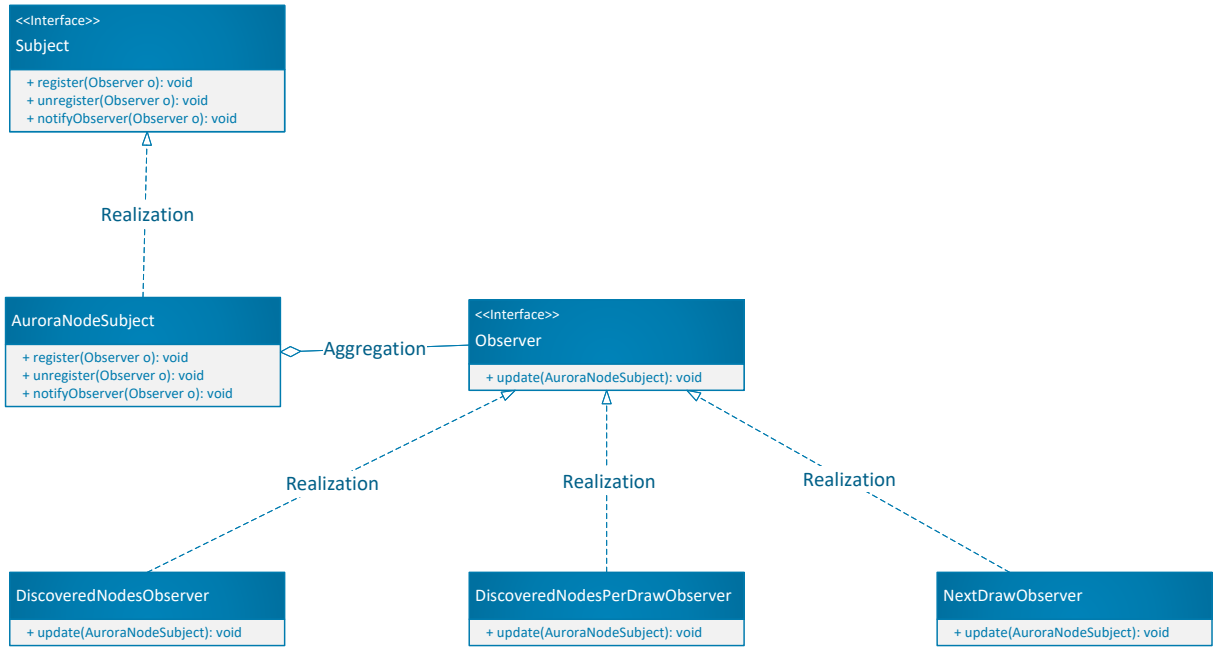

Figure 4.7: UML class diagram showing the observer design pattern used to track data collected by the Aurora node through a simulation.

The Aurora node implements the *AuroraNodeSubject* interface and registers dependencies (i.e., objects implementing the *Observer* interface). After each draw, the Aurora node notifies the registered observers about its state, and the observers keep track of the relevant data. Fig. 4.7 shows example realizations of the *Observer* interface and three example observers, where the *DiscoveredNodesObserver* marks the number of discovered nodes at the end of the gathering, the *DiscoveredNodesPerDrawObserver* marks the number of discovered nodes at the end of

each draw during the gathering, and the *NextDrawObserver* marks the sequence of nodes selected for a draw.

Experiments were run as a series of simulations, with the flow of an experiment shown in Fig. 4.8. A simulation instance is created using the specified configuration (see Fig. 4.6). The configuration includes Aurora node observers (see Fig. 4.7). After a simulation was run, the results were collected and the results were aggregated after sufficient simulations were run.

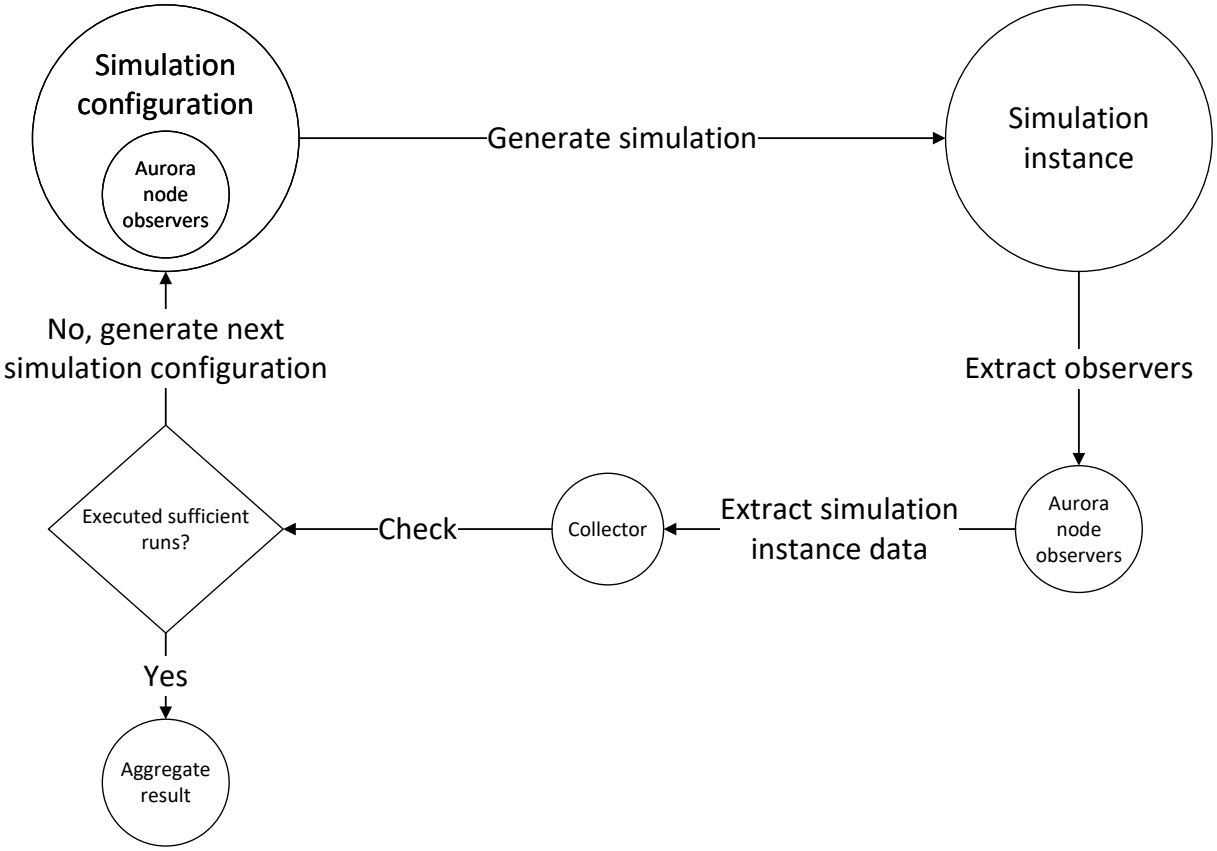

Figure 4.8: Flow of an experiment.

4.4.4 Results

To verify our solution, five different experiments were conducted on a network containing 6356 nodes generated as defined in Section 4.4.1. Each experiment is designed to answer a specific research question. For the sake of simplicity, there was no block generation during the execution of experiments.

The Adversary: a malicious node is a member of a malicious clique of size $|M|$ that exhibits malicious or Byzantine behavior. We assume that the members of the malicious clique only reveal other members of the clique as their neighbors to prevent the Aurora node from contacting honest nodes. Honest nodes, on the other hand, reveal their neighbors randomly (i.e., according

to the protocol). Honest nodes reply to a status request or a proof request message honestly (according to the protocol). Finally, there was only one malicious cluster present in the network.

Experiment 1

Research Question 5. How efficient is the execution of the Aurora module in an idealistic scenario where the Aurora node has unlimited time and resources, when applied in a realistic network with malicious nodes?

Methodology: The efficiency of the Aurora module is measured by running a Monte Carlo simulation of the network while gradually increasing the number of malicious nodes from 0% to 100%. Malicious nodes are selected uniformly at random from the existing nodes in the topology and marked as malicious. First contact nodes fc are also selected uniformly and randomly from malicious nodes to ensure a 99% confidence level and a 1% margin of error. We use the default halting condition for each gathering, i.e., the Aurora node halts the gathering if there are no more nodes to ask for peer lists. The desired malicious node tolerance κ is always set equal to the maximum value $|M|$. The ultimate goal of the Aurora node is to:

1. Check for the presence of a transaction (TPC), which means executing Algorithm 2 followed by Algorithm 5.
2. Initiate THS with an honest peer, which means executing Algorithm 2 followed by Algorithm 4.

An outcome of a simulation run is classified as a *halt* (trace *PPS not constructed*) if Algorithm 2 cannot guarantee protection against a desired number of malicious nodes and the Aurora node decides to abort operation. In contrast, an outcome is classified as *progressed* (trace *TPC successful*) when Algorithm 2 managed to construct a Π_p, and Algorithm 5 was executed. An outcome is marked as *success* (trace *Success rate*) if the Aurora node has chosen to *halt* or has *progressed* to execute TPC with a Π_p containing a majority of honest nodes. Otherwise it is marked as *failure* (trace *Failure rate*) since Aurora node has constructed a Π_p that does not contain a majority of honest nodes and has failed to execute TPC correctly. Finally, trace *PSS constructed* shows outcomes where the Aurora node managed to construct a Π_s which means that THS can be executed with an honest node with probability ρ. The results are displayed in Fig. 4.9.

Research Question Answer 13. [Answers RQ. 5] The Aurora module operates as expected, i.e., it maintains a high success rate regardless of the underlying network topology or entry point, and creates a Π_h or halts in 99.9% of cases regardless of the network conditions. The Aurora module consistently halts when the number of malicious nodes in the network exceeds 50%. It halts only when the first contact is malicious and the gathering could not collect a desired number of nodes, effectively preventing the victim to be eclipsed.

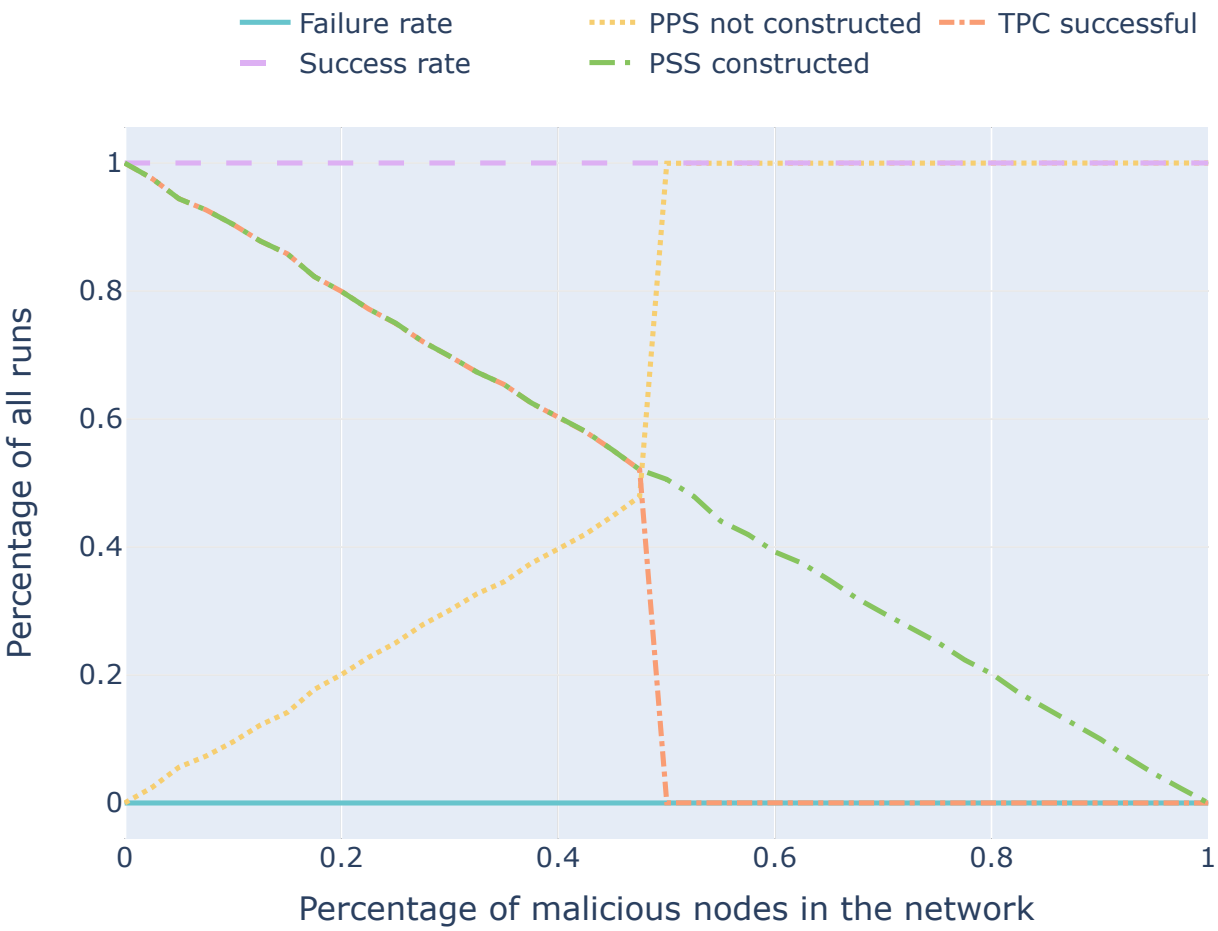

Figure 4.9: Outcomes of the experiment with an increase of the percentage of malicious network nodes.

Experiment 2

Research Question 6. How does the number of draws relate to the number of discovered nodes in a gathering when there are no malicious nodes in the network?

Methodology: A total of 1000 experiments were conducted where the Aurora node enters a network using a randomly-selected first contact node fc. The Aurora node has unlimited time and resources, and the halting condition remains the default one. We define the **node discovery rate** as the percentage of all network nodes discovered per draw.

Fig. 4.10 shows the average node discovery rate for the executed 1000 runs, while Fig. 4.11 shows a sample of 10 such runs. The shaded region around the average represents uncertainty and is bounded by one standard deviation. The results show that the node discovery rate is irregular and depends on the underlying topology and entry point. In conclusion, the following can be stated:

Research Question Answer 14. [Answers RQ. 6] If a node encounters a high-degree node

early, more new remote nodes are discovered per draw compared to the situation when a low-degree node is encountered early.

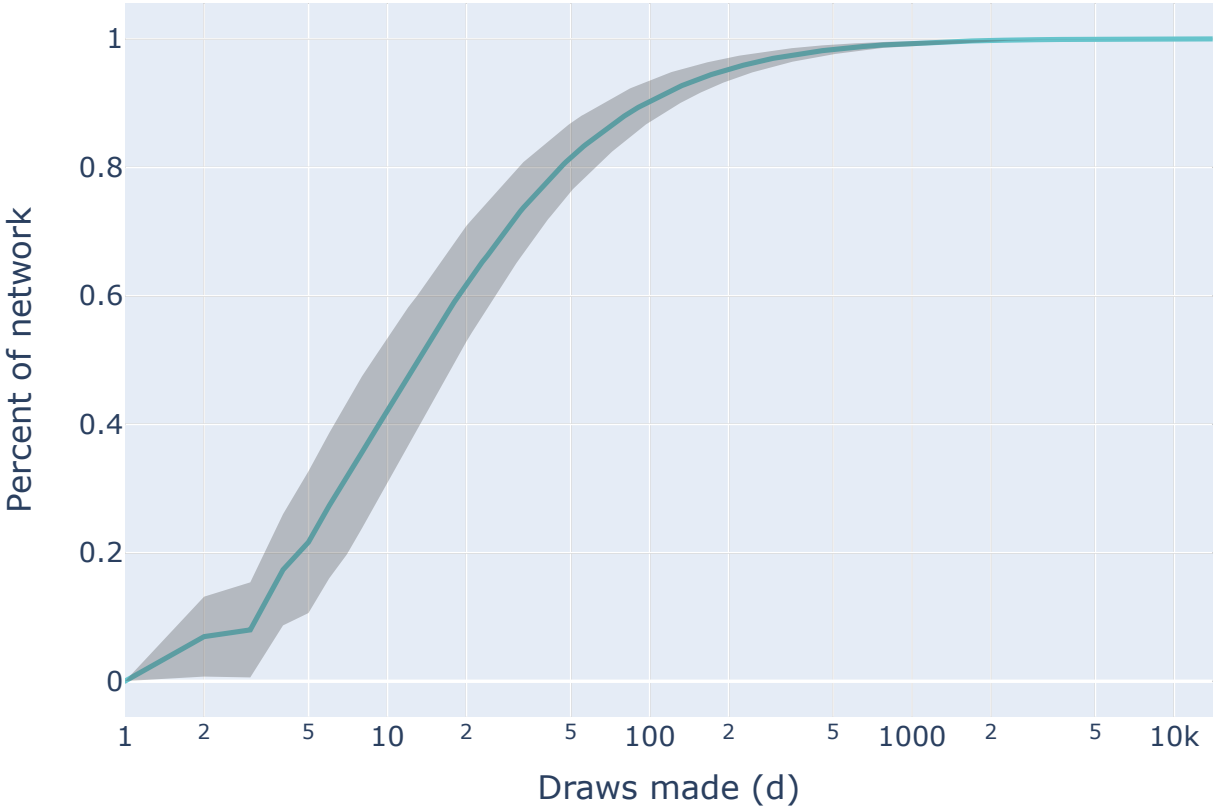

Figure 4.10: The average and standard deviation of the node discovery rate with an increase of the number of gathering steps during 1000 experiments.

The results lead us to two conclusions. First, the number of newly discovered nodes at the beginning of a gathering shows a linear dependence on the number of gathering steps. As the gathering progresses, the expansion rate slows down, which means that new nodes are harder to find. Second, after a certain point in time, the continuation of a gathering makes little sense, as very few, if any, new nodes are discovered. From a pragmatic point of view, it is unreasonable to discover the entire network before the gathering terminates. The communication complexity of the gathering can be reduced by introducing a more complex halting condition which we investigate in the following experiment.

Experiment 3

Research Question 7. Can a custom halting condition reduce the duration, and thus the resource consumption of a gathering while still allowing the gathering to discover a significant number of network nodes?

Methodology: The Aurora node enters a network using a randomly-selected first contact node fc and monitors the average number of newly discovered nodes in a draw (at least 10 draws are

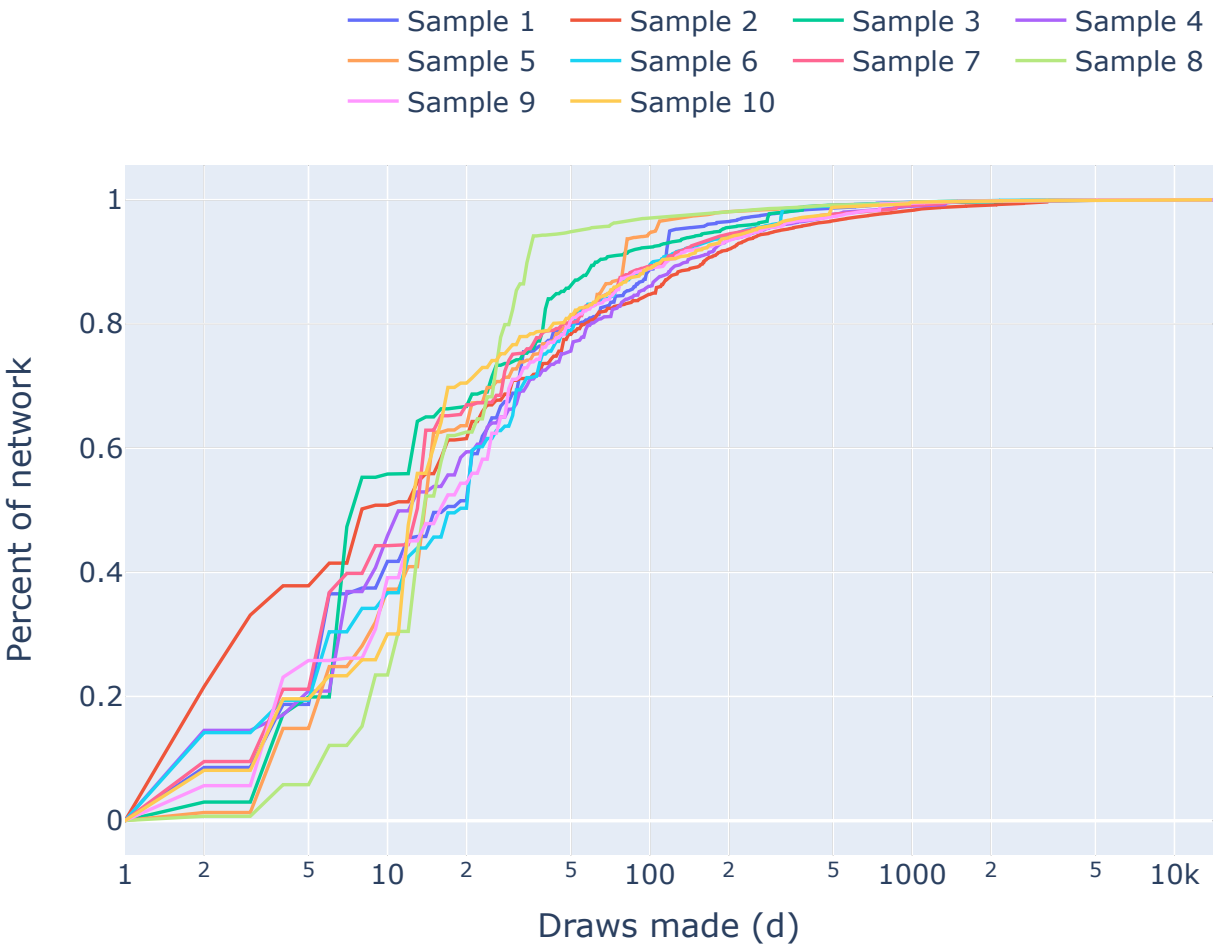

Figure 4.11: Sample of 10 traces that were used to calculate the average number of nodes discovered per draw in a gathering.

needed before an average is calculated). If the average number of newly discovered nodes in a draw falls below a threshold, the node halts. The threshold varies from 1 to 500, and for each threshold, a total of 1000 experiments is performed, based on which the average and standard deviation of the percentage of the network discovered is calculated.

Results: Fig. 4.12 shows the average percentage of the network that has been discovered as the threshold for the average number of newly discovered nodes increases. The shaded region around the average represents uncertainty and is bounded by one standard deviation. The experiment confirms the expected behavior, and the following can be stated:

Research Question Answer 15. [Answers RQ. 7] Larger thresholds are not reliable in terms of percentage of network discovered, while smaller thresholds provide a more consistent result, regardless of the number of nodes known by the first contact node.

Since Fig. 4.12 shows that when introducing a minimal threshold on the average number of newly discovered nodes in a draw results with a high percentage of the network nodes discovered, we continue by examining a setup when the threshold is set to 15 (i.e., the Aurora node

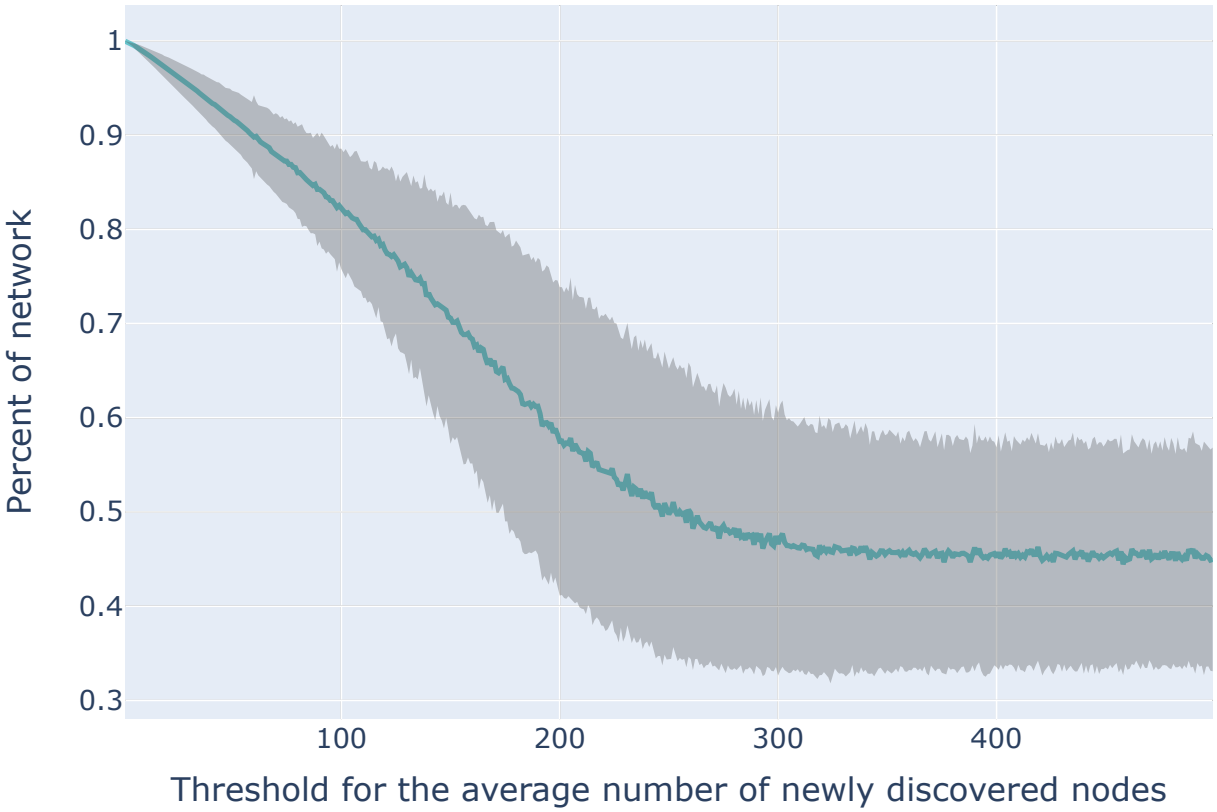

Figure 4.12: The average and the uncertainty of the percentage of the network nodes seen at the end of a gathering with an increase of the threshold for the average number of newly discovered nodes in a draw.

must discover on average at least 15 new nodes after each draw). When such a threshold is chosen, the node discovers on average 98.00% of network nodes with a standard deviation of 0.911%, as shown in Fig. 4.13. This particular threshold is used in the following experiments, as we consider the result beneficial for a real world usage scenario leading to a reliable result in terms of the percentage of network nodes discovered. Note that the threshold chosen in the scope of this experiments is not applicable to any network. The method for obtaining the threshold is generic and should be repeated for a new network topology.

Experiment 4

Research Question 8. Is the expression for communication complexity given in Eq. (4.15) valid and does it hold in a real network regardless of the number of malicious nodes?

Methodology: We ran two simulation scenarios and for each scenario 1000 experiments were performed where the Aurora node enters the network using a randomly-selected first contact node fc. The threshold on the average number of newly discovered nodes per draw is set to 15 in both scenarios[††]. In each experiment, the Aurora node attempts to construct and query a

[††]Note that this value has the same semantics as Z, which is used in Eq. (4.15).

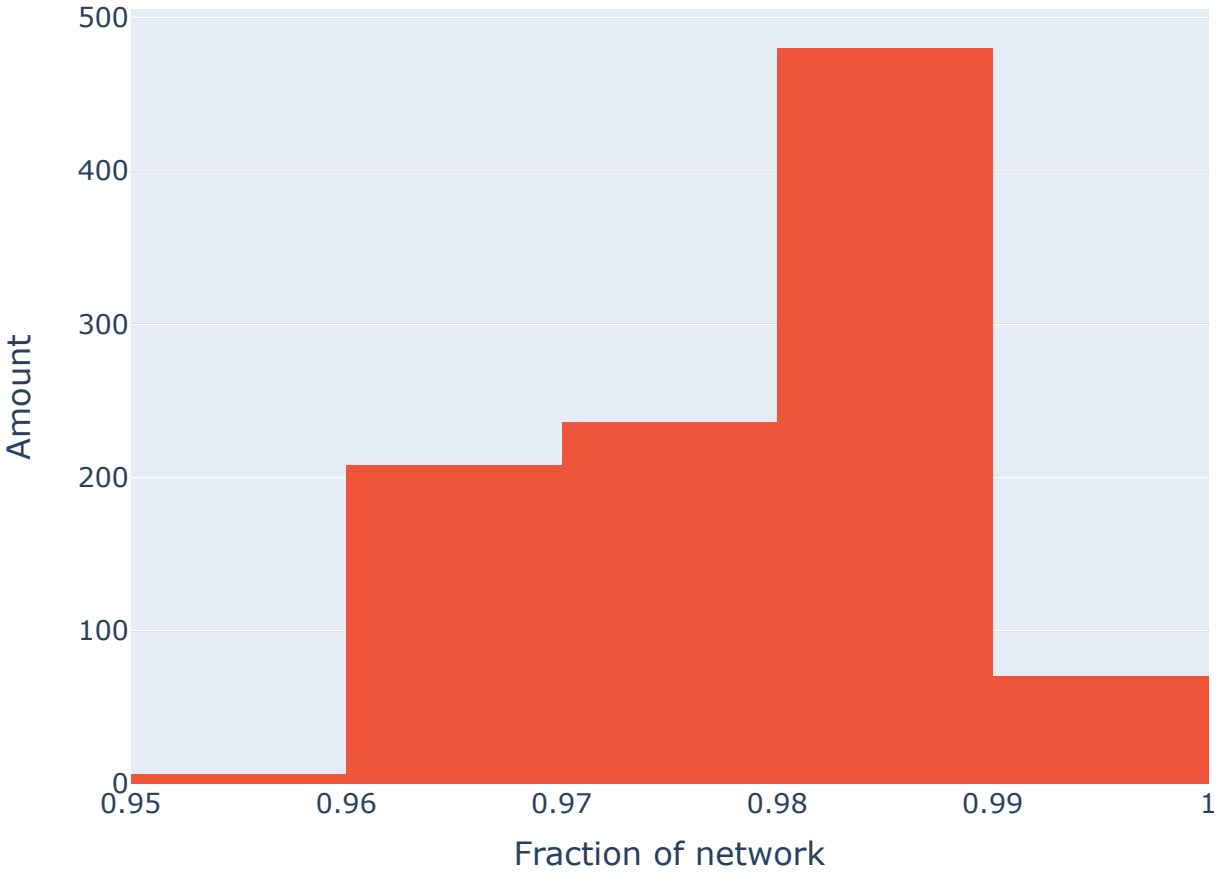

Figure 4.13: The histogram of the average percent of the network discovered when a threshold of the average number of unique nodes discovered per draw is used as a halting condition and set to 15.

probabilistic progress set Π_p. The construction and use of a probabilistic safe set Π_s is intentionally omitted here, as it is much more difficult to construct a Π_p, making Π_p more suitable for this experiment, which can be viewed as a stress test. Nevertheless, Eq. (4.15) can be applied regardless of the probabilistic set type (safe of progress). During the experiment we count the number of messages exchanged before the execution of Algorithm 2 followed by Algorithm 5 terminates, when either the Aurora node managed to progress or because it halted.

In the first scenario, we set $\kappa = |M| = 1272$. In this setting, the probabilistic set Π_p can be constructed and Const. 1 can be met quite easily. Eq. (4.15) gives us an upper bound of 728 exchanged messages. In the second scenario, we set $\kappa = |M| = 1614$. Compared to the first setting, this one is more demanding: the Aurora node must collect at least 6000 total nodes in a gathering in order to satisfy Const. 1. Eq. (4.15) gives us an upper bound of 880 exchanged messages. The results are shown in Fig. 4.14.

Research Question Answer 16. [Answers RQ. 8] In all simulation runs, the actual number of generated messages is smaller then or equal to the corresponding analytic bound, which strongly suggests that Eq. (4.15) is valid.

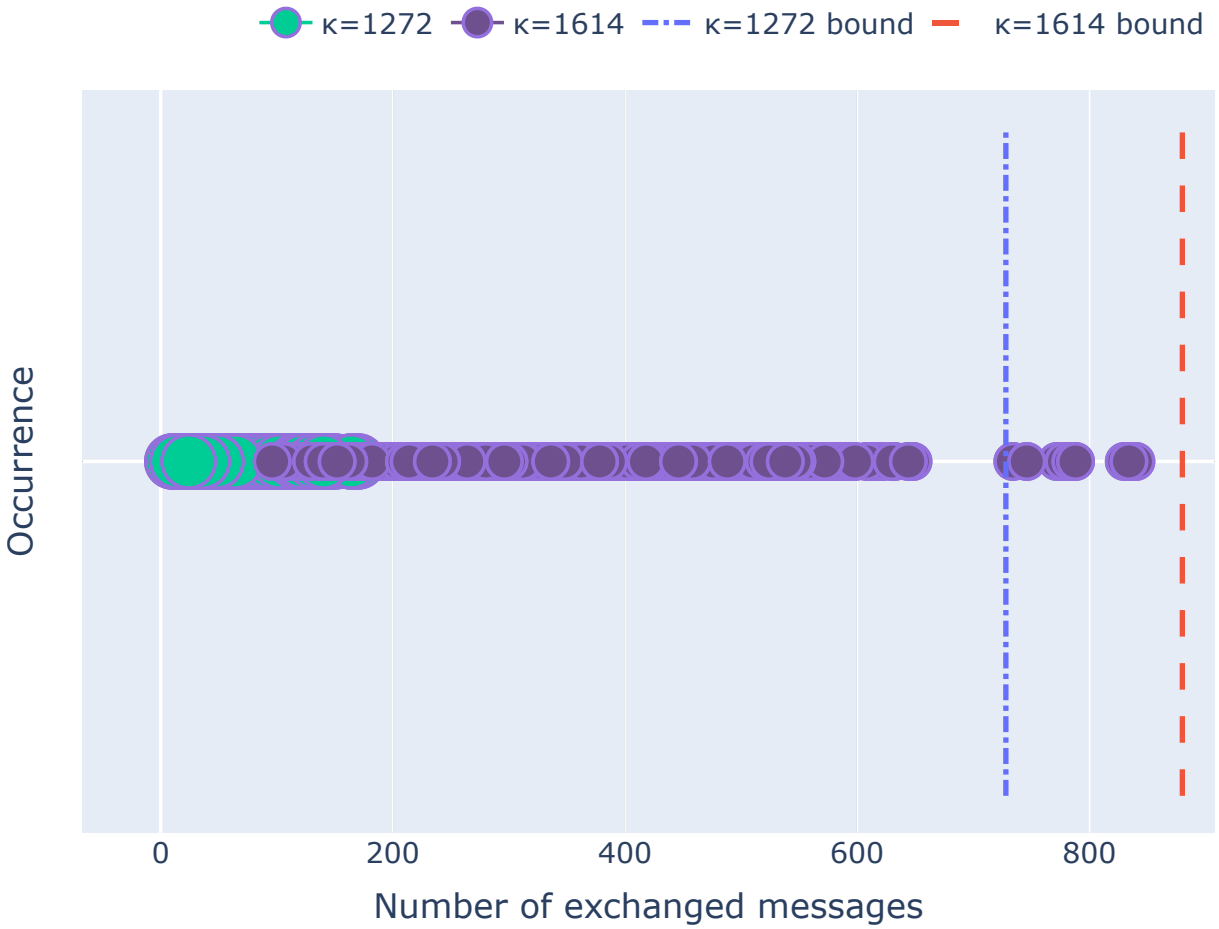

Figure 4.14: Comparing the actual number of generating messages with the analytical bound on communication complexity. Markers (dots) mark one or more gatherings that ended with the corresponding number of exchanged messages.

Experiment 5

Research Question 9. How efficient is the execution of the Aurora module in a scenario where the Aurora node is resource bound, when applied to a realistic network with malicious nodes?

Methodology: The simulation setup and methodology is the same as in Section 4.4.4, with two key differences. First, Const. 1 must be satisfied — if the Aurora node is unable to respect the given constraint, it will not construct a probabilistic honest set and will consequently halt. Second, the halting condition is changed to take into account the threshold for the average number of unique nodes per draw, which is set to 15, but only if 10 or more draws were previously made.

Results: The results shown in Fig. 4.15 confirm that the Aurora module operates within the expected success rate with good progress while the number of reachable nodes which are malicious is below one quarter.

Research Question Answer 17. [Answers RQ. 9] The Aurora module behaves as expected and identifies an honest node for transient communication (i.e., construct a Π_p) in 99.9% of the trials, while halting consistently when less than a quarter of nodes in Γ are malicious.

Thus, we conclude that the Aurora module is adequate for resource-constrained devices when similar network conditions can be expected.

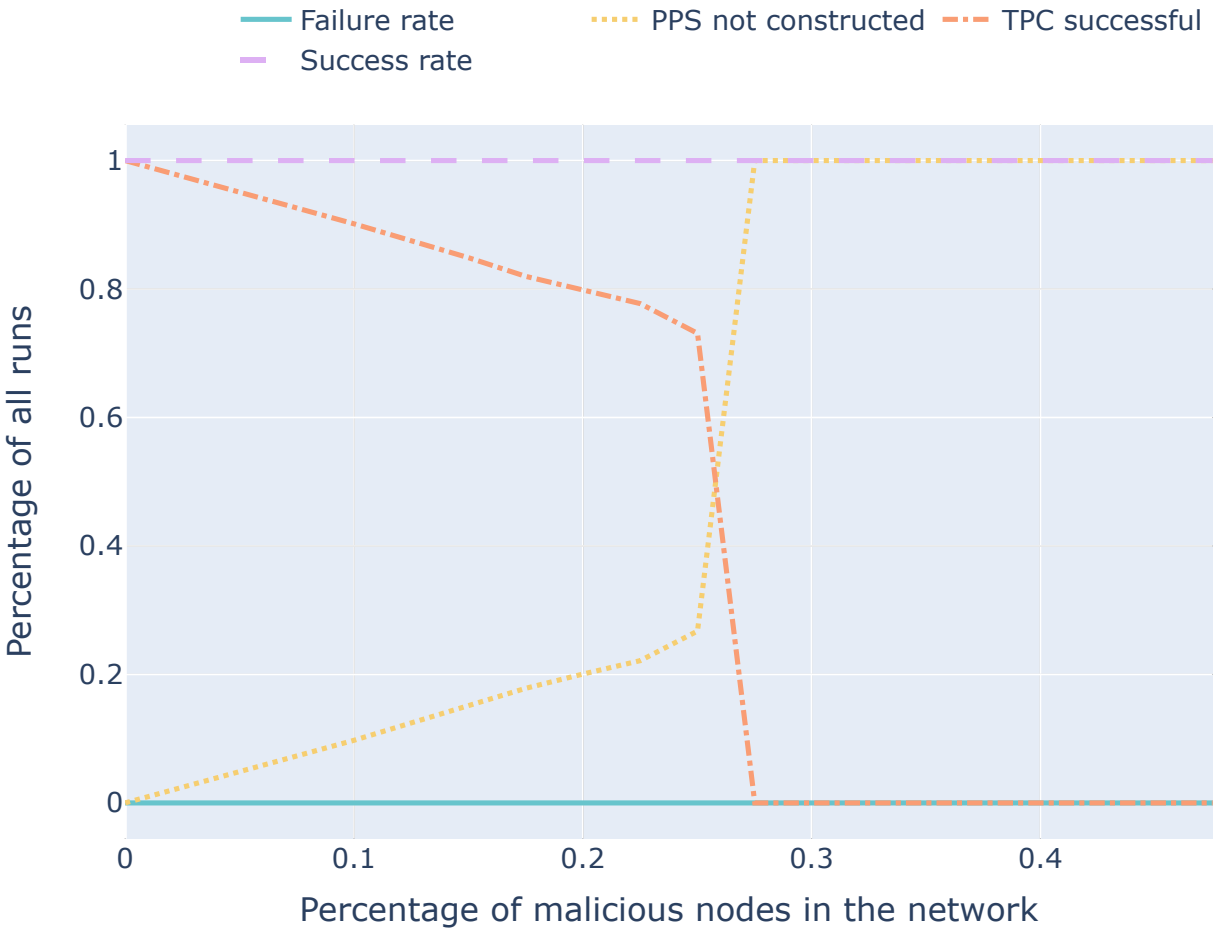

Figure 4.15: Outcomes of the experiment in a realistic setup when Const. 1 is satisfied and the average message size threshold is set to 15, while increasing the malicious number of network nodes.

4.5 Aurora in an open source ETH client

As the second step of our evaluation procedure, we address five specific research questions in this section:

Research Question 10. Are the capabilities of prominent existing DLT protocols such as ETH developed enough to support the integration of Aurora algorithms?

Research Question 11. Can an Aurora module operate in a production DLT network?

Research Question 12. How much time is required to execute a gathering while operating in a production DLT network?

Research Question 13. How much storage and RAM does an Aurora module require while operating in a production DLT network?

Research Question 14. How can the initialization of κ be abstracted from the end user?

We answer these research questions by implementing an older and depreciated version of Aurora using an open source ETH client *Trinity* written in the object-oriented and interpreted programming language *Python*, and suggest a rudimentary user interface to showcase how direct initialization of κ can be abstracted from the end user.

4.5.1 Depreciated Aurora version

By the *Depreciated Aurora version*, we refer to the solution implemented in [4, 25]. By the *Current Aurora version*, we refer to the set of algorithms introduced in Section 4.3, meaning Algorithm 2, Algorithm 4 and Algorithm 5. The Depreciated Aurora version and the the Current Aurora version share relevant features that allow us to provide answers to the following research questions: RQ. 10, RQ. 11, RQ. 13 and RQ. 12. The conclusions drawn in this section are applicable to both the Depreciated Aurora version and the Current Aurora version.

Similarities: The main similarities between the Depreciated Aurora version compared to the Current Aurora version are as follows:

1. Both solutions in essence execute a gathering to discover network nodes by the exchange of ping and pong messages.
2. Both solutions define a halting condition.
3. Both solutions exchange status request, status response, proof request and proof request messages at the end of a gathering.
4. In both solutions, a gathering can be associated with a DAG consisting of nodes contacted during a gathering.

Differences: The main differences (listed for completion) between the Depreciated Aurora version compared to the Current Aurora version are as follows:

1. The Depreciated Aurora version does not construct probabilistic honest sets, but evaluates the confidence (i.e., the level of trust) in a node at which the last draw was executed (and thus the gathering ended) by evaluating pong messages after each draw.
2. Although the Depreciated Aurora version relies on *hypergeometric experiments*, the Depreciated Aurora version does not give a clear probabilistic output, but gives a derived value $z \in \mathbb{R}$; $0 \leq z \leq 1$, which is an estimate of the correctness of the Depreciated Aurora version output. A value of 1 indicates that there is no suspicious activity and a value of 0 indicates that there is enough suspicious activity to halt the execution of the client.

3.The halting condition in the Depreciated Aurora version was reached when the z reached 0 or a maximum number of steps was executed, which was calculated by Markov chain analysis. The Current Aurora version has a more complex halting condition, as explained in Section 4.2 that can be modified at runtime depending on the actual DLT network.

4.The Depreciated Aurora version is more tightly coupled to the underlying network topology compared to the updated solution, since the construction of Π_h in the Current Aurora version is decoupled from the topology, whereas the Depreciated Aurora version depends on the network topology in every step of the execution.

4.5.2 Ethereum network protocols

To understand how messages specific for the Aurora algorithms (ping, pong, status request, status response, proof request and proof response) integrate in the ETH ecosystem, we will provide a brief overview of the relevant ETH network protocols.

devp2p devp2p is a set of network protocols which form the ETH P2P network. Amongst others[‡‡], the relevant protocols discussed hereinafter will be *Node Discovery Protocol (NDP)*[*], *RLPx Transport Protocol (RLPx)*[†], *Ethereum Wire Protocol (ETH)*[‡] and *Light Ethereum Subprotocol (LES)*[§].

NDP: This *User Datagram Protocol (UDP)*-based protocol used for node discovery relies on a *Distributed Hash Table (DHT)*, which is a derivative of the P2P DHT Kademlia [83], to efficiently organize a distributed index of nodes. The protocol specifies messages relevant to the maintenance of the P2P network. In the context of our solution, the relevant messages are the *FindNode (0×03)* corresponding to an ping message. Similarly, the *Neighbors Packet (0×04)* is the equivalent of a pong message.

Storage of remote nodes on the host node: ETH remote nodes are stored on the host node using the *Ethereum Node Records (ENR)* open format for P2P connectivity information[¶]. The table consists of at most 255 rows. Each row contains at most l nodes. The network parameter l is not set globally, but is usually $l = 16$. The position of a remote node in the host's table row is determined based on the *distance* of the remote node from the host node. The distance between nodes in ETH is calculated as the bitwise XOR of the public key hashes and interpreted as a

[‡‡] https://github.com/ethereum/devp2p/
[*] https://github.com/ethereum/devp2p/blob/master/discv4.md
[†] https://github.com/ethereum/devp2p/blob/master/rlpx.md
[‡] https://github.com/ethereum/devp2p/blob/master/caps/eth.md
[§] https://github.com/ethereum/devp2p/blob/master/caps/les.md
[¶] https://github.com/ethereum/devp2p/blob/master/enr.md

number. When a remote node is queried with a *lookup* request from a host node, the remote node responds with a node list consisting of a row from its table corresponding to the host node[‖].

Recursive lookup: This is an iterative peer discovery process used by ETH nodes over the NDP protocol. When a host node initiates the recursive lookup, the host node selects r nodes closest to the remote node as determined by the distance parameter. The host node then sends a *lookup* request to the remote node. After retrieving the remote node's responses, the host node iteratively sends a *lookup* request to r nodes discovered using the remote node, but only to nodes that have not yet been queried. This continues until no more nodes can be found to populate the DHT table with newly discovered nodes. When a new node joins the network, it runs a *boot process*, a *recursive lookup* process initiated with bootnodes, to populate its DHT table with remote nodes.

RLPx A *Transmission Control Protocol (TCP)* based protocol used for communication between ETH nodes. An RLPx connection is established by creating a TCP connection between two ETH nodes and agreeing on an ephemeral key for further encrypted and authenticated communication. The process of creating these session keys is called *Initial Handshake*. The protocol also serves as the basis for other application-level protocols specified in devp2p.

Ethereum Wire Protocol ETH is a RLPx-based application-level protocol[**] and is used by full nodes. Nodes running the protocol store the entire ledger and participate in maintaining consensus. The protocol processes messages relevant to ledger synchronization, block propagation and transaction exchange. In the context of our solution, the relevant messages defined by this protocol are the initialization of a an ETH connection, which corresponds to a status request message, and the *Status (0×03)*, which corresponds to a status response message.

Light Ethereum Subprotocol LES is an application-level RLPx-based protocol used by light clients. As described in Chapter 1, light clients only download the header chain as it appears, and other parts of the ledger as needed. They do not participate in maintaining consensus (i.e., mining). In addition to the ETH protocol, full nodes can also support the LES protocol to serve light clients. In the context of our solution, the relevant messages defined by this protocol are the initialization of a LES connection corresponding to a status request message, the *Status (0×03)*, corresponding to a status response message, the *GetProofsV2 (0×0f)* corresponding to a proof request message, and the *ProofsV2 (0×10)* corresponding to a proof response message.

[‖]https://github.com/ethereum/devp2p/blob/master/discv4.md

[**]Note that the acronym for the wire protocol ETH is the same as the acronym for the Ethereum network and the exact meaning of the acronym must be derived from context

Trinity is able to run in two different modes of operation, namely by running the ETH protocol and the LES protocol [84]. Table 4.2 indicates whether messages required by Aurora algorithms are supported by ETH and LES. Since the Trinity client running the ETH protocol does not support the proof request and proof response messages, the verification in a production environment must be done with the client running the LES protocol.

Table 4.2: Mapping of Aurora algorithms requirements to ETH and LES capabilities.

Description	Aurora algorithms	LES v2	ETH v63
Peer list	ping, pong	✓	✓
Status	status request, status response	✓	✓
Merkle proof	proof request, proof response	✓	✗

Research Question Answer 18. [Answers RQ. 10] The capabilities the ETH protocol is developed enough to support the integration of Aurora algorithms, which can be done in a modular fashion.

4.5.3 Ethereum Trinity DLT client

DLT client of choice: Having in mind that the focus of this paper is on light clients, we have devised the client selection criteria accordingly. The DLT client was selected based on three main requirements. First, the DLT network of choice must be distinguished, well maintained, and documented. Second, the client must have existing light client capabilities. Third, the client must be easily extensible and allow modular integration of our solution. The official ETH reference implementations at the time of writing were: *Aleth* written in C++[††], Trinity written in Python[‡‡], and geth written in go[*]. Both geth abd Trinity provide the ability to run as light clients. Given that the ETH client Trinity[†] fulfilled all the above requirements at the time of implementation[‡], due to its expressive code base, developer ecosystem, popularity and low entry barrier of the Python programming language, it was chosen as a suitable client for integrating our solution.

The architecture of the solution is displayed in Fig. 4.16. The *Main Application Process* starts the *Database Process*, the *Networking Process* and the *Component Process*. The Database

[††]Aleth: `https://github.com/ethereum/aleth`
[‡‡]Trinity: `https://github.com/ethereum/trinity/`
[*]geth: `https://github.com/ethereum/go-ethereum`
[†]Trinity v0.1.0-alpha.34 'Caroline Herschel', commit SHA 7ebac95d
[‡]The project was officially discontinued on 01.07.2021

Process executes operations related to the blockchain and uses LevelDB by default. The Networking Process is related to P2P communication. The *Component Process* spawns isolated components. All components communicate via the the *Lahja* event bus. Lahja is a generic event bus implementation that enables non-blocking asynchronous lightweight inter-process communication[§]. Data such as remote peer information, block information and transaction proofs are shared over the event bus. The Trinity client provides an API, called *Component API*, which allows the creation of components for modular extension of existing system functionality as part of the Component Process. The Aurora Component represents how our solution can be integrated into the client.

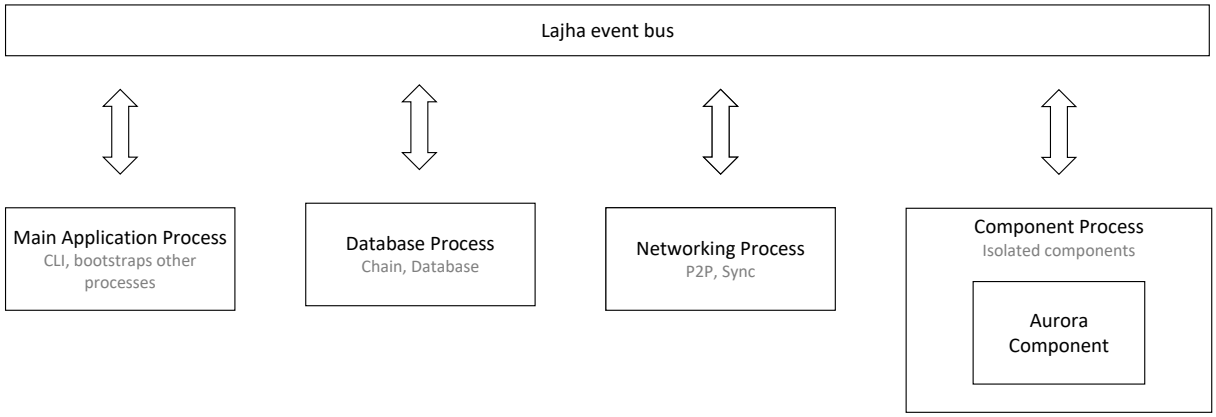

Figure 4.16: High-level representation of communication between Trinity components.

Implementation consolidation: In order to extend the Component API to integrate our solution with the Trinity client, the *proxy peer pool* component was modified. This component is to be used by processes to access and interact with peer pool data. At the time of integrating our solution with Trinity, the Trinity client was refactored and switched from using the *asyncio* concurrency library to the (*trio*) concurrency library, which was not interoperable with asyncio. Since the discovery component was based on a *trio* service while the *proxy peer pool* component was based on an *asyncio*, we resorted to a temporary workaround. Our solution was not implemented as an independent component, but as part of the discovery component logic. The driving factor behind this decision was the tight coupling between the service for peer discovery and the discovery component (i.e., it was not trivial to isolate the service for peer discovery and the discovery component). As a result, the Aurora module is tightly coupled with the core code of the Trinity client. However, should the Trinity API, stabilize in the future, it will be possible to decouple the Aurora module from other Trinity components. This means that the Aurora module can be added by installing a Python package[¶].

[§]https://github.com/ethereum/lahja

[¶]`https://trinity-client.readthedocs.io`

4.5.4 Experiments on the ETH mainnet

Experiment 1

Methodology: Since ETH does not support the proof request and proof response messages (see Table 4.2), we focus on using the LES protocol. To better understand the capabilities of nodes on the mainnet, we ran the Trinity client in LES mode. In total, 4807 connections were attempted with remote nodes in LES mode and the outcomes were classified.

Results The connection attempts shown in Table 4.3 have produced predominantly two classes of responses. First, 3088 or $64,24\%$ attempts were rejected because the Aurora node and the remote nodes did not have matching capabilities (i.e., the remote node did not support the LES protocol). These findings conicide with the findings in [84]. Second, 1061 or $22,07\%$ attempts were rejected by remote nodes because the remote peer had already reached the maximum number of nodes allowed. For example, and at the time of writing, in Trinity the default maximum number of nodes is 25[‖] while the default value for the geth client (ETH client written in the programming language go) is 50.[**]. Hence, we are able to state the following:

Research Question Answer 19. [Answers RQ. 11] The ETH mainnet as of yet does not sufficiently support the LES protocol, therefore the Aurora module is not ready to be used on the mainnet for TPC.

Experiment 2

Methodology: The time to send a ping request and receive a pong response was measured on a sample in the mainnet and the measured average average measured is $7s$. Given that the average the average number of nodes in a pong message is 16[††], we can use expression:

$$y(|\Gamma|) = \frac{|\Gamma|}{16} \tag{4.28}$$

which describes the minimum amount of draws required to construct a set of unique nodes Γ of size $|\Gamma|$ as a function of $|\Gamma|$, and the expression:

$$y(|\Gamma|) = \frac{|\Gamma|}{16} \times \frac{7}{60} \tag{4.29}$$

which describes the minimum amount time in minutes required to construct a set of unique nodes Γ of size $|\Gamma|$ as a function of $|\Gamma|$. Fig. 4.17 shows both the minimum number of draws and the corresponding time in minutes required to construct Γ of a given size. To illustrate:

[‖] https://github.com/ethereum/trinity/blob/master/p2p/constants.py#L113
[**] https://geth.ethereum.org/docs/interface/command-line-options
[††] https://github.com/ethereum/devp2p/

Table 4.3: Results of LES connection attempts with peers in the mainnet ETH network.

Result	LES
NoMatchingPeerCapabilities	3088 (64.24%)
HandshakeFailureTooManyPeers	1061 (22.07%)
TimeoutError	402 (8.36%)
UnreachablePeer	169 (3.51%)
HandshakeFailure	49 (1.01%)
WrongNetworkFailure	24 (0.50%)
PeerConnectionLost	12 (0.25%)
Successful	1 (0.02%)
MalformedMessage	1 (0.02%)
WrongGenesisFailure	—

Research Question Answer 20. [Answers RQ. 12] If the Aurora node is to discover 6356 nodes[‡‡] through the duration of a gathering, the minimum duration of a gathering should be approximately 47 minutes. Given that our solution is intended for resource constrained devices such as mobile phones, this value is not negligible, and ways to reduce the duration of a gathering are discussed in Chapter 6.

Experiment 3

Methodology: To quantify the amount of storage and RAM used by the Trinity client enhanced with our solution, we added a new Trinity client to the ETH mainnet and performed gatherings until the client discovered at least a 1000 unique nodes. We then measured the amount of RAM used by the Trinity client in LES mode, and the amount of RAM and storage used by the Aurora module itself with a memory profiler for Python.

Results: Fig. 4.18 shows the amount of RAM the Trinity client stared as a light client consumes as a function of time. In particular, the client requires approximately 800 MB of RAM. Fig. 4.18 shows the amount of RAM and storage Aurora module itself consumes when the Trinity client is stared as a light client a function of time. Starting from approximately 0.31

[‡‡]the size of the network slice used for experiments in Section 4.4

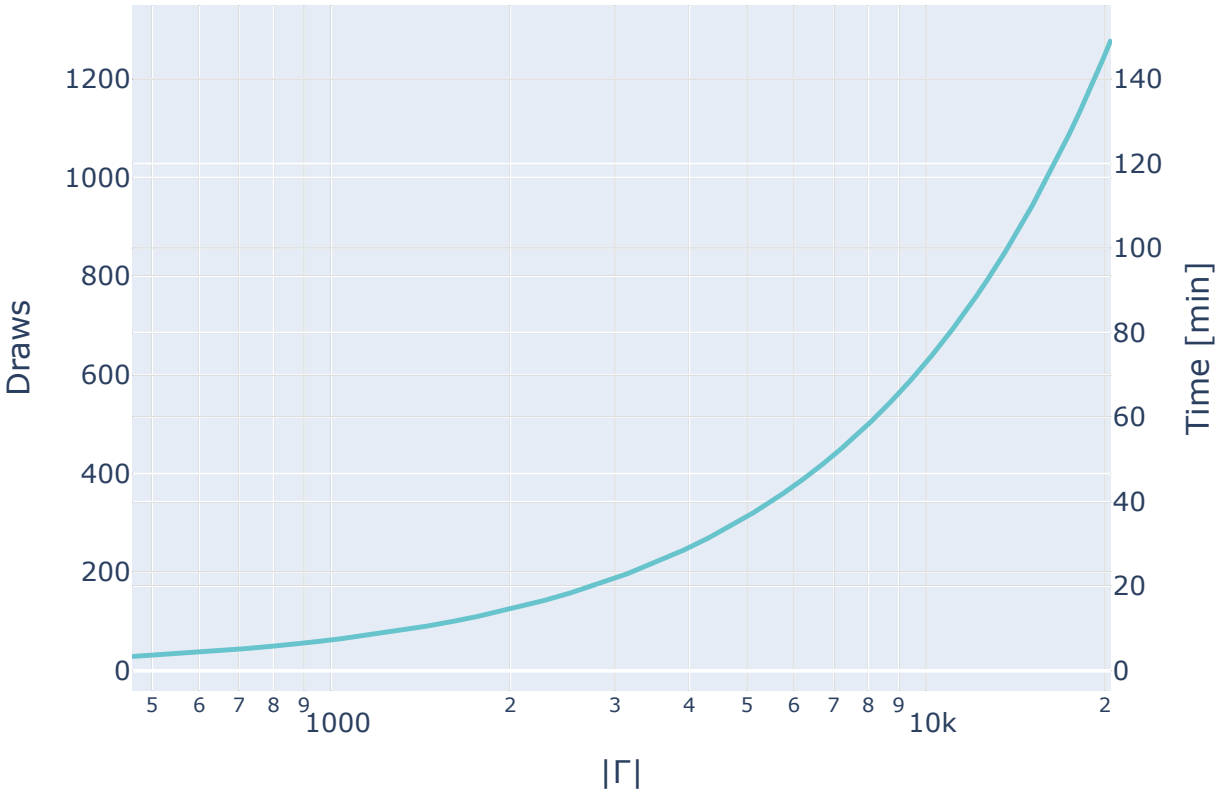

Figure 4.17: The number of draws and the time required to discover Γ nodes.

MB when execution begins and the client has not yet discovered any remote nodes, memory consumption increases to about 0.7 MB when 1000 nodes are discovered. When linear regression is applied to the graph shown in Fig. 4.19, we obtain an expression describing memory consumption in MB as a function of the number of draws:

$$y(draw) = 0.00565 * draw + 0.307 \tag{4.30}$$

Given that the number of Ethereum nodes discovered on 27.02.2022.* was 2552. Assuming 16 peer identifiers are discovered per draw, 2552 nodes are collected in 160 draws, using the above expression, our client would consume approximately 1.2 MB.

Comparison to other ETH clients: Table 4.4 compares the Trinity client enhanced with the Aurora module to other ETH clients [4]. When comparing our solution with geth (fastsync), geth (light), Trinity (full) and Trinity, our solution requires significantly less memory, which is its main advantage. When comparing our solution with BlockQuick, BlockQuick requires additional data structures to be added to the ledger. This essentially requires introducing disruptive, backwards incompatible changes to the network, while our solution does not require backward incompatible changes. On the other hand, compared to BlockQuick, our solution consumes more storage and RAM. When comparing our solution with Infura, our solution requires more

*https://etherscan.io/nodetracker

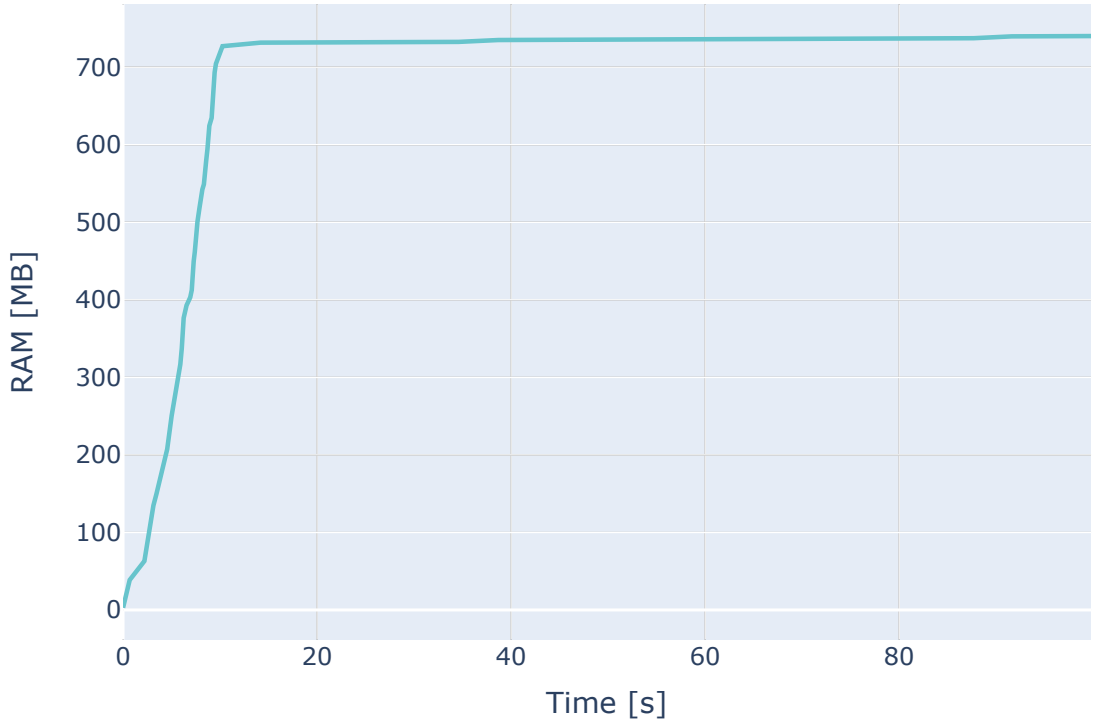

Figure 4.18: Memory consumption of the Aurora node in light mode measured with memory-profiler for Python

Table 4.4: Approx. specifications for the Trinity Aurora node compared to various ETH clients.

Client	Trust	Storage	RAM	breaking changes
Trinity (full)	Trustless	>7334 GB	∼2 GB	no
Trinity (light)	Trustless	>5 GB	∼800 MB	no
Trinity Aurora	Trustless	∼1 MB	∼800 MB	no
geth (light)	Trustless	>5 GB	∼150 MB	no
geth (full)	Trustless	>7334 GB	∼1 GB	no
BlockQuick	Trustless	∼20 KB	∼50 KB	yes
Infura	Gateway (Trusted)	∼4 KB	∼10 KB	no

storage and significantly more RAM . However, our solution is trustless unlike Infura, which is a trusted solution. In summary:

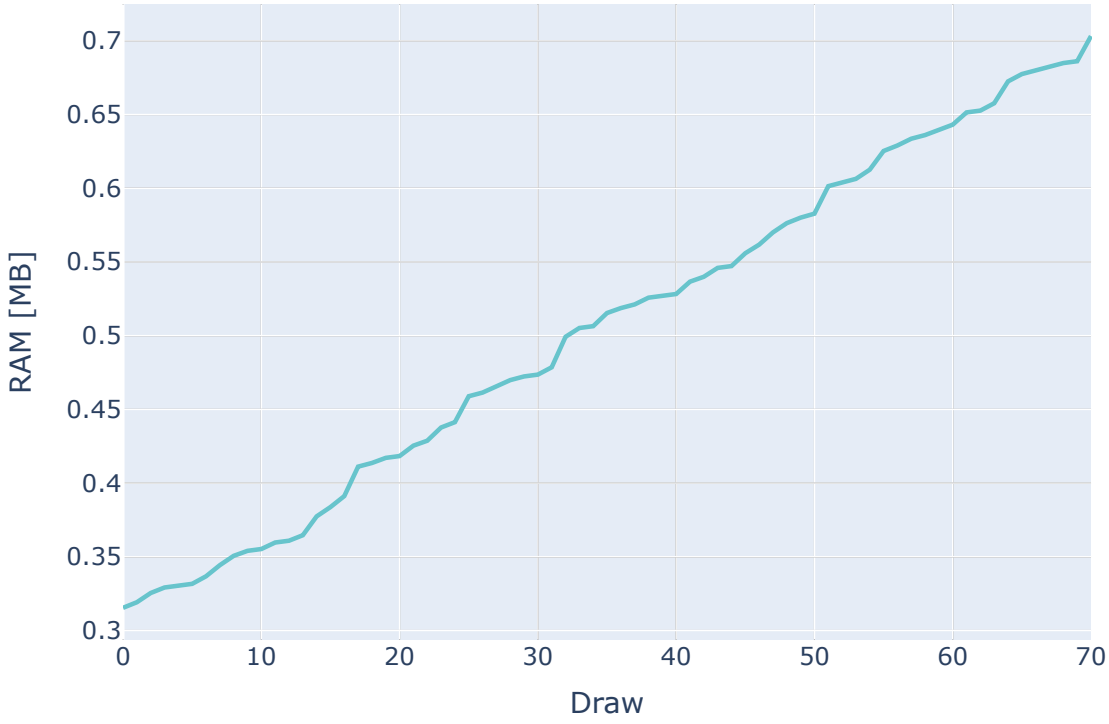

Figure 4.19: Memory consumption of the Aurora module measured with memory-profiler for Python

Research Question Answer 21. [Answers RQ. 13] The PoC Aurora module consumes approximately 1.2 MB and as such can run on constrained devices [85] with as little RAM as the Raspberry Pi 2 Model B (1 GB RAM)[†].

4.6 The user interface

In Section 4.3 we have specified the necessary parameters for the execution of Aurora. We also discussed the proposed initial parameters and pointed out that initializing κ is non-trivial, which opened RQ. 14. In search of an answer to this question, we design a user interface that uses time as an abstraction. We are using ETH as the underlying DL and assuming that the user interface is designed for a light client.

4.6.1 Client initialization

When a user starts an Aurora node for the first time, the first contact node fc must be specified, as shown in Fig. 4.20. Note that this node need not remain static, i.e., the user can update such

[†]`https://www.raspberrypi.org/products/raspberry-pi-2-model-b/`

a node or even use multiple nodes. These functionalities have been omitted in this example for simplicity.

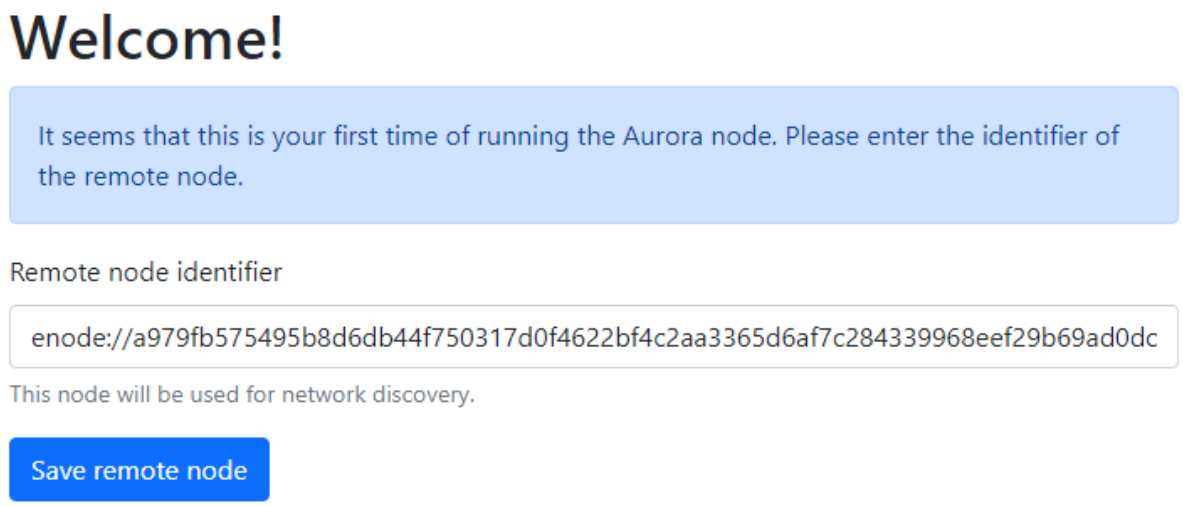

Figure 4.20: Aurora node user interface — initialization.

4.6.2 Gathering execution

After saving the remote node, execution of a gathering can begin, as indicated in Fig. 4.21. The user must wait for a time interval, which in this example is set to 5 minutes. During this time, the gathering discovers the network and extends the set Γ.

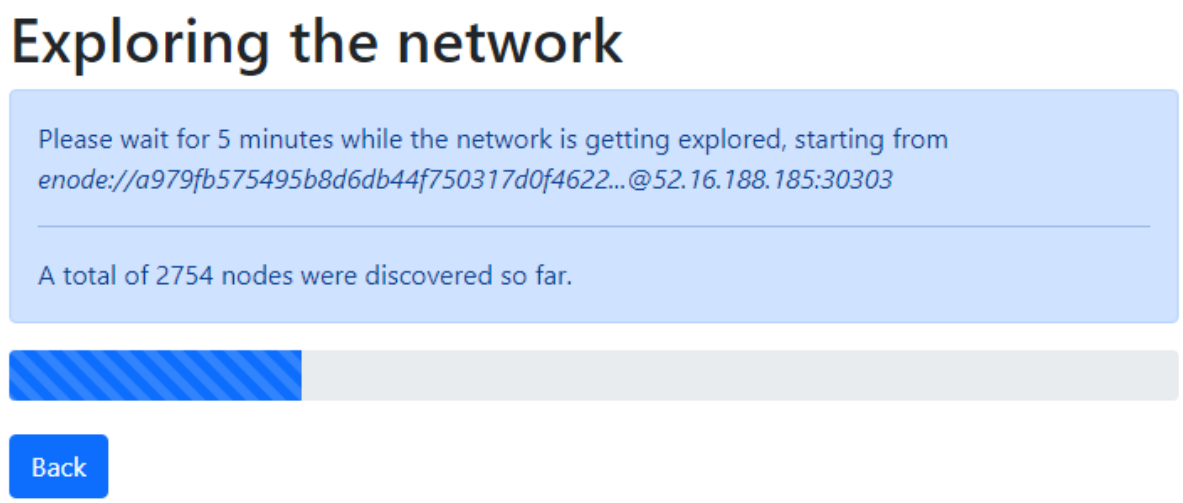

Figure 4.21: Aurora node user interface — gathering execution.

4.6.3 PSH overview

After the 5 minutes have passed, the user is presented with the results of the gathering as displayed in Fig. 4.22. Here we assume that 6356 nodes were discovered in 5 minutes. Note that the actual time interval would be around 47 minutes (as discussed in Section 4.5.4), when no optimization has been done in favour of execution time reduction. A more detailed analysis of execution time reduction is given in Chapter 6. The user can opt to commit on the results (which will create a Π_s containing 76 nodes which is tolerant to 5807 malicious nodes and a Π_p containing 41 nodes which is tolerant to 1741 malicious nodes. These honest sets will then be used for future interactions. The user can also opt to continue the gathering, which will take more time, potentially increase the size of the honest sets, but also increase the tolerance to malicious nodes. Thus, direct initialization of κ has been abstracted from the user (RQ. 14).

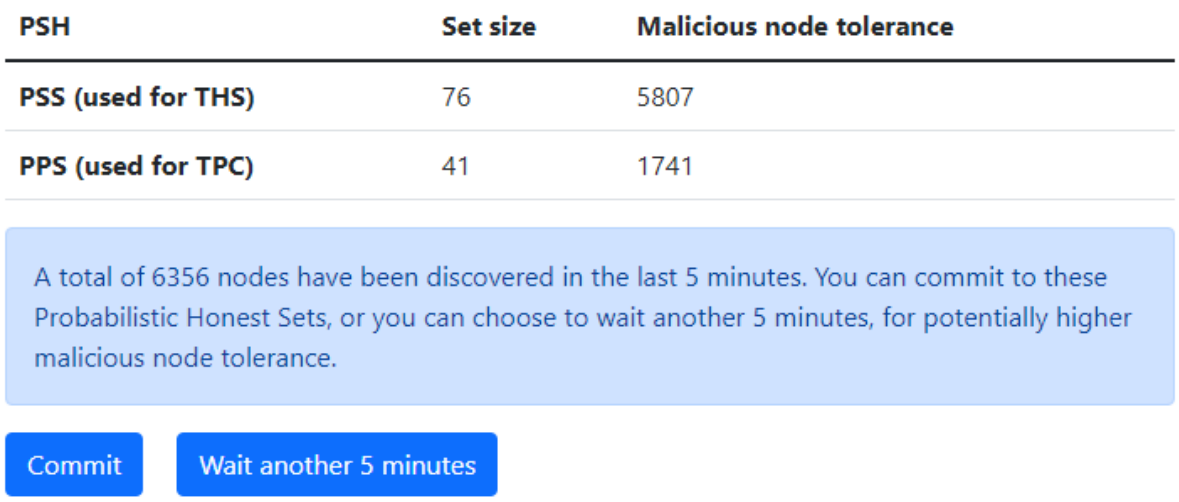

Figure 4.22: Aurora node user interface — Π_h confirmation.

After creating both Π_s and Π_p, the user can start the THS or verify the status of a transaction, as shown in Fig. 4.23. The value **PSS (used for THS)** represents $|\Pi_s|$, while the value **PPS (used for TPC)** represents $|\Pi_p|$. Both values can be extracted from Fig. 4.5[‡]. The user interface also shows some potential features, such as the ability to recreate already generated sets or even share the generated sets with trusted entities (family members, etc.).

4.6.4 Transaction history synchronization

Should a user decide to start the THS, members of Π_p can be queried as described in Algorithm 4, where the client will attempt to download the ledger and verify its state. If any discrepancy is found within the ledger, the problematic remote node is skipped (see Fig. 4.24), and

[‡]Table 4.1 was generated in the same manner

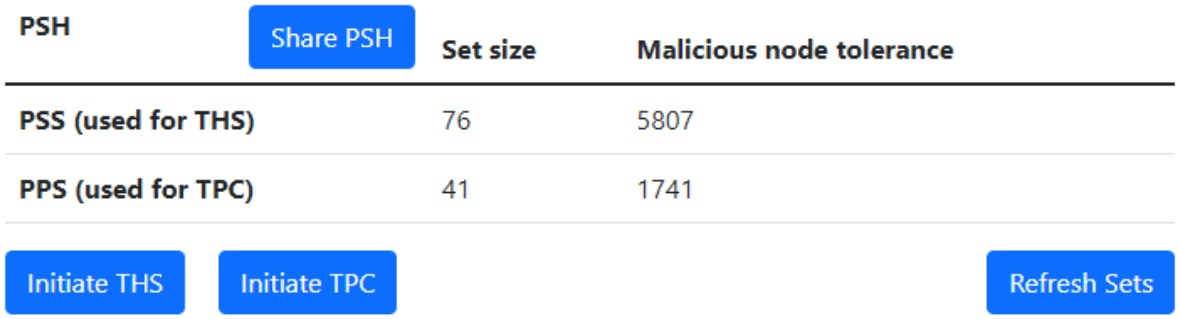

Figure 4.23: Aurora node user interface — Π_h are constructed.

another node will be queried until an honest remote node is eventually identified with a correct ledger. Once the THS has completed, the user is notified. The user may still want to check the status of a transaction. However, a Π_p is no longer necessary in this case, since the ledger has been downloaded and can be easily inspected locally (see Fig. 4.25).

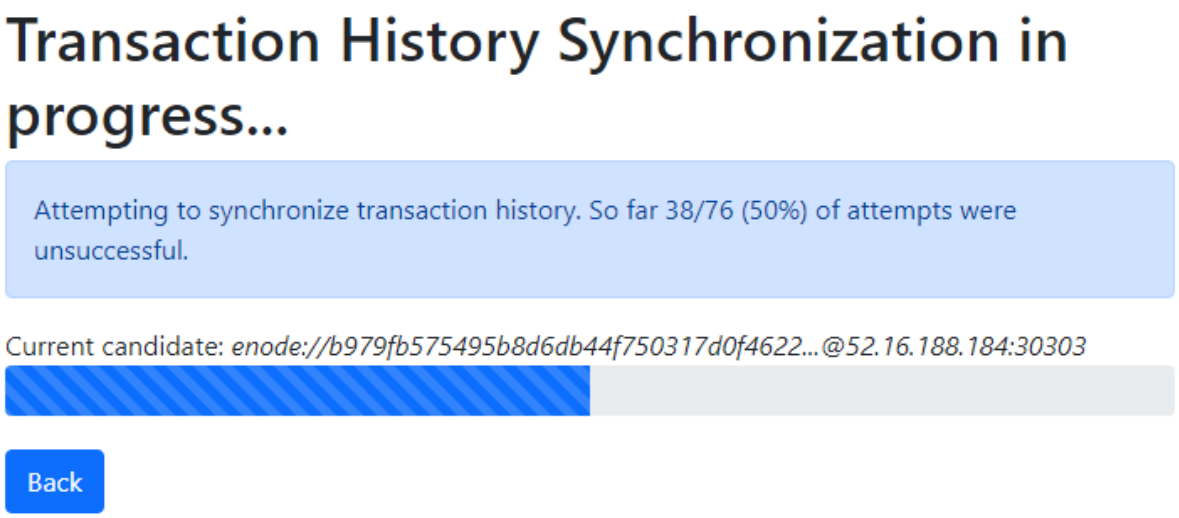

Figure 4.24: Aurora node user interface — THS attempt.

4.6.5 Transaction presence checking

Should a user decide to check the presence of a transaction, three input parameters are required: the transaction identifier tx_{id}, the block identifier blk_{id}, and the Merkle root. Fig. 4.26 displays an example of such data for ETH. Once the required parameters have been set, TPC can commence, as displayed in Fig. 4.27. Members of Π_p are queried to submit the Merkle proof for the transactions (as specified in Algorithm 5), and their responses are reviewed. In the given example, 7 out of 41 members of Π_p have been queried. Out of the 7 nodes, 4 have provided their Merkle proof, while 3 did not. The collection of votes continues until a majority of responses

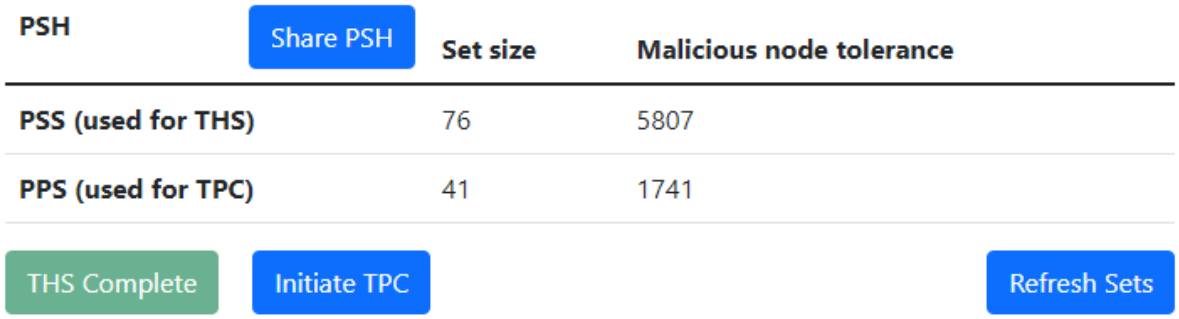

Figure 4.25: Aurora node user interface — THS complete.

have been collected (whether affirmative or negative). In our example, the conclusion is that the transaction is included in the ledger because the majority of the members in Π_p have responded with a valid Merkle proof (see Fig. 4.28). It was necessary to contact a total of 25 members of Π_p before the state of a transaction could be determined. Therefore, even if 1741 out of 6356 node detected by the client are malicious, the user can be sure (with the small probability of error which equals 0.001 in this example) that the transaction is present in the ledger.

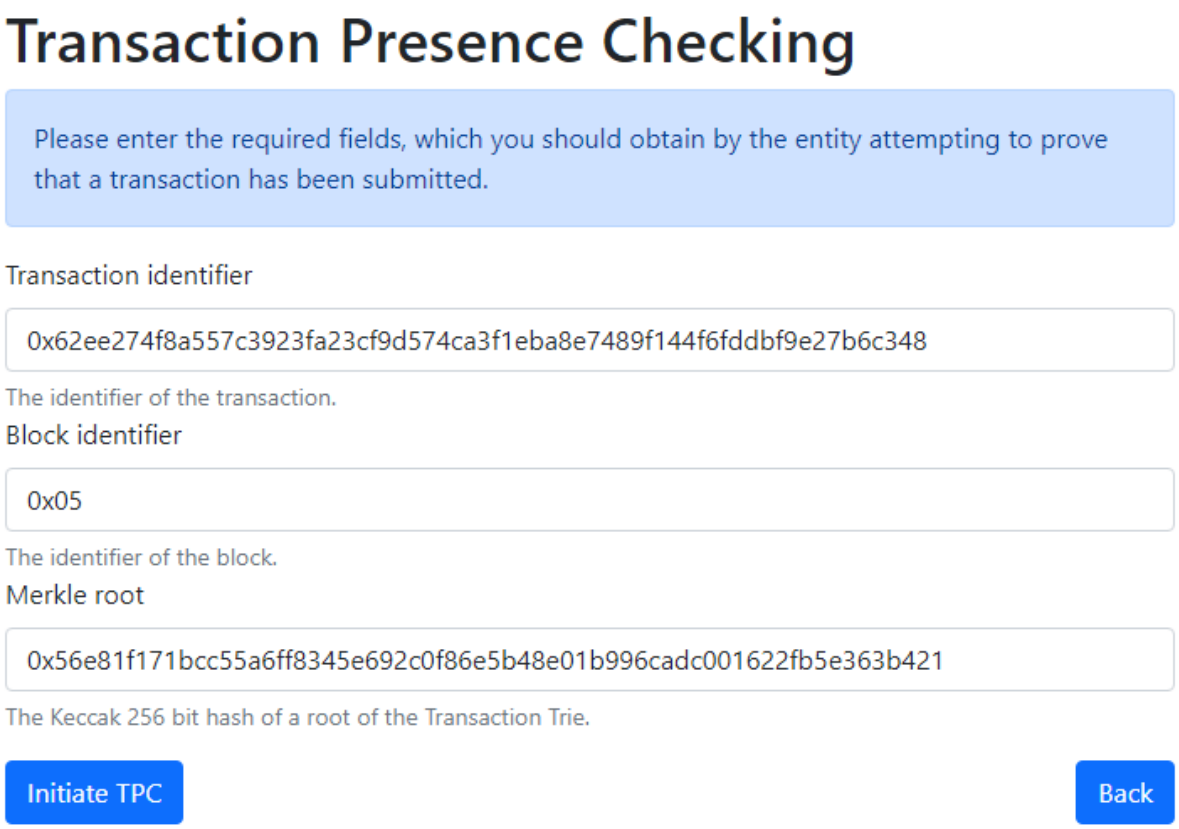

Figure 4.26: Aurora node user interface — TPC parameters initialization.

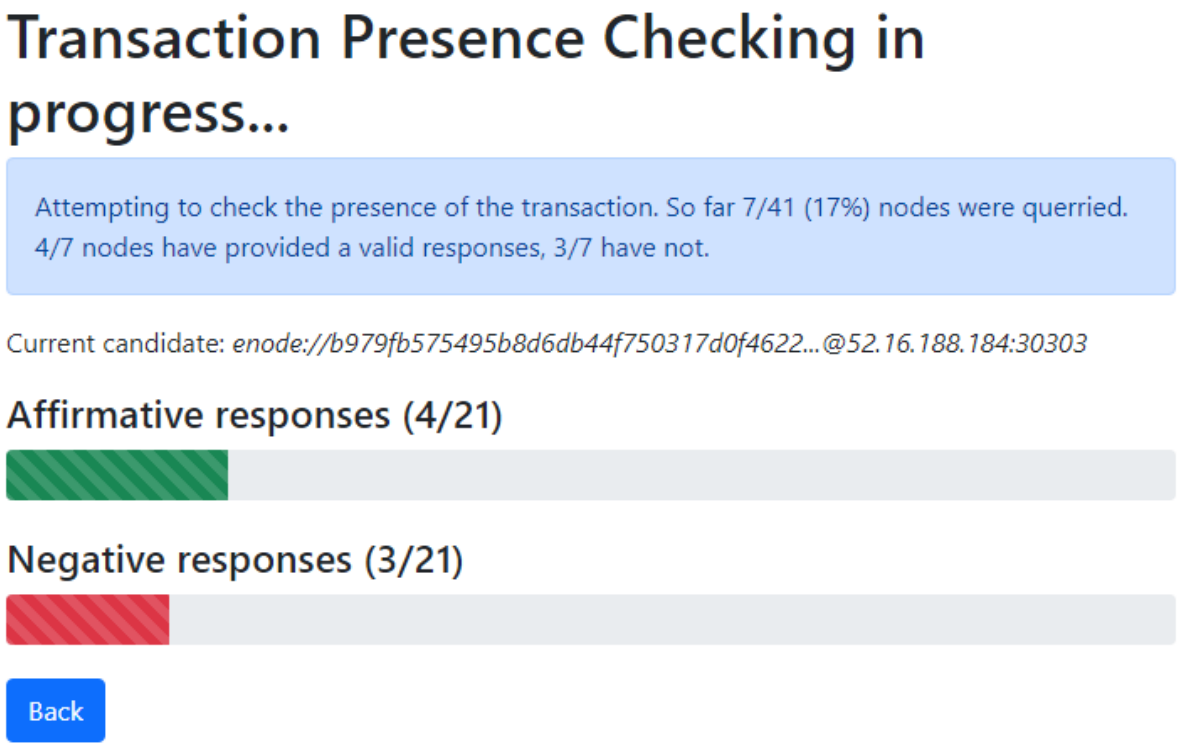

Figure 4.27: Aurora node user interface — TPC in progress

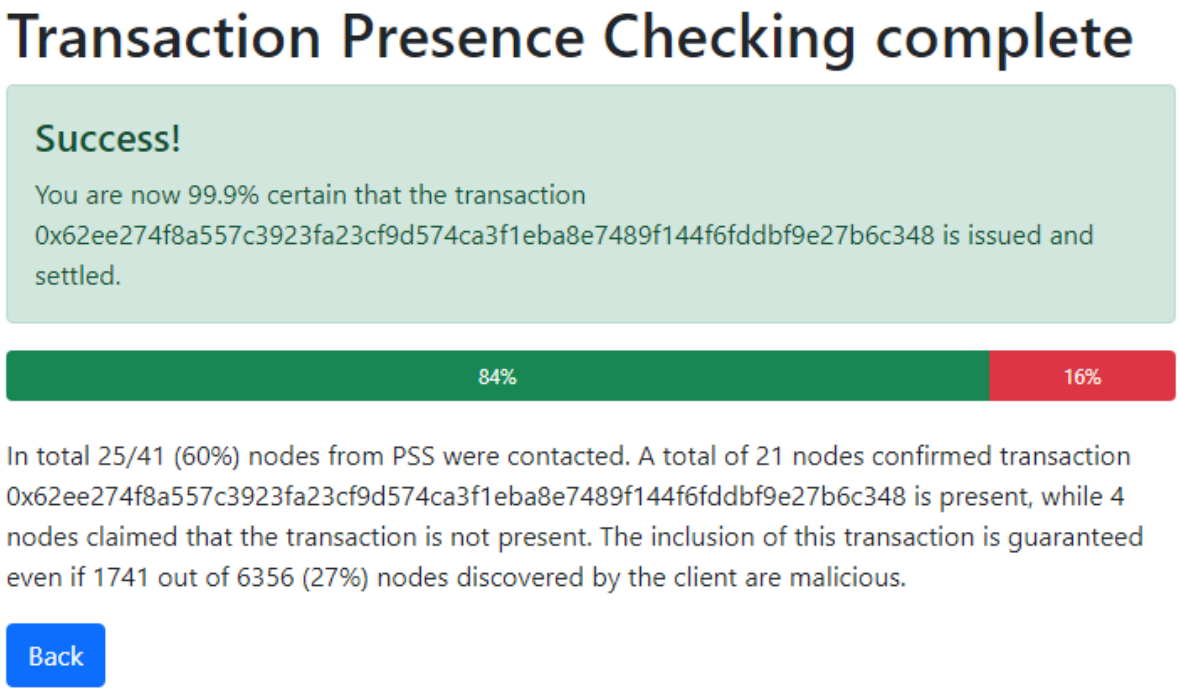

Figure 4.28: Aurora node user interface — TPC completed.

Chapter 5

Overview of scientific contribution

The main scientific contribution of this thesis is a set of consensus-agnostic probabilistic algorithms for blockchain networks designed to enable a new node to avoid the adverse influence of Byzantine or malicious nodes, as well as a procedure to evaluate the proposed solution using a case study on a selected distributed ledger and in a network with resource constrained nodes.

5.1 A new probabilistic honest set creation algorithm

The fundamental algorithm essential for the solution of Problem 1 and Problem 2 identifies a subset of network nodes containing a predefined number of honest nodes with a high and controllable probability. If the predefined number of honest nodes cannot be identified, the algorithm terminates. It is a prerequisite to the successful execution of both Algorithm 4 as well as Algorithm 5, since both algorithms rely on the presence of a probabilistic honest set generated by Algorithm 2. The algorithm is presented in Section 4.3.

5.2 A new probabilistic transaction history synchronization algorithm

For a DL network to be truly resistant to censorship, immutable, decentralized, and trustless, every node in the network should store a replicated version of the ledger and verify all of its contents. The process of synchronizing the transaction history (THS) is both resource intensive and to some extent centralized, as a new node joining the network implicitly trusts a set of known nodes that serve as entry points for the network discovery process (bootnodes). In their absence, the new node could fall under malicious influence.

There are several solutions that allow a new node to either resolve or mitigate the fact that the THS is resource intensive and relatively centralized. However, three problems have been iden-

tified with such solutions [25]. First, some of these solutions lose at least one of the interesting properties of DLT, e.g., resistance to censorship and trustlessness due to trusting a remote node. Second, some solutions assume the presence of honest bootnodes. Third, if a solution does not sacrifice any relevant DLT properties and does not assume the presence of honest bootnodes, its implementation requires changing the consensus rules and/or the DL data structure, which requires the introduction of backwards incompatible changes. In addition, methods for detecting anomalies in P2P networks have been widely explored, but such solutions have not been studied in the context of the DLT application domain, as they generally focus on identifying malicious nodes as opposed to finding honest nodes used for persistent communication in a DL network.

Aurora provides a holistic approach to the above problems by allowing a new node entering a DL network with κ malicious nodes to identify, with probability ρ, a subset of network nodes that contains a single honest node that can be used for the THS process, or signal the new node to cease operation. The solution relies on an inherent DL property, i.e., the entire ledger can be verified and any change to the ledger is eventually detected. It can be implemented in existing DL solutions without changing the consensus rules or the underlying data structure and is designed to operate when no bootnodes are available. Assuming that malicious actors expose manipulated data to a new node and thus waste resources and time (as opposed to honest nodes), our solution circumvents this issue by detecting an honest remote node from which a new node can synchronize the transaction history.

5.3 A new probabilistic transaction presence checking algorithm

Similar to the scenario in Section 5.2, if a DL node wants to check that a particular transaction is present in a ledger in a decentralized and trustlessness manner, it should synchronize transaction history, meaning downloading the ledger and verifying its entire contents, which is time and resource intensive. Resource constrained devices such as smartphones or IoT devices do not have enough memory or computational capacity to download and verify the ledger.

A resource constrained device can opt to delegate the storage and verification of the ledger to a remote node, and exchange only a subset of the ledger data with the remote node. The degree of delegation varies, where a resource constrained device can opt to completely trust the remote node thus sacrificing all the relevant properties of DLT, or it could partially trust the remote node and download a subset of the ledger, effectively compromising which properties to sacrifice. Such downloaded subsets of the ledger can also be significantly large relative to the storage available on a resource constrained device.

Aurora allows a new node in a network with κ malicious nodes to check, with probability ρ whether a particular transaction has been included in the ledger, or to signal the new node to

stop operating. This is done by identifying a subset of network nodes that contains a majority of honest nodes that can be queried about the presence of a transaction, where the presence is then inferred by a majority vote. The solution relies on an inherent property of DL, i.e., the transactions within blocks are present in an integrity validating structure that allows efficient checking whether a transaction is present in a block without downloading the entire block [15]. Our solution does not sacrifice any relevant DLT property, does not require the new node to synchronize transaction history (i.e., to download the ledger) or a subset of it (making it suitable for resource constrained devices), and can be implemented in existing DL solutions without changing the consensus rules or the underlying data structure.

5.4 A new evaluation procedure in a resource constrained environment

Our evaluation procedure was executed in two specific steps designed to address two main topics: first, the complexity and efficiency of our solution in the presence of malicious actors; second, the time and space complexity of our solution as well as its resource consumption. We relate the utility of Aurora algorithms to a case-study developed in our research which revolves around the use of DLT for the supply chain. The solution provides a novel, decentralized, privacy-preserving and verifiable management of a product during its lifecycle in the supply chain [1]. The case study was thoroughly analyzed and the possible manifestations of the problems identified in this work (Problem 1 and Problem 2) were clearly identified.

Topic 1 — Complexity and efficiency (Section 4.4) : The measurement of the complexity and the efficiency of our solution in a realistic DL network topology with an increase of malicious actors requires execution in a controlled environment where the network topology is known. Moreover, the process requires the presence of malicious actors in the network who are actively trying to subvert a new node. Here, we identified four specific problems. First, in public and permissionless DL networks, a complete network topology is not available. Second, nodes in such a network can leave and join the network at will, which means that the environment is very dynamic. Third, the introduction of malicious nodes into a production network is detrimental and doubtful. Fourth, recreating a production DL network as a private network is extremely resource intensive and considered impractical. Therefore, to verify our solution, we used a simulated network closely resembling the network topology of the BTC production network.

There are several DL network simulators that abstract features of a given DL network. However, at the time of writing, no simulators have been identified that have the capabilities relevant for the evaluation of our solution. In response to the above problems, we modified Simblock,

an existing open source simulation tool and extended it with features relevant to verify the proposed solution. Using the simulation tool, we measure the total number of messages exchanged before our solution terminates and the efficiency of our solution in a realistic DL network topology with an increase of malicious actors. The simulation results confirm our analytical findings.

Topic 2 — Resource consumption and compatibility (Section 4.5): Since our solution is intended to be applicable on resource constrained devices, it is imperative to measure the resource consumption of our solution in a realistic scenario (i.e., in a production DL network using an adequate DL client). To measure the resource consumption of our solution, we modified Trinity, an existing open source DLT client for the ETH network [4] and showcased that our solution consumes about 0.31 MB of RAM and 1 MB of storage at runtime. We identified existing capabilities of the ETH protocol and show that they are sufficient for the integration of our solution so that a client extended with the Aurora module can interact with existing ETH nodes. Furthermore, we propose further changes for the ETH mainnet protocols designed to improve the efficiency of our solution.

Chapter 6

Conclusions and future work

6.1 The main conclusions

In the context of this thesis, we explore public and permissionless DLT solutions, a new and groundbreaking technology that allows the maintenance of a decentralized ledger which is resistant to censorship, immutable, and eliminates the need for a trusted third party in a network. Network nodes can be malicious or Byzantine, they do not trust each other, their number is unknown, and they can join and leave the network at will.

Public and permissionless DLT solutions require a large amount of resources to operate, making it difficult for resource constrained devices to participate in the network. The focus of this thesis is to identify mechanisms that allow such devices to participate in the network without compromising on features relevant to DLT. We address and resolve two identified problems in the current state of DLT. First, the process of initial synchronization of transaction history is relatively centralized, as it depends on a set of known nodes that are both available and honest, which runs counter to the ethos of DLT. The absence of such nodes may have undesirable consequences, as it opens new attack surfaces on the new node. Second, should a client wish to check whether a particular transaction is present in a DL without downloading all or part of the ledger, the client must again compromise on the relevant DLT features mentioned above.

The original scientific contribution of this thesis in response to the above problems can be summarized as follows: first, a new probabilistic honest set creation algorithm (see Algorithm 2). The algorithm creates a subset of network nodes which contain a predefined amount of honest nodes with a predefined probability. Second, a non-deterministic algorithm for synchronizing transaction history in distributed ledger networks with malicious nodes to reduce resource consumption (see Algorithm 4). Third, an efficient algorithm for checking the presence of a transaction in a ledger (see Algorithm 5). Fourth, a procedure to evaluate the proposed solutions using a case study on a selected distributed ledgers (ETH and BTC) and in a network with resource constrained nodes is presented (see Section 4.4 and Section 4.5).

We provide a comprehensive summary of DLT features that are relevant to the implementation of our solution and justify the benefits of Aurora algorithms in the context of a separate project that provides privacy-preserving, verifiable, and decentralized management of a product throughout its supply chain lifecycle (DL-T).

At its core, the Probabilistic honest set construction algorithm, Transaction history synchronization algorithm and Transaction presence checking algorithm are designed to be easily integrated into existing DL clients without changing consensus rules or underlying data structures. In a DL network that contains κ malicious nodes, an Aurora node iteratively discovers a subset of network nodes from which another subset of nodes is subsequently sampled without replacement and tests whether this subset contains h honest nodes with at least probability ρ. We call this subset a probabilistic honest set or Π_h. If a Π_h can not be constructed, the client is signaled to halt operation. We have singled out two specific Π_h. First, a set that contains at least one honest node is used to select a remote node for transaction history synchronization. Second, a set containing a majority of honest nodes is used for TPC. The pseudocode of the Probabilistic honest set construction algorithm, Transaction history synchronization algorithm and Transaction presence checking algorithm, all necessary parameters and their suggested default values have been presented.

The total number of messages exchanged before an Aurora node generates a response was given (see Section 4.3.1), where the generation of a Π_h and communication with its members terminates after $2 * \left\lceil \frac{\omega * \kappa}{Z} \right\rceil + 2 * \lfloor \sqrt{\kappa} \rfloor$ messages have been exchanged and each subsequent use of the generated Π_h generates no more than $2 * \lfloor \sqrt{\kappa} \rfloor$ messages. The analytical expressions for time and space complexity were given, and compared to SPVs, which are the current *State of the Art (SOTA)* solution.

The procedure for evaluating the Probabilistic honest set construction algorithm, Transaction history synchronization algorithm and Transaction presence checking algorithm consisted of two parts. First, we measure the efficiency of our solution using the discoverable BTC IPv4 network slice as a case study and adapt the solution to be applicable and efficient in a resource constrained environment, meaning a Π_h can be constructed only if $|\Pi_h| \leq \sqrt{\kappa}$. To this end, we extended the Java-based discrete event-driven open source simulator Simblock with the functionality required to verify our client and measure the efficiency of the probabilistic variant of our solution against it's analogous deterministic variant. Second, we implement our solution on an open source ETH client and measure the client's resource consumption on the ETH production network.

The results in the simulated environment confirm our analytical findings, where our solution constructs with a probability of ρ either a probabilistic honest set containing at least one honest node for transaction history synchronization, a probabilistic honest set containing at least a majority of honest nodes for transaction presence checking, or signals the client to stop further

operation if the above sets could not be constructed. If the client running our solution is not resource constrained (i.e., bandwidth and time are not of primary concern), the solution is able to construct a probabilistic honest set if no more than 50% of the network nodes are malicious, while the resource constrained variant of the solution is able to construct a probabilistic honest set up to the point until about 25% of the network nodes are malicious. Under realistic circumstances, the probabilistic variant of our solution can be by two orders of magnitude more efficient in communicating with members of Π_h than its deterministic variant. The results gathered from the execution of an Aurora node with the ETH production network show that the integration of our solution into existing client requires approximately 0.31 MB of RAM and 1 MB of storage at runtime. As such, the solution can be executed on devices with as little RAM as the Raspberry Pi 2 Model B (1 GB RAM), but our experiments also demonstrated that remote nodes in the ETH network in its current deployment do not sufficiently support all the required messages for our solution to be used for transaction presence checking.

In summary, in this thesis, we have proposed and developed a solution that enhances the resilience and decentralization properties of public and permissionless DLT solutions by allowing new nodes entering a DL network containing malicious nodes to synchronize the ledger history even in the absence of known and trusted remote nodes on which new nodes are currently relatively dependent. Moreover, our solution enables efficient state checking of a transaction without downloading the entire ledger or even its part, without affecting relevant DLT properties. These contributions lower the barrier to entry for consumer hardware and incentivize users to run their own clients instead of implicitly relying on remote nodes to interact with the ledger, meaning that the proposed solution is consistent with the decentralized and trustless ethos of DLT.

6.2 Further research and discussion

The exploration and development of our Proof of Concept solution has raised new research questions that are open for discussion and possible future work:

Π_h size in relation to ρ: The construction of a Π_h has the following properties. First, a Π_h can be constructed in repeated trials. The construction of a Π_h has two possible outcomes — either it contains h honest members or it does not. The probability that a given Π_h contains h honest members in each trial is ρ, and the construction of two different Π_h is independent. This means that the use of Π_h can be modeled as a binomial experiment [86]. In practice, this means that the correctness probability ρ could be reduced and a Π_h could be constructed multiple times to increase the probability that at least one Π_h contains at least h honest members. If we lower the correctness probability ρ to 0.5 and revisit the example of Electrum maintaining connections to

10 remote nodes (see Section 4.1), the following two assertions can be made if a client were to be enhanced with our solution: first, if the client has discovered 6356 nodes and is connected to 10 remote nodes, the client can construct a Π_s and can be 50% certain that 1 node out of 10 is honest even if 5930 ($\approx 93\%$) of discovered nodes are malicious. Second, if the probability that the constructed Π_s contains a single honest node is 0.5 (the probability of success on a single trial), if 10 such Π_s are constructed (the number of trials), then the probability that one or more such sets (the number of successes) will contain at least one honest node is approximately 0.999 (the upper cumulative probability). Simply put, this allows us to reduce ρ and in turn increase κ, at the cost of constructing Π_s multiple times.

Unavailability of public and reachable nodes: DLT solutions such as BTC allow ping and pong messages to be exchanged only with public remote nodes with free TCP slots. We can address the problem by filtering out remote nodes without free slots, by changing the network protocol so that remote nodes respond to ping messages regardless of the number of free slots, or as a permanent solution, it is likely that as DLT becomes more widespread and mature, more publicly reachable nodes will emerge, so the problem will be alleviated.

Stochastic consensus with respect to our solution: The consequences of changing the state of the ledger during the execution of the Aurora node, as well as the influence of the stochastic nature of some consensus mechanisms, have not yet been studied. However, one could compensate for this by not only querying the last state, but also searching for a relatively small amount of data prior to the current state (i.e., prior to the referent block), to find a common ancestor state to verify that a soft fork is not in progress.

Lack of light clients and incentives: The lack of light client capabilities and incentives for full nodes to provide services to light client in the ETH mainnet is an open problem. Full nodes lack incentives to serve light clients. Proposals for generic super-light clients already exist [9]. Incentive mechanisms are outside the scope of this work, but our solution can be modified to support an incentive mechanism. For example, micropayments can be used to reward honest members of a given Π_h when a transaction is verified or a THS is successfully completed.

Reduction of the duration of a gathering: The solution could benefit from three improvements that shorten the duration of a gathering. First, a device running the Aurora node could perform the gathering while idle or charging, and cache the results to shorten the total time a client must wait before receiving a final response. In particular, this means that the total number of messages exchanged before receiving a final response, expressed in Eq. (4.15), the first term of the equation can be omitted (the complexity of a gathering). Second, the Probabilistic honest set construction algorithm is suitable for parallel execution and would benefit greatly from it. If

q represents the number of processor cores on a given machine, then the minimum time required to execute a gathering is divided by a factor of q. Third, if a user knows a trusted person (e.g., a family member), the constructed probabilistic honest sets can be shared and reused rather than constructed from scratch. In particular, this means that it is not necessary to perform gathering and the total number of messages exchanged before receiving a final response, expressed in Eq. (4.15), the first term of the equation can be omitted (the complexity of a gathering).

Adversary with unlimited computational resources: With Assumption 5, we assume that low-resource Eclipse attacks on DLT solutions are infeasible and therefore the attacker can spawn up to κ sybils. It would be interesting to compare some of the existing solutions for detecting sybils (besides the already mentioned [16]) and compare them with our solution to see if such solutions can be modified to solve Problem 1 and Problem 2 and to analyze the relevant trade-offs.

Bibliography

[1]Ben čić, F. M., Skočir, P., Podnar Žarko, I., "Dl-tags: Dlt and smart tags for decentralized, privacy-preserving, and verifiable supply chain management", IEEE access, Vol. 7, 2019, str. 46 198–46 209.

[2]Hileman, G., Rauchs, M., "2017 global blockchain benchmarking study", Available at SSRN 3040224, 2017.

[3]Ben čić, F. M., Podnar Žarko, I., "Aurora: a probabilistic algorithm for distributed ledgers enabling trustless synchronization and transaction inclusion verification", CoRR, Vol. abs/2108.08272, 2021, dostupno na: https://arxiv.org/abs/2108.08272

[4]Ben čić, F. M., Podnar Žarko, I., "Aurora-trinity: A super-light client for distributed ledger networks extending the ethereum trinity client", dostupno na: https://www.mdpi.com/1424-8220/22/5/1835 2022.

[5]Homoliak, I., Venugopalan, S., Hum, Q., Szalachowski, P., "A security reference architecture for blockchains", in 2019 IEEE International Conference on Blockchain (Blockchain). IEEE, 2019, str. 390–397.

[6]Nakamoto, S., "Bitcoin: A peer-to-peer electronic cash system", Manubot, Tech. Rep., 2019.

[7]Wood, G. *et al.*, "Ethereum: A secure decentralised generalised transaction ledger", Ethereum project yellow paper, Vol. 151, No. 2014, 2014, str. 1–32.

[8]Zamyatin, A., Avarikioti, Z., Perez, D., Knottenbelt, W. J., "Txchain: Efficient cryptocurrency light clients via contingent transaction aggregation", in Data Privacy Management, Cryptocurrencies and Blockchain Technology. Springer, 2020, str. 269–286.

[9]Lu, Y., Tang, Q., Wang, G., "Generic superlight client for permissionless blockchains", in European Symposium on Research in Computer Security. Springer, 2020, str. 713–733.

[10]Heilman, E., Kendler, A., Zohar, A., Goldberg, S., “Eclipse attacks on bitcoin’s peer-to-peer network”, in 24th {USENIX} Security Symposium ({USENIX} Security 15), 2015, str. 129–144.

[11]Saad, M., Spaulding, J., Njilla, L., Kamhoua, C., Shetty, S., Nyang, D., Mohaisen, D., “Exploring the attack surface of blockchain: A comprehensive survey”, IEEE Communications Surveys & Tutorials, Vol. 22, No. 3, 2020, str. 1977–2008.

[12]Apostolaki, M., Zohar, A., Vanbever, L., “Hijacking bitcoin: Routing attacks on cryptocurrencies”, in 2017 IEEE Symposium on Security and Privacy (SP). IEEE, 2017, str. 375–392.

[13]Neudecker, T., Characterization of the bitcoin peer-to-peer network (2015-2018). KIT Karlsruher Institut für Technologie, Fakultät für Informatik, 2019.

[14]Nakamoto, S., “Bitcoin: A peer-to-peer electronic cash system”, Decentralized Business Review, 2008, str. 21260.

[15]Ben čić, F. M., Podnar Žarko, I., “Distributed ledger technology: Blockchain compared to directed acyclic graph”, in 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2018, str. 1569–1570.

[16]Alangot, B., Reijsbergen, D., Venugopalan, S., Szalachowski, P., “Decentralized lightweight detection of eclipse attacks on bitcoin clients”, in 2020 IEEE International Conference on Blockchain (Blockchain). IEEE, 2020, str. 337–342.

[17]Zhu, Q., Loke, S. W., Trujillo-Rasua, R., Jiang, F., Xiang, Y., “Applications of distributed ledger technologies to the internet of things: A survey”, ACM computing surveys (CSUR), Vol. 52, No. 6, 2019, str. 1–34.

[18]Antonopoulos, A. M., Wood, G., Mastering ethereum: building smart contracts and dapps. O’reilly Media, 2018.

[19]Antonopoulos, A. M., Mastering Bitcoin: unlocking digital cryptocurrencies. " O’Reilly Media, Inc.", 2014.

[20]Popov, S., “The tangle”, White paper, Vol. 1, No. 3, 2018.

[21]LeMahieu, C., “Nano: A feeless distributed cryptocurrency network”, Nano [Online resource]. URL: https://nano. org/en/whitepaper (date of access: 24.03. 2018), Vol. 16, 2018, str. 17.

[22]Sompolinsky, Y., Zohar, A., “Phantom”, IACR Cryptology ePrint Archive, Report 2018/104, 2018.

[23]Lewenberg, Y., Sompolinsky, Y., Zohar, A., "Inclusive block chain protocols", in International Conference on Financial Cryptography and Data Security. Springer, 2015, str. 528–547.

[24]Narayanan, A., Clark, J., "Bitcoin's academic pedigree", Communications of the ACM, Vol. 60, No. 12, 2017, str. 36–45.

[25]Ben čić, F. M., Hrga, A., Podnar Žarko, I., "Aurora: a robust and trustless verification and synchronization algorithm for distributed ledgers", in 2019 IEEE International Conference on Blockchain (Blockchain). IEEE, 2019, str. 332–338.

[26]Mizrahi, A., Koren, N., Rottenstreich, O., "Optimizing merkle proof size for blockchain transactions", in 2021 International Conference on COMmunication Systems & NETworkS (COMSNETS). IEEE, 2021, str. 299–307.

[27]Singh, A. *et al.*, "Eclipse attacks on overlay networks: Threats and defenses", in In IEEE INFOCOM. Citeseer, 2006.

[28]Castro, M., Druschel, P., Ganesh, A., Rowstron, A., Wallach, D. S., "Secure routing for structured peer-to-peer overlay networks", ACM SIGOPS Operating Systems Review, Vol. 36, No. SI, 2002, str. 299–314.

[29]Cholez, T., Chrisment, I., Festor, O., Doyen, G., "Detection and mitigation of localized attacks in a widely deployed p2p network", Peer-to-Peer Networking and Applications, Vol. 6, No. 2, 2013, str. 155–174.

[30]Delgado-Segura, S., Pérez-Solà, C., Herrera-Joancomartí, J., Navarro-Arribas, G., Borrell, J., "Cryptocurrency networks: A new p2p paradigm", Mobile Information Systems, Vol. 2018, 2018.

[31]Wüst, K., Gervais, A., "Ethereum eclipse attacks", ETH Zurich, Tech. Rep., 2016.

[32]Marcus, Y., Heilman, E., Goldberg, S., "Low-resource eclipse attacks on ethereum's peer-to-peer network.", IACR Cryptol. ePrint Arch., Vol. 2018, 2018, str. 236.

[33]Xu, G., Guo, B., Su, C., Zheng, X., Liang, K., Wong, D. S., Wang, H., "Am i eclipsed? a smart detector of eclipse attacks for ethereum", Computers & Security, Vol. 88, 2020, str. 101604.

[34]Henningsen, S., Teunis, D., Florian, M., Scheuermann, B., "Eclipsing ethereum peers with false friends", arXiv preprint arXiv:1908.10141, 2019.

[35]Touceda, D. S., Sierra, J. M., Izquierdo, A., Schulzrinne, H., "Survey of attacks and defenses on p2psip communications", IEEE Communications Surveys & Tutorials, Vol. 14, No. 3, 2011, str. 750–783.

[36]Motlagh, S. G., Miši ć, J., Mišić, V. B., "Impact of node churn in the bitcoin network", IEEE Transactions on Network Science and Engineering, Vol. 7, No. 3, 2020, str. 2104–2113.

[37]O'Connor, R., "Simplicity: A new language for blockchains", in Proceedings of the 2017 Workshop on Programming Languages and Analysis for Security, 2017, str. 107–120.

[38]Al Khalil, F., Butler, T., O'Brien, L., Ceci, M., "Trust in smart contracts is a process, as well", in International Conference on Financial Cryptography and Data Security. Springer, 2017, str. 510–519.

[39]Pinto, G. V., Dias, J. P., Ferreira, H. S., "Blockchain-based pki for crowdsourced iot sensor information", in International conference on soft computing and pattern recognition. Springer, 2018, str. 248–257.

[40]Xia, P., Wang, H., Yu, Z., Liu, X., Luo, X., Xu, G., "Ethereum name service: the good, the bad, and the ugly", arXiv preprint arXiv:2104.05185, 2021.

[41]Kalodner, H. A., Carlsten, M., Ellenbogen, P., Bonneau, J., Narayanan, A., "An empirical study of namecoin and lessons for decentralized namespace design.", in WEIS. Citeseer, 2015.

[42]Karlsson, K., Jiang, W., Wicker, S., Adams, D., Ma, E., van Renesse, R., Weatherspoon, H., "Vegvisir: A partition-tolerant blockchain for the internet-of-things", in 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2018, str. 1150–1158.

[43]Wang, Q., Li, R., Wang, Q., Chen, S., "Non-fungible token (nft): Overview, evaluation, opportunities and challenges", arXiv preprint arXiv:2105.07447, 2021.

[44]Hardjono, T., Smith, N., Pentland, A. S., "Anonymous identities for permissioned blockchains", 2014.

[45]Andoni, M., Robu, V., Flynn, D., Abram, S., Geach, D., Jenkins, D., McCallum, P., Peacock, A., "Blockchain technology in the energy sector: A systematic review of challenges and opportunities", Renewable and Sustainable Energy Reviews, Vol. 100, 2019, str. 143–174.

[46]Tian, F., “A supply chain traceability system for food safety based on haccp, blockchain & internet of things”, in 2017 International conference on service systems and service management. IEEE, 2017, str. 1–6.

[47]Orjuela, K. G., Gaona-García, P. A., Marin, C. E. M., “Towards an agriculture solution for product supply chain using blockchain: case study agro-chain with bigchaindb”, Acta Agriculturae Scandinavica, Section B—Soil & Plant Science, Vol. 71, No. 1, 2021, str. 1–16.

[48]Rakic, B., Levak, T., Drev, Z., Savic, S., Veljkovic, A., “First purpose built protocol for supply chains based on blockchain”, OriginTrail, Ljubljana, Slovenia, Tech. Rep, Vol. 1, 2017.

[49]Bocek, T., Rodrigues, B. B., Strasser, T., Stiller, B., “Blockchains everywhere-a use-case of blockchains in the pharma supply-chain”, in 2017 IFIP/IEEE symposium on integrated network and service management (IM). IEEE, 2017, str. 772–777.

[50]Hrga, A., Ben čić, F. M., Podnar Žarko, I., “Technical analysis of an initial coin offering”, in 2019 15th International Conference on Telecommunications (ConTEL). IEEE, 2019, str. 1–8.

[51]Lee, W.-M., “Beginning ethereum smart contracts programming”, With Examples in Python, Solidity and JavaScript, 2019.

[52]Kwon, J., “Tendermint: Consensus without mining”, Draft v. 0.6, fall, Vol. 1, No. 11, 2014.

[53]Xu, L., Chen, L., Gao, Z., Xu, S., Shi, W., “Efficient public blockchain client for lightweight users”, arXiv preprint arXiv:1811.04900, 2018.

[54]Leung, D., Suhl, A., Gilad, Y., Zeldovich, N., “Vault: Fast bootstrapping for the algorand cryptocurrency.”, in NDSS, 2019.

[55]Chen, J., Micali, S., “Algorand”, arXiv preprint arXiv:1607.01341, 2016.

[56]Bünz, B., Kiffer, L., Luu, L., Zamani, M., “Flyclient: Super-light clients for cryptocurrencies”, in 2020 IEEE Symposium on Security and Privacy (SP). IEEE, 2020, str. 928–946.

[57]Letz, D., “Blockquick: Super-light client protocol for blockchain validation on constrained devices.”, IACR Cryptol. ePrint Arch., Vol. 2019, 2019, str. 579.

[58]Yu, H., Kaminsky, M., Gibbons, P. B., Flaxman, A., “Sybilguard: defending against sybil attacks via social networks”, in Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications, 2006, str. 267–278.

[59]Ding, Q., Katenka, N., Barford, P., Kolaczyk, E., Crovella, M., “Intrusion as (anti) social communication: characterization and detection”, in Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, 2012, str. 886–894.

[60]Chen, H.-H., Giles, C. L., “Ascos: an asymmetric network structure context similarity measure”, in 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013). IEEE, 2013, str. 442–449.

[61]Chowdhury, S., Khanzadeh, M., Akula, R., Zhang, F., Zhang, S., Medal, H., Marufuzzaman, M., Bian, L., “Botnet detection using graph-based feature clustering”, Journal of Big Data, Vol. 4, No. 1, 2017, str. 1–23.

[62]Zhang, J., Xiang, Y., Wang, Y., Zhou, W., Xiang, Y., Guan, Y., “Network traffic classification using correlation information”, IEEE Transactions on Parallel and Distributed systems, Vol. 24, No. 1, 2012, str. 104–117.

[63]Zhang, J., Chen, C., Xiang, Y., Zhou, W., Vasilakos, A. V., “An effective network traffic classification method with unknown flow detection”, IEEE Transactions on Network and Service Management, Vol. 10, No. 2, 2013, str. 133–147.

[64]Hildrum, K., Kubiatowicz, J., “Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks”, in International Symposium on Distributed Computing. Springer, 2003, str. 321–336.

[65]Awerbuch, B., Scheideler, C., “Robust random number generation for peer-to-peer systems”, in International Conference On Principles Of Distributed Systems. Springer, 2006, str. 275–289.

[66]Jesi, G. P., Montresor, A., van Steen, M., “Secure peer sampling”, Computer Networks, Vol. 54, No. 12, 2010, str. 2086–2098.

[67]Bakker, A., Van Steen, M., “Puppetcast: A secure peer sampling protocol”, in 2008 European Conference on Computer Network Defense. IEEE, 2008, str. 3–10.

[68]Bortnikov, E., Gurevich, M., Keidar, I., Kliot, G., Shraer, A., “Brahms: Byzantine resilient random membership sampling”, Computer Networks, Vol. 53, No. 13, 2009, str. 2340–2359.

[69]Al-Bassam, M., Sonnino, A., Buterin, V., Khoffi, I., “Fraud and data availability proofs: Detecting invalid blocks in light clients”, in International Conference on Financial Cryptography and Data Security. Springer, 2021, str. 279–298.

[70]Rivadulla, A., “Mathematical statistics and metastatistical analysis”, Erkenntnis, Vol. 34, No. 2, 1991, str. 211–236.

[71]Biryukov, A., Khovratovich, D., Pustogarov, I., “Deanonymisation of clients in bitcoin p2p network”, in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014, str. 15–29.

[72]Deshpande, V., Badis, H., George, L., “Btcmap: Mapping bitcoin peer-to-peer network topology”, in 2018 IFIP/IEEE International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN). IEEE, 2018, str. 1–6.

[73]Miller, A., Litton, J., Pachulski, A., Gupta, N., Levin, D., Spring, N., Bhattacharjee, B., “Discovering bitcoin’s public topology and influential nodes”, et al, 2015.

[74]Paavolainen, S., Carr, C., “Security properties of light clients on the ethereum blockchain”, IEEE Access, Vol. 8, 2020, str. 124 339–124 358.

[75]Motlagh, S. G., Miši ć, J., Mišić, V. B., “An analytical model for churn process in bitcoin network with ordinary and relay nodes”, Peer-to-Peer Networking and Applications, Vol. 13, No. 6, 2020, str. 1931–1942.

[76]Ben čić, F. M., Podnar Žarko, I., “Aurora-trinity: A super-light client for distributed ledger networks extending the ethereum trinity client”, Sensors, Vol. 22, No. 5, 2022, str. 1835.

[77]Miši ć, J., Mišić, V. B., Chang, X., Motlagh, S. G., Ali, M. Z., “Modeling of bitcoin’s blockchain delivery network”, IEEE Transactions on Network Science and Engineering, Vol. 7, No. 3, 2019, str. 1368–1381.

[78]Hancock, P. J., “An empirical comparison of selection methods in evolutionary algorithms”, in AISB workshop on evolutionary computing. Springer, 1994, str. 80–94.

[79]Aoki, Y., Otsuki, K., Kaneko, T., Banno, R., Shudo, K., “Simblock: A blockchain network simulator”, in IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE, 2019, str. 325–329.

[80]Dagon, D., Gu, G., Lee, C. P., Lee, W., “A taxonomy of botnet structures”, in Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007). IEEE, 2007, str. 325–339.

[81]Yang, H. Y., Tempero, E., Melton, H., “An empirical study into use of dependency injection in java”, in 19th Australian Conference on Software Engineering (aswec 2008). IEEE, 2008, str. 239–247.

[82]Naumovich, G., “Using the observer design pattern for implementation of data flow analyses”, ACM SIGSOFT Software Engineering Notes, Vol. 28, No. 1, 2002, str. 61–68.

[83]Maymounkov, P., Mazieres, D., “Kademlia: A peer-to-peer information system based on the xor metric”, in International Workshop on Peer-to-Peer Systems. Springer, 2002, str. 53–65.

[84]Kim, S. K., Ma, Z., Murali, S., Mason, J., Miller, A., Bailey, M., “Measuring ethereum network peers”, in Proceedings of the Internet Measurement Conference 2018, 2018, str. 91–104.

[85]Bormann, C., Ersue, M., Keranen, A., “Terminology for constrained-node networks”, Internet Engineering Task Force (IETF): Fremont, CA, USA, 2014, str. 2070–1721.

[86]Saperstein, B., “On the occurrence of n successes within n bernoulli trials”, Technometrics, Vol. 15, No. 4, 1973, str. 809–818.

Nomenclature

C A constant.

H L'Hôpital's Rule.

K The number of success in the population in the context of a hypergeometric experiment.

M Malicious nodes set

N A population size in the context of a hypergeometric experiment.

S A set of all network nodes.

T A network topology mapping.

X A random variable.

Y Random variable denoting the number of unique nodes found throughout a single gathering

Z Random variable denoting the average number of unique nodes found throughout a single gathering, $Z \sim \mathcal{N}(\mu, \sigma^2)$.

Δ_p A *deterministic honest set* guaranteed to contain at least a majority of honest nodes.

Δ_s A deterministic honest set guaranteed to contain at least one honest node.

Δ A shorthand for Δ_h when h is implied from context or not relevant.

Γ A set containing nodes found during the process network of discovery (i.e., gathering).

Π_p A probabilistic honest set guaranteed to contain a majority of honest node with at least probability ρ.

Π_s A probabilistic honest set guaranteed to contain at least one honest node with at least probability ρ.

Π A shorthand for Π_h when h is implied from context or not relevant.

$\aleph$ The contents of a pong message.

Δ_h Deterministic honest set

Π_h Probabilistic honest set

Γ_i Subset of network nodes encountered up to step i

$\hat{U}$ Vector of weights containing all weights of nodes unfamiliar to node t_l.

$\hat{\kappa}$ An approximation of κ

$\hat{c}$ Node capacities vector.

$\hat{w}$ Node weights vector.

κ Desired malicious node tolerance, $\kappa \in \mathbb{N}$.

λ Exponential distribution parameter.

ω Measured ratio between $|\Gamma|$ and κ when the corresponding probabilistic honest set size starts to behave as a sub-linear function of κ.

ρ Probability guarantee for Π_h.

a A new node entering a DL network not enabled with the Aurora module.

blk_{id} A block identifier.

c Node capacities vector element.

d Number of draws made in a gathering

fc First contact node identifier.

hc Halting condition encapsulation.

h Number of headers

k The number of success in the sample in the context of a hypergeometric experiment.

l An Ethereum node maintains l-number of nodes in each row of the Kademlia-like structure.

$m(\kappa, Z)_A$ Messages function

$majority$ The majority number of nodes in a Π_p used in Algorithm 5.

$notPresent$ The number of nodes in a Π_p that asserted that a transaction is not present in the ledger, used in Algorithm 5.

$nxtDraw$ Identifier of a next draw in a gathering.

n The sample size in the context of a hypergeometric experiment.

$present$ The number of nodes in a Π_p that asserted that a transaction is present in the ledger, used in Algorithm 5.

q A number of processor cores on a given machine.

$r(\kappa, Z)_A$ Rounds function

r The number of nodes to which a *lookup* request is sent in each iteration of a *recursive lookup*.

$s(\kappa)_A$ Storage function

$sync$ A boolean indicating whether a THS has been successfully executed in Algorithm 4.

t_{max} Maximum execution time in seconds

tx_{id} A transaction identifier.

t A network topology mapping element.

w Node weights vector element.

x Random variable occurrence.

z $0 \leq z \leq 1; z \in \mathbb{R}$, represents an estimate of the suspicious behavior of the supposedly present malicious clique.

$|M|$ Number of malicious nodes in the network

$|\Gamma|$ The number of unique nodes discovered during a gathering, also the population size used for Π_h construction.

addNode A procedure in a BTC client implementation that connects to a remote peer manually.

ADDR A pong message in the BTC network.

Aleth An open source ETH network client written in *C++*.

Aurora A set of three specific algorithms: the Probabilistic honest set construction algorithm (Algorithm 2), the Transaction history synchronization algorithm (Algorithm 4) and the Transaction presence checking algorithm (Algorithm 5) that offer a solution to Problem 1 and Problem 2.

Aurora module A runtime able to execute Algorithm 2, Algorithm 4 and Algorithm 5.

Aurora node A DLT node enhanced with the Aurora module, which is able to execute able to execute Algorithm 2, Algorithm 4 and Algorithm 5 as a consequence.

Aurora node observers Injected dependencies and part of an observer design pattern used to collect metadata from a simulation instance.

block Units in a blockchain. Blocks contain headers and transactions. Each block header contains, among other metadata, a reference to its predecessor in the form of the predecessor's hash. The initial state is hard-coded in the first block, the genesis block. Unlike other blocks, the genesis block has no predecessor.

block header A header of a block containing various metadata (e.g., a reference to its predecessor).

block lattice A free variable in a PoW puzzle.

blockchain A DL data structure consisting of ordered units called blocks.

bootnode A set of known nodes that are usually hardcoded in the client. These nodes are contacted when a new node enters the network, and used for the process of discovering the network.

bootstrapping The process of gaining network access in P2P and DLT environments.

canonical chain See longest chain.

chain fork A sequence of blocks in a blockchain that diverges from the canonical chain.

chain head The latest block in a blockchain.

Component API A component of the Trinity client which allows the creation of components for modular extension of existing system functionality.

consumer An entity in the TIS system that consumes a product.

Current Aurora version The latest version of out solution, the same as Aurora.

Depreciated Aurora version An older and depreciated version of Aurora implemented in [4, 25].

desired malicious node tolerance The maximum number of malicious nodes that can exist (i.e. be tolerated) in a given DL network such that an Δ_h or Π_h can be constructed. The term is used synonymously with κ

deterministic progress set Given that there are $|\Gamma|$ nodes, out of which κ are malicious, the set contains a majority of honest nodes. This set has a deterministic size of $2\kappa+1$.

deterministic safe set Given that there are $|\Gamma|$ nodes, out of which κ are malicious, the set contains at least one honest node. This set has a deterministic size of $\kappa + 1$.

devp2p A set of network protocols that make up the ETH P2P network.

difficulty target A measure of the difficulty of a PoW puzzle.

DL-T proxy Middleware in DL-T designed to make the solution DL agnostic.

draw A step performed during the gathering.

E-commerce store A stakeholder in the TIS system that distributes a product to the end consumer.

Ether The native token of the ETH network.

Fake bootstrapping The process of bootstrapping with a malicious peer.

fast sync An algorithm in ETH whose goal is to exchange processing power for bandwidth usage during THS.

full node A node that stores and validates the ledger in its entirety, and is typically capable of serving light clients.

gathering The process of discovering nodes in an DL network by exploring the network in a way that draws an DAG on an abstract network topology where vertices are nodes and edges between vertices have a direction assigned to them.

genesis block An initial block in a blockchain, referencing no predecessor and containing the initial ledger state.

genesis transaction In the context of a DAG based DL, the first transaction with no predecessor, similar to the genesis block in a blockchain.

GETADDR A ping message in the BTC network.

geth An implementation of the ETH protocol in the go programming language.

GHOSTDAG A heuristic variant of Phantom.

go A compiled and statically typed programming language.

halting condition The condition at which a gathering should halt in the form of a predicate that can be defined arbitrarily for a network and/or a use case.

header chain In the context of blockchain, a chain containing block headers instead of entire blocks.

hypergeometric experiment The random selection, without replacement, of a subset of elements from a finite population, where each element in the population can be classified as a success or failure.

Initial Handshake The process of creating session keys at the establishment of a RLPx connection.

IOTA A DAG-oriented DL.

leaf Data item in a Merkle tree with no children.

light client A solution that, compared to a full node, reduces the amount of data that must

be stored on a node in order for that node to operate within the network, and is largely dependent on full nodes to provide the metadata necessary for operation.

longest chain The sequence of blocks in a blockchain supported by the majority of the voting power, also called the canonical chain, which contains the most blocks.

mainnet The main public ETH ledger.

Merkle proof A proof of the existence of certain data corresponding to a leaf in a Merkle tree consisting of the leaf's sibling path containing the identifiers for the siblings of the nodes in a path from the leaf to the Merkle root [26].

Merkle root Top (root) element of a Merkle tree.

Merkle tree A binary tree of hash pointers [24], also a well-known tool in cryptography that allows efficient proof of a data item's membership in a set without revealing the entire set [26].

Message Call The act of passing a message from one ETH address to another [7].

miner A node in a PoW driven DL network.

mining The process of generating blocks in a PoW driven DL network.

Nakamoto consensus A method for achieving consensus in an environment where the number of participants is unknown, participants can enter and leave the network at will, and participants may or may not be malicious and do not trust each other [24].

Nano A DAG-oriented DL.

node A participant in a DL network, also an element of a Merkle tree.

nonce A free variable in a PoW puzzle.

permissioned In the context of DLT, a property meaning that any public entity can write to the the ledger, as opposed to the permissionless property.

permissionless In the context of DLT, a property meaning that any public entity can write to the the ledger, as opposed to the permissioned property.

Phantom A hybrid DL solution using both a DAG and a blockchain as data structures.

ping Request from the Aurora node to the remote node for the remote's peer list.

pong A remote node's response to a ping message containing a subset of all peers known to the remote.

private In the context of DLT, a property meaning that selected entities can read from the ledger, as opposed to the public property [2].

probabilistic honest set For a given Γ, a set Π_h is a subset of Γ which contains at least h honest nodes with probability ρ. The probability ρ is derived from an underlying hypergeometric distribution.

Probabilistic honest set construction algorithm Non-deterministic algorithm for the generation of probabilistic honest sets (see Algorithm 2).

probabilistic progress set Given that there are $|\Gamma|$ nodes, out of which κ are malicious, the set

contains at least one honest node with at least probability ρ.

probabilistic safe set Given that there are $|\Gamma|$ nodes, out of which κ are malicious, the set contains at least one honest node with at least probability ρ.

producer A stakeholder in the TIS system that creates a product.

proof request Request from the Aurora node to the remote node for a specific Merkle proof.

proof response A remote node's response to a proof request message containing a specific Merkle proof.

public In the context of DLT, a property meaning that any public entity can read from the ledger, as opposed to the private property [2].

Python An object-oriented, interpreted, and high-level programming language.

Script Stack-based language for BTC.

Smart Contract Code written and executed for a distributed environment that enables support for custom data storage and arbitrary business logic based on user requirements.

Smart Tag Markers used to track digital products in the TIS ecosystem.

soft fork A natural phenomenon in a blockchain network where two blocks are generated at roughly the same time and claim the same predecessor.

Solidity A domain-specific high-level Turing-complete language for ETH Smart Contracts.

state The data contained in a ledger at a given point in time, stored directly in the ledger or derived from the initial state to which a sequence of transactions has been applied.

status The latest ledger as defined by the consensus protocol (the highest total difficulty, the longest chain, etc.).

status request Request from the Aurora node to the remote node for the remote's ledger status.

status response A remote node's response to a status request message containing the remote's ledger status.

strongest chain The sequence of blocks in a blockchain supported by the majority of the voting power, also called the canonical chain, which contains the most accumulated work.

Tangle In the context of a DAG based DL, a structure used for the IOTA [20] DL.

timestamp In the context of BTC, the approximate time in seconds elapsed from the Unix Epoch to the creation of this block.

transaction Input for a DL that may cause a change to the ledger state.

Transaction history synchronization algorithm In distributed ledger networks with malicious nodes which are actively trying to subvert a new node interacting with the network, the algorithm communicates with a set of remote network nodes created by Algorithm 2, where the set contains at least one honest node with a predefined probability. The algorithm ensures that the transaction history can eventually be synchronized from an honest node (see Algorithm 4).

Transaction presence checking algorithm In distributed ledger networks with malicious nodes

which are actively trying to subvert a new node interacting with the network, the algorithm communicates with a set of remote network nodes created by Algorithm 2, where the set contains a majority of honest nodes with a predefined probability. The algorithm ensures that the presence of a transaction within a DL can be inferred by a majority vote (see Algorithm 5).

Trinity An open source ETH network client written in Python.

trustless An action is performed in a trustless manner if it respects the feature of trustlessness.

trustlessness In the context of DLT, a property meaning the elimination of the need for a centralized trusted third party responsible for resolving conflicts and maintaining a global truth that is trusted by all parties who do not trust each other.

Acronyms

API Application Programming Interface
BTC Bitcoin
CA Certificate Authority
DAG Directed Acyclic Graph
DApp Distributed Application
DARPA Defense Advanced Research Projects Agency
DDoS Distributed Denial of Service
DHT Distributed Hash Table
DL Distributed Ledger
DL-T DL-Tags
DLT Distributed Ledger Technology
DNS Domain Name System
DoS Denial of Service
ENR Ethereum Node Records
ETH Ethereum
ETH Ethereum Wire Protocol
EVM Ethereum Virtual Machine
ICO Initial Coin Offering
IM Instant Messaging
IoT Internet of Things
IP Internet Protocol
LES Light Ethereum Subprotocol
NDP Node Discovery Protocol
NFT Non-Fungible Token
ORV Open Representative Voting
P2P Peer-to-Peer
PKI Public Key Infrastructure
PoC Proof of Concept
PoS Proof of Stake

PoW Proof of Work
QR Quick Response
RAM Random Access Memory
RFID Radio Frequency Identification
RLPx RLPx Transport Protocol
SOTA State of the Art
TCP Transmission Control Protocol
THS Transaction History Synchronization
TIS TagItSmart
TPC Transaction Presence Checking
UDP User Datagram Protocol
XNO Nano

Index

List of Figures

List of Tables

Biography

Federico Matteo Benčić received the M.Sc. degree in information and communication technology from the University of Zagreb, Faculty of Electrical Engineering and Computing in 2017, where he is currently pursuing the Ph.D. degree with the Department of Telecommunications. He is currently an Assistant with the Department of Telecommunications, Faculty of Electrical Engineering and Computing, University of Zagreb.

Within the DL-Tags project, as a Researcher at University of Zagreb, he designed and developed a DLT-agnostic solution for supply chain management enhanced by the Internet of Things (IoT) that relies on special tags (i.e., QR-codes) to verify the authenticity of a product across the supply chain. The solution was implemented on top of Ethereum, deployed and tested in Docker containers, and integrated in an existing, open source, PHP-based e-commerce platform Magento.

Within the IoT4us project, as a Researcher at University of Zagreb, visited of the University of Brno to acquire new knowledge and establish cooperation with the University of Brno within the framework of activities related to the scientific project IoT4us.

He has published five peer-reviewed papers in international journals and conferences.

Journal publications

1.Ben čić, F. M., Podnar Žarko, I., "Aurora-trinity: A super-light client for distributed ledger networks extending the ethereum trinity client", Sensors, vol. 22, no. 5, p. 1835, Feb. 2022, doi: 10.3390/s22051835.

2.Ben čić, F. M., Skočir, P., Podnar Žarko, I., "Dl-tags: Dlt and smart tags for decentralized, privacy-preserving, and verifiable supply chain management" IEEE Access, vol. 7, pp. 46198-46209, 2019, doi: 10.1109/ACCESS.2019.2909170.

Conference publications

1.Ben čić, F. M., Podnar Žarko, I., "Distributed ledger technology: Blockchain compared to directed acyclic graph", in 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2018, pp. 1569–1570.

2.Ben čić, F. M., Hrga, A., Podnar Žarko, I., “Aurora: a robust and trustless verification and synchronization algorithm for distributed ledgers”, in 2019 IEEE International Conference on Blockchain (Blockchain). IEEE, 2019, pp. 332–338.

3.Hrga, A., Ben čić, F.-M., Podnar Žarko, I., “Technical analysis of an initial coin offering”, in 2019 15th International Conference on Telecommunications (ConTEL). IEEE, 2019, pp. 1–8.

Životopis

Federico Matteo Benčić diplomirao je na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu 2017. godine, gdje trenutno pohađa doktorski studij. Diplomirao na Zavodu za telekomunikacije. Trenutno je asistent na Zavodu za telekomunikacije Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu.

Unutar projekta DL-Tags, kao istraživač na Sveučilištu u Zagrebu, dizajnirao je i razvio DLT-agnostičko rješenje za upravljanje lancem opskrbe poboljšano Internetom stvari (IoT) koje se oslanja na posebne oznake (tj. QR-kodove) za provjeru autentičnost proizvoda u cijelom opskrbnom lancu. Rješenje je implementirano povrh Ethereuma, te testirano korištenjem Docker kontejnera te integrirano u postojeću platformu za e-trgovinu Magento s otvorenim kodom temeljenu na PHP-u.

U sklopu projekta IoT4us, kao istraživač na Sveučilištu u Zagrebu, posjetio je Sveučilište u Brnu radi stjecanja novih znanja i uspostavljanja suradnje sa Sveučilištem u Brnu u okviru aktivnosti vezanih za znanstveni projekt IoT4us.

Objavio je pet recenziranih radova u međunarodnim časopisima i konferencijama.