Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Diploma Thesis

Global P2P Network for Confidential Sharing of Threat Intelligence and Collaborative Defense

Author: Bc. Martin Řepa
Supervised by García Sebastián, Assist. Prof., PhD

May 2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Řepa** Jméno: **Martin** Osobní číslo: **466119**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Kybernetická bezpečnost**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Globální P2P Systém pro Sdílení Kybernetických Dat a Kolaborativní Ochranu v Kyberprostoru

Název diplomové práce anglicky:

Global P2P Network for Confidential Sharing of Threat Intelligence and Collaborative Defense

Pokyny pro vypracování:

The goal of this thesis is to design and implement a global peer to peer networking system to allow reliable, secure and confidential sharing of distributed threat intelligence data using the libp2p project. Unlike standard P2P networks, the system will allow peers to be members of trusted groups to minimise the risk of being targeted by malicious actors. Messaging protocols shall be designed along with peer discovery and peer routing techniques while utilising peers' reliability which is assumed to be dynamically computed by a blackbox trust model. The work will incorporate theoretical discussion and if possible practical experiments about its mitigation of known P2P network attacks. Finally, the implementation will be integrated into Stratosphere Linux intrusion prevention system (SLIPS) to allow sharing data with other SLIPS instances.

Seznam doporučené literatury:

[1] Xuemin Shen, Heather Yu, John Buford, and Mursalin Akon. 2009. Handbook of Peer-to-Peer Networking (1st. ed.). Springer Publishing Company, Incorporated.
[2] Maymounkov, Petar & Eres, David. (2002). Kademlia: A Peer-to-peer Information System Based on the XOR Metric. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. 2429. 10.1007/3-540-45748-8_5.
[3] Baptiste Pretre: Attacks on Peer-to-Peer Networks, 2005
[4] Libp2p project: https://github.com/libp2p/specs
[5] SLIPS: Stratosphere Linux intrusion prevention system https://github.com/stratosphereips/StratosphereLinuxIPS/

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Sebastián García, Ph.D. centrum umělé inteligence FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **22.02.2022** Termín odevzdání diplomové práce: **20.05.2022**

Platnost zadání diplomové práce: **19.02.2024**

Ing. Sebastián García, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. | Datum převzetí zadání

Podpis studenta

Declaration

I hereby declare that the presented work has been composed solely by myself and that I have listed all sources of information used within it in accordance with the methodical instructions about ethical principles in the preparation of academic theses.

Prague, date *Signature*

Acknowledgements

I would like to thank my supervisor *García Sebastián, Assist. Prof. PhD*, for numerous, highly constructive and inspiring consultations, patience and the significant amount of time he devoted to help me enhance my thesis.

Profound gratitude comes also to all members of The Stratosphere Team for sharing knowledge, creating an inspirational academic background and proofreading my work. Thank you for the opportunity to finish my master studies under your guidance!

Abstract

Despite the severity and amount of daily cyberattacks, the best solutions our community has so far are centralised, threat intelligence shared lists; or centralised, commercially-based defence products. No system exists yet to automatically connect endpoints globally and share information about new attacks to improve their security. This thesis proposes Iris, a global Peer-to-Peer (P2P) system specifically designed for cybersecurity needs. Iris allows collaborative defence in cyberspace with emphasis on security and privacy concerns. It is a pure and completely decentralised P2P network that allows peers to (i) share threat intelligence files, (ii) alert peers about detected attacks, and (iii) ask peers about their opinion on potential attacks. Iris addresses the problem of confidentiality of local threat intelligence data by introducing the concept of *Organisations*. Organisations are cryptographically-verified and trusted groups of peers within the P2P network. They allow Iris to send content only to pre-trusted groups of peers. Iris is optimised for fast, low-bandwidth spread of information by experimenting on different spreading strategies for epidemic protocols. We present a complete implementation of Iris in Go using the LibP2P project together with an integration of Iris into Slips IPS as a collaborative module.

Keywords: Threat Intelligence, P2P Network, Collaborative Defence, Cybersecurity

Abstrakt

Navzdory četnosti a závažnosti kybernetických útoků jsou zatím nejlepším řešením pro sdílení informací o kybernetických hrozbách centralizované systémy sdílených informací nebo centralizované komerční produkty. Neexistuje žádné řešení, které by automaticky spojovalo koncová zařízení a umožnilo rychle a bezpečně sdílet data o kybernetických hrozbách s cílem zlepšení bezpečnosti. Tato práce navrhuje Iris, globální Peer-to-Peer (P2P) systém, který je speciálně navržen pro potřeby sdílení informací o kybernetických hrozbách s důrazem na bezpečnost samotného systému a ochranu soukromí. Iris je plně decentralizovaná P2P síť, která umožňuje připojeným zařízením i) sdílet analýzy hrozeb, ii) varovat ostatní zařízení před právě detekovaným útočníkem, iii) ptát se ostatních zařízení na hodnocení potenciálního útočníka. Pro ochranu soukromí Iris navrhuje nový koncept zvaný Organizace. Organizace reprezentují kryptograficky ověřené skupiny zařízení a umožňují Iris adresovat citlivá data pouze skupinám důvěryhodných zařízení. Práce porovnává různé strategie šíření informací v síti s využitím epidemických protokolů pro optimalizaci šíření zpráv v Iris. Práce zároveň prezentuje kompletní implementaci Iris v Go s využitím knihoven LibP2P a kompletní integraci Iris do systému Slips IPS jako modul pro kolaborativní bezpečnost.

Klíčová slova: Analýzy Hrozeb, Globální P2P Síť, Kolaborativní Kyberbezpečnost

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

ARP Address Resolution Protocol. 10

CIDN Collaborative Intrusion Detection Networks. 2, 23

DDoS Distributed Denial of Service. 20, 60, 76

DHT Distributed Hash Table. 2, 16, 24, 25, 30, 73

DoS Denial of Service. 40

IoC Indicator of Compromise. 2, 24, 28, 34, 68, 73

IoT Internet of Things. 76

IPFS InterPlanetary File System. 2, 21, 33

IPS Intrusion Prevention System. 10

MISP Malware Information Sharing Platform and Threat Sharing. 23

MitM Man-in-the-Middle. 19, 74

NAT Network Address Translation. 13

P2P Peer-to-Peer. 10

Slips Slips. 3, 10, 73, 74

TI Threat Intelligence. 5, 28

TOR The Onion Router. 13, 14

Chapter 1

Introduction

Every device currently connected to the Internet is constantly being attacked [53]. Some organisations capture these attacks to produce threat intelligence (TI) data that is essential for a comprehensive defence against malicious actors on the Internet. In an ideal scenario, we would (i) detect and analyse an attack, (ii) share threat intelligence data, (iii) aggregate or curate the data, and (iv) apply defensive measures across all devices. However, no security and privacy-based system exists that would offer a way to share general threat intelligence and alerts fast, directly between end devices without a central authority.

Currently, the best solutions that our security community has to share information and be better protected are (i) lists of threat intelligence data downloaded for free or paid to companies; or (ii) commercial products that centralise data from clients to improve their defence. In both cases, sharing of threat intelligence is the most used technique.

In order to share threat intelligence, users can use email lists, web downloads, RSS feeds, or text alerts [26]. However, one of the most used open-source systems is the Malware Information Sharing Platform (MISP) [51] . MISP is a software for collecting, storing, visualising and sharing threat intelligence data in the community. It is a semi-centralised service that serves as a middleman between devices, and it is mostly used by humans. Each MISP instance has many users, and instances of different organisations can connect to each other if manually configured.

Malicious incidents in cyberspace repeat. Optimally, the community would spend resources to detect such incidents only once and immediately share the alerts with other devices. Thus, effective sharing of alerts and threat intelligence data could significantly improve security. However, for an effective collaborative defence, the community needs an option to securely and privately share alerts and possibly confidential threat intelligence data in a timely and fully decentralised manner.

Theoretical proposals to improve defence in cyberspace using collaborative knowledge have appeared, mainly in the field of Collaborative Intrusion Detection Networks (CIDNs). Nonetheless, to our knowledge, no CIDN implements a fully decentralised, secure and privacy-based network for fast sharing of alerts and general-purpose threat intelligence data. Moreover, no other P2P system addresses the needs of the security community, where files are not large (as in file sharing P2P systems) but they have other needs: fast distribution of TI, reporting of attackers, consultation of the reputation of Indicators of Compromise (IoC), consideration of trust issues, adversarial peers trying to compromise the network, and pre-trusted groups.

This thesis proposes Iris, a global peer-to-peer system for collaborative defence with special emphasis on security and privacy. Iris is a fully **decentralised** (without central authority) and **pure** (all peers share the same rights and responsibilities) network that allows peers to exchange threat intelligence data. Since Iris needs to spread information fast without saturating the network, we conduct experiments on the speed of convergence of messages in epidemic protocols used in the P2P environment. We implement a working prototype of Iris using Go and the LibP2P project [30], together with a module for the Slips IPS (Intrusion Prevention System). All code is open-source and can be accessed online [41].

The first major contribution of Iris is the concept of organisations. Organisations represent cryptographically-verified trusted groups in the P2P system. Peers can be members of larger specific organisations, or they can create their own trusted organisations. The reason is that some detections and threat intelligence data cannot be publicly disclosed for privacy and security reasons. Therefore, Iris allows peers to authorise particular messages to be given only to specific organisations. Iris uses Distributed Hash Table (DHT) to store members of organisations. This allows peers to easily find other pre-trusted peers from the same organisation. The specific design of Iris allows us to mitigate DHT attacks against an adversarial that wants to prevent peers from finding other members of the same organisation.

The second major contribution is the design of three protocols for sharing threat intelligence data between peers: Alert Protocol, File Sharing Protocol and Network Opinion Protocol.

The Alert Protocol allows peers to disseminate alerts about specific Indicator of Compromise (IoC) in the network as fast as possible. Iris allows addressing alerts only to a set of authorised peers that are part of organisations.

The File Sharing Protocol allows peers to securely share large threat intelligence data. Inspired by the InterPlanetary File System (IPFS), Iris stores information about the *providers* of files in the DHT. A *provider* is a peer that has the file and is sharing it. An important part of the File Sharing Protocol is that it spreads the knowledge that a new file is shared in the P2P network using a novel dissemination

mechanism that respects the access control authorisation mechanism and checks if a peer can receive the file.

The Network Opinion Protocol allows peers to ask other peers about their opinion on a specific IoC. Using the Network Opinion Protocol, peers can share knowledge about resources that have not yet been labelled as malicious. Using this approach, Iris can accelerate the detection of malicious entities because it allows using collective knowledge.

In order to better define how messages should be spread in the P2P system, we conduct an experiment about the speed of convergence of a gossip in the Iris environment while employing different spreading strategies from Epidemic Protocols. The results of the experiment show that it is possible to decrease the number of repeated messages in the network by spreading messages slowly. Iris uses this fact in the File Sharing Protocol when disseminating the information about a new shared uncritical file.

The last and significant part of the thesis is a working implementation of Iris in the Go programming language using the LibP2P project [30]. A special effort has been made to integrate the implementation into Slips (Slips) as a module to provide a collaborative defence.

Our work offers a design of the first P2P network for sharing threat intelligence that takes into account the security of the entire system and considers the privacy of the threat intelligence data. Even though special attention was paid to security, the system may still be vulnerable to a set of attacks related to the storage of file providers in the DHT. These attacks are difficult to mitigate as they are inherent to the characteristic nature of the DHTs and not specifically to Iris.

Iris does not focus on interpreting the credibility of the received data from other peers. Such problem is non-trivial and therefore it has been addressed in a another thesis done in parallel by Bc. Lukáš Forst [17] that proposes *Fides*, a trust model for adversarial collaborative defence networks. Both theses were designed in tight cooperation to assure their mutual compatibility, yet both theses focus on different topics.

To summarise, the contributions of this thesis are:

- The design and implementation of P2P system for sharing threat intelligence and collaborative protection.
- The protocol for requesting information from the P2P network about a specific IoC.
- The protocol for fast alerting peers and distribution of new IoC in the P2P network.
- The protocol for secure sharing of threat intelligence data in the P2P network.

- The design of pre-trusted organisations that support addressing confidential data to trusted groups of peers in the P2P network
- The design of the first P2P system that takes privacy of threat intelligence data into account by sharing it only within a trusted organisation.
- A consideration of adversarial peers in the design of the entire P2P system to enhance the robustness of the network.
- The working implementation of the Iris P2P system along with the integration of Iris into Slips IPS.
- Evaluation of different spreading strategies for epidemic protocols to optimise dissemination of information in the P2P network.

Chapter 2

Background

2.1 Threat Intelligence (TI)

There has not been yet a general consensus on a definition of Threat Intelligence. According to ISO27005 [28], a threat is "A potential cause of an unwanted incident, which may result in harm to a system or organization." and Gartner's Rob McMillan [32] defines Threat Intelligence as "evidence-based knowledge, including context, mechanisms, indicators, implications and actionable advice about an existing or emerging menace or hazard to assets that can be used to inform decisions regarding the subject's response to that menace or hazard".

TI can be either used inside an organisation to understand the background behind attacks, or shared publicly to help the community mitigate similar incidents in the future.

Nevertheless, an important detail is that Threat Intelligence might contain private or sensitive data bound to the intellectual property of an organisation, such as IP addresses, domains, tools, etc. That is why practitioners should be extremely cautious about who are the recipients of the TI. This is because if private TI is disclosed to attackers, it might help attackers adjust their strategies. It is therefore crucial to have the possibility to share specific data only to a specific trusted groups of peers.

In general, there exist four main types of threat intelligence [8]:

- **Strategic** - broader trends that provide high level information typically for non-technical audience and high level decision makers.

- **Operational** - contextual information about security incidents which helps defenders to understand relevant factors like nature, timing, sophistication, intent and motivation of malicious actors.

- **Tactical** - information related to tactics, techniques and procedures used by malicious actors. Such information is useful to design behaviour models of attacks.

- **Technical** - technical resources the threat actors have at their disposal such as tools, IP addresses, domains, malware programs or any other technical details collected and analysed during an ongoing attack or postmortem forensics.

This thesis considers threat intelligence any data that peers share with other peers in the P2P network with the intention of increasing the level of collaborative defence. Nonetheless, most often we specifically mean Technical Threat Intelligence provided by Slips (see Section 2.3.1).

2.2 Cryptography

Cryptography [1] is the study of a secure communication in the presence of an adversarial third party. Modern cryptography explores several aspects of information security such as:

- **Confidentiality** - protection of secret information against disclosure to unauthorised 3rd party
- **Integrity** - assurance that transmitted information has not been altered
- **Authentication** - assurance of a message author's true identity
- **Non-Repudiation** - a property of undeniability assuring an author of the message cannot deny being the author

We can divide modern encryption and decryption schemes into two major groups:

- **Symmetric** cryptography is a type of cryptography with a single cryptographic key that is being used both for encryption of a plaintext and decryption of a ciphertext.

- **Asymmetric** cryptography uses a cryptographic key-pair to perform encryption and decryption.

Asymmetric cryptography is a key aspect of the security mechanisms of this thesis. That is why we elaborate on its mechanism deeper in Subsection 2.2.2. However, firstly, we need to define what a cryptographic hash function is.

2.2.1 Cryptographic Hash Functions

Cryptographic hash functions are mathematical algorithms that map variable sized input to a fixed sized output. The output is usually called a *hash*. The formal definition of the function can be seen in Definition 2.2.1 [46].

A type of keyed hash functions also exist. Those functions take in addition a cryptographic key as a second argument. However, we do not use such functions in the thesis, and thus we concentrate solely on describing the un-keyed hash functions.

Definition 2.2.1 (Hash Function)**.**

$$\begin{aligned} & h : D \to R \\ \text{where} \quad & D \in \{0,1\}^* \\ & R \in \{0,1\}^n \quad n \in \mathbb{N} \end{aligned}$$

Cryptographic hash functions have many applications in computer science and especially in computer security, notably in digital signatures, authentication, stenography, time stamping, etc. [46]. To achieve the security objectives they must satisfy the following properties [1]:

- The computation is deterministic - the same input is mapped to the same output.
- It is a one way function - it is infeasible to reverse the computation.
- It is infeasible to find inputs x and y such that $h(x) = h(y) \wedge x \neq y$
- A small change to the input extensively changes the hash value. The output should appear uncorrelated with the previous hash value.
- Variable-sized inputs map to a fixed sized outputs.

Ideally, if the hash function is correctly designed, given the output C, the only way to find input x such that $h(x) = C$ is to attempt a brute-force search.

2.2.2 Asymmetric Cryptography

Asymmetric cryptography (also called public-key cryptography) uses mathematically related key-pairs to perform cryptographic operations. Each key-pair consists of a *public key* and a corresponding *private key*. Each of the keys, and only that key, can decrypt what the other key encrypts. However, the properties of the private and public keys are not the same and they should not be exchanged.

Effective security requires hiding the private key from everyone except the owner. Protecting the private key from disclosure to unauthorised actors ensures that only the owner of the private key can read the secret information. The public key, on the

other hand, can be disseminated widely and openly without breaching the security. It shall be computationally infeasible to find a private key with only the knowledge of the corresponding public key.

Asymmetric cryptography algorithms are essential security primitives in modern crypto-systems. Protocols built on top of the public-key algorithms offer confidentiality, integrity, authentication, and non-repudiation.

Two of the best-known use cases of asymmetric cryptography are encryption/decryption and digital signatures.

Encryption/Decryption

Asymmetric encryption works by encrypting a secret message using the recipient's public key. Only a person that possesses the corresponding private key is able to decrypt the message. Since we assume that the private key is kept hidden and not disclosed to the public, only the true recipient can decrypt and read the message. This process provides confidentiality of the message but does not reveal any information about an author of the message. A scheme of this mechanism can be seen in Figure 2.1.

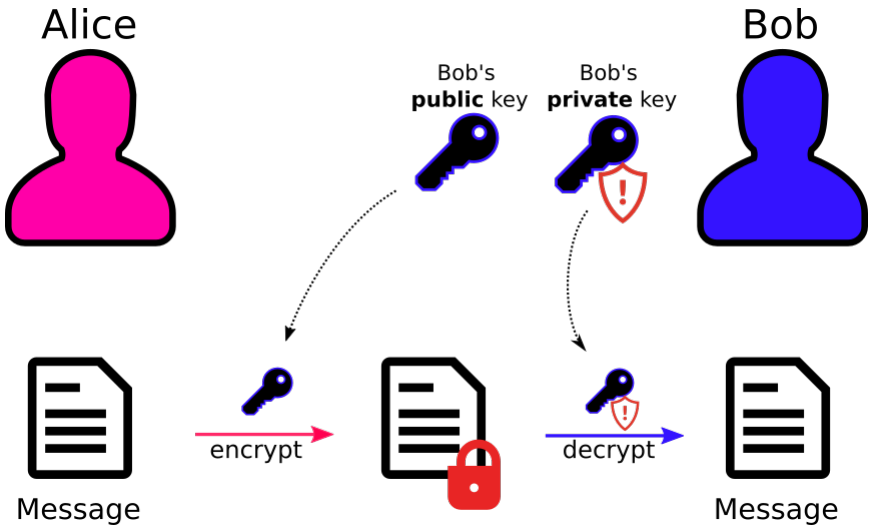


Figure 2.1: Encryption/decryption scheme using asymmetric cryptography

Digital Signatures

Digital signatures provides robust authentication, non-repudiation and integrity assertion for any message. The process (visualised in Figure 2.2) between a sender (Alice) and a receiver (Bob) goes as follows:

1. Alice computes the hash $H = h(M)$ from the message M.
2. Alice computes the signature $S = e_{\text{priv.k}}(H)$ by encrypting the hash H using her private key.
3. Alice sends the message M (M in plaintext) and signature S to Bob.
4. Bob computes the hash $H' = h(M)$ from the received message.
5. Bob decrypts the signature $H = d_{\text{pub.k}}(S)$ using Alice's public key.
6. Bob compares the computed hash H' with the received hash H. If the values are equal, it implies:

   - **authenticity** - only the owner of the private key could have sent the message because nobody else possesses the private key to encrypt the hash H.
   - **non-repudiation** - only the owner of the private key could have sent the message, thus the owner cannot dispute its authorship.
   - **integrity** - the content of the message did not change after the signature was made. Otherwise the hash H' computed by Bob would differ from the received hash.

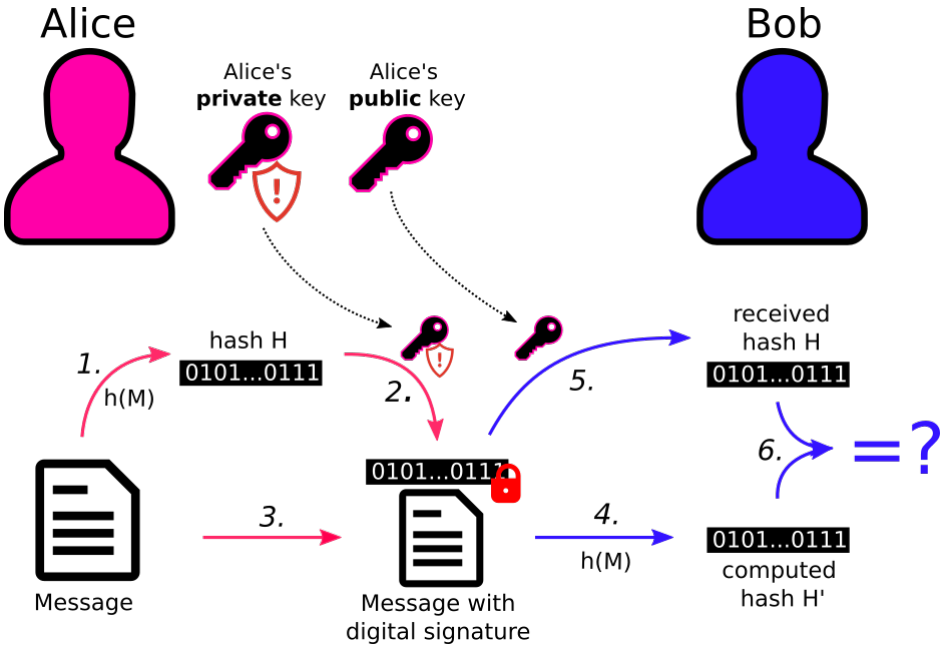


Figure 2.2: Digital signature scheme using asymmetric cryptography

2.3 Intrusion Prevention System

An IPS (Intrusion Prevention System) is a defence mechanism against threats in cyberspace. It monitors actions on the local machine and/or network behaviour to detect threats and then block them. The goal is to either prevent an incident or stop the ongoing attack before any damage occurs.

2.3.1 Stratosphere Linux IPS

Slips is an open-source IPS developed by the Stratosphere research laboratory at the Faculty of Electrical Engineering of the Czech Technical University in Prague [18].

Slips is a modular software implemented in Python. It focuses on analysis of network behaviour using machine learning models to detect threats, such as port scans, ARP poisoning attacks, communication with known malicious IP addresses, and others [19].

Slips is the IPS software for which Iris was designed for.

2.4 Peer-to-Peer Networks

A Peer-to-Peer (P2P) network connects devices directly to each other without the use of a server acting as an intermediary. Unlike conventional client-server communication, no peers in a P2P network are designated solely to serve or receive data. Every device in a P2P network has equal permissions to join and leave the network at any time. This thesis uses the term *peer*, *node* and *device* interchangeably.

2.4.1 Types of P2P Networks

Based on the responsibilities given to the peers, we classify peer-to-peer networks into two types [25]:

- **Pure** - Every peer has the same permissions and responsibilities in the network. Any peer can leave without disrupting the network (e.g. Gnutella [5]). Unless otherwise specified, in this thesis we always refer to pure P2P networks.
- **Hybrid** - A subset of centrally controlled peers is responsible for core functioning features of the network (e.g. Domino [54]). Such peers might, for example, authorise other peers, offer search engine for content shared in the network, etc. Without these peers, the network cannot function to its full extent.

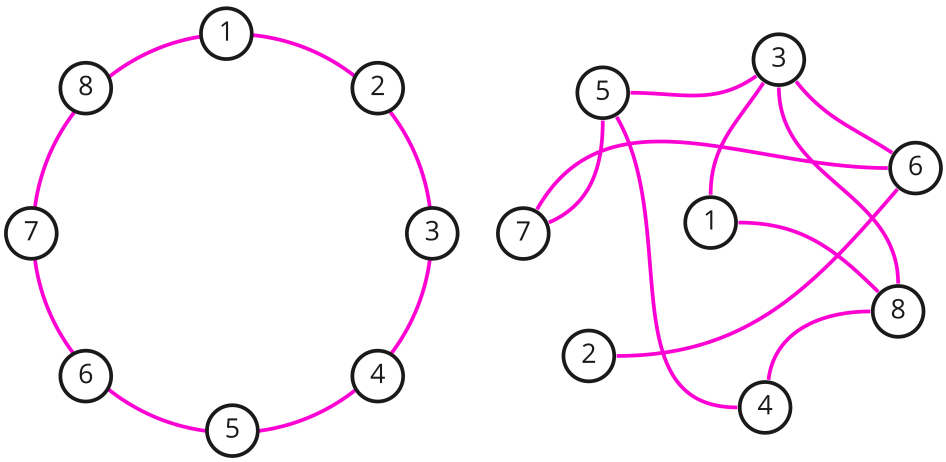
Another way to classify P2P networks is by the topology of the network. If the network enforces peers to be connected in a specific structure, we call the network

structured. Otherwise, the network is called **unstructured** [25]. An example of a structured and unstructured network can be seen in Figure 2.3.

An important variable defining a P2P network is the dynamics of peer participation, also called **churn rate**. Churn defines the frequency of peers joining and leaving the network [47]. It plays an important role in the operation and design of P2P networks.

For example, if the churn rate is extremely high, a structured network would malfunction. The topology would be often disrupted as peers leave, and the network would have to consume resources to re-establish the structure.

However, measuring the churn rate is often difficult, as it requires fine-grained and unbiased information about the arrival and departure of peers [47], and active monitoring of peers in P2P networks is almost impossible to implement.



(a) Structured network (b) Unstructured network

Figure 2.3: Example of structured and unstructured P2P networks

2.4.2 Properties

Due to its inherent characteristics, a peer-to-peer architecture offers the following properties (with a few exceptions discussed later):

- Scalability - easy to scale with large number of peers.
- Reliability - any peer can leave the network at any time without disrupting the network.
- Efficiency - no extra latency added by intermediate server.

On the other hand, the P2P architecture has also its disadvantages.

Pros of P2P Networks

- **No Central Point of Failure** - In a traditional client-server communication, the client cannot function without a properly working server. Which means that the server represents a single point of failure without which the clients essentially cannot operate. In a P2P network, such entity does not exist and any peer can leave the network as it wishes.

- **No Central Authority** - Usually, in a client-server communication, we have no knowledge about logic and business code that runs on the server side of a service we use. For example, the server might jeopardise the clients' security or privacy by storing passwords in a plain text or harvesting user's data for further monetization. That being said, we are involuntary forced to trust the central authority and its good intentions and technical abilities to follow security standards. In contrast, no central authority exists in a peer-to-peer network.

- **Easy To Join The Network** - Every potential peer has the same right to join the network. The only assumption is to know at least one peer connected to the network and follow the protocol. Because of that, P2P networks offer great scalability potential.

Cons of P2P Networks

- **Trust Problem** - The trust problem is one of the major challenges in peer-to-peer networks. How can we identify peers that are trustworthy? How can we detect that peers deliberately lie to our disadvantage? The problem grows if we cannot computationally verify the correctness of the data received from other peers. We do not elaborate further on this problem as it is not in the scope of this thesis.

- **Security** - Designing a secure P2P network is a difficult task as there are many known attack vectors which exist because of the inherent nature of P2P networks. Some of the attacks do not have a clear mitigation and often require relaxation of some key properties of the network. Such as, for example, pivoting from pure network architecture to hybrid network architecture or hardening the process of joining the network. We discuss known attack vectors later in Subsection 2.4.6.

- **Networking Complexity** - Even-though the Internet can be viewed as a huge peer-to-peer network of small networking devices, our P2P application logic is built on top of the Internet topology. Unfortunately, the Internet was not designed with a peer-to-peer network architecture in mind. Many devices

nowadays have not been assigned public IPv4 address and are hidden behind Network Address Translation. This enormously increases the complexity of P2P networks if we want to allow direct communication of two peers that are hidden behind different NATs.

2.4.3 Peer Discovery

Peer discovery refers to the problem of finding initial peers which a new peer can use to join the network. After that, consecutive peers can be discovered through already existing connections. We describe next several known approaches.

Crawling the Internet

The most naive approach suggests to crawl the Internet to find devices that follow the same protocol. Crawling the Internet is not a feasible solution in terms of time complexity. That is why no P2P network actually implements it.

Bootstrapping Nodes

One of the reasonable options is to use static bootstrapping nodes that act as an entry-point to the network. The listening addresses of these nodes are usually hard-coded into the clients. Employing this approach, the network becomes hybrid because a specific subset of nodes have more responsibility than others. Also, if the bootstrapping nodes happen to be controlled by one group of people, the network becomes less decentralised.

TOR [14], for example, utilises a very similar approach called *directory authorities* [12]. Directory authorities are special-purpose peers that periodically publish together with other directory authorities a signed consensus about current active relays. Addresses of these authority peers are hard-coded into the tor clients. Clients then download and verify the consensus and randomly choose relays to establish the connection.

However, the selection of the bootstrapping nodes must be done with caution because if attackers manage to control them, they can essentially control all the connections of their victims. This situation results in a Bootstrapping Attack [16]. The TOR organisation tries to mitigate this attack vector by selecting maintainers of directory authorities with the special heuristics “we-know-you-and-have-had-many-beers-with-you” [48]. More about this attack vector can be found in the Security Subsection 2.4.6.

Usage of DNS

Another option to obtain an initial list of active peers is to use DNS. A list of supported domains is hard-coded into the clients. To find the initial peers, the client queries some of the domains which resolve into a list of IP addresses with active peers. This technique (among others) can be seen deployed in practice in Bitcoin [36].

A big advantage of this approach is that an owner of the domain can dynamically change the returned addresses. The owner can deploy a crawler into the network to keep an up-to-date list of active peers. Random peers from such a list can be rotated in the DNS record.

This technique is very similar to the previous one, with the difference that in this case domains are hard-coded into the clients instead of IP addresses. And since the domains dynamically resolve to different peers, this approach contributes to the decentralised nature of the network and the robustness of the whole system.

Unfortunately, a raw DNS resolution is vulnerable to man-in-the-middle attacks, cache poisoning, stale records, and many other attacks [44], and as such, should be used very carefully. For example, a man-in-the-middle actor could resolve only IP addresses of peers controlled by the attacker, essentially launching a Bootstrapping Attack [16].

Manual Configuration

Another option is to manually configure a listening address of some peer that is used to connect to the network. An advantage of this approach is that we can connect to the network through peers we personally trust or know. However, this approach assumes knowledge of such peer.

Again, the TOR [14] project supports this option via the so-called *bridge relays* [13]. The motivation behind bridge relays is to bypass censorship in a form of firewalls (e.g. Chinese Great Firewall) that block known IP addresses linked to TOR network.

Cache

Smart clients might store peers and corresponding addresses in a local persistent cache for a future usage. Nonetheless, after some time, the stored addresses are likely to become deprecated. Also, enumerating all stored peers might require a considerable amount of time.

Mixed Approaches

To summarise, none of the mentioned techniques is superior to the others. Every technique has its own pros and cons. That is why, for example, the official devel-

opment documentation of Bitcoin recommends [36] the usage of all techniques to minimise the risk of becoming isolated in a malicious network.

2.4.4 Epidemic Protocols

In a common client-server communication, the server acts as a central authority that distributes messages or any other information to clients. In P2P networks, such entity does not exist. That is why we need another method that ensures that information is disseminated to all peers in the network.

A naive approach is to recursively flood all peers with the message. A peer that wants to share a piece of information, floods all the peers it knows with information. Recipient peers do the same until the information propagates through the entire network. However, this approach introduces unnecessary traffic overhead.

Epidemic protocols (also called gossip protocols) try to guarantee information dissemination to the entire network. The epidemic terminology was first introduced in 1987 in [11]. Depending on the nature of information and events in the network, many techniques and optimisations may be applied. Nevertheless, the concept is always the same. Some nodes are aware of the information and some are not. And the ones that know about the message should cooperate in order to disseminate the message to all peers [33].

Many modern systems implement some form of epidemic protocols. For example, the BitTorrent file sharing network [9], and botnets such as Storm and Sality, etc. [42]. Even Amazon reported the use of epidemic protocols to maintain a fully-connected overlay network inside their data centers [10] [33].

As the name suggests, the epidemic protocol is a paradigm inspired by the way viral biological infections spread in a population. Another useful analogy that describes the concept is how gossip spreads in a population. Let us consider a group of office workers and some gossip initiated by one of the workers. If workers form bonds and periodically spread gossip, after a certain amount of time, the gossip reaches all the workers.

The nodes in the network can be in one of three states [33]:

- **Susceptible** - the node has not heard about the message yet.
- **Infected** - the node has heard about the message and is actively spreading it.
- **Removed** - the node has heard about the message but stopped spreading it.

In the basic model (also called Simple Epidemics), three algorithms of spreading exist [33]:

- **Push** - Nodes actively spread the messages by sending them to connected nodes.

- **Pull** - Nodes actively try to learn about the newest updates by asking connected nodes.
- **Push/Pull** - Nodes actively share the most recent update to see if anyone knows a more recent update.

However, for an optimal spreading, we need to figure out:

1. What algorithm of spreading to use?
2. To how many nodes should the infected nodes spread the message at once?
3. How long should the infected nodes wait until they ask/spread again?
4. How do the infected nodes choose recipients?
5. After how much time should the infected nodes move to a Removed state?

Setting these values incorrectly might result in either too slow or too fast gossip convergence. If the gossip converges slowly, it might not be up-to-date when it reaches the whole network (if it even reaches the whole network). On the other hand, if the gossip converges too fast, we might unnecessarily flood the network with messages.

As a consequence, plenty of protocols and their versions exist [33]. Also, every distributed system behaves differently and can guarantee different properties such as network topology, churn rate, etc. That is why it is difficult to re-use some specific protocol for a new system.

Optimally, peers would spread the message only to peers that have not heard about it yet. However, we cannot achieve that because without central authority, we cannot determine who already knows about the message. Hence, we cannot avoid sending repeated messages to peers that have already received the message before.

Therefore, an optimal spreading strategy is the one that optimises the trade-off between flooding the network with repeated messages and speed of convergence.

2.4.5 Distributed Hash Table (DHT)

Distributed Hash Table is a decentralised key-value storage paradigm. In general, hash tables should offer following operations:

- *put(k, v)* - store a key-value pair (k, v)
- *get(k)* - retrieve the value associated with the key k

DHT assigns keys to different subset of nodes in the network that are responsible for storing the corresponding value. This way, every node manages a small part of the global table.

A first specification of DHT was published in 2001 and was called CAN [39]. Very soon CAN was followed by CHORD [40], PASTRY [43] and TAPESTRY [55]. Nowadays, Kademlia DHT [31] with some modifications is the most widely used.

Conceptually, all DHTs work the same. They differ mostly in caching, lookup algorithms, and resilience against common attacks and peers leaving the network. In this thesis, we use the Kademlia implementation, therefore in the following Subsection we focus just on describing Kademlia.

Kademlia DHT

Kademlia [31] requires that identifiers of DHT keys and identifiers of nodes share the same space. This way, we can deterministically map keys to nodes with a notion of closeness. Also, we can calculate the distance between two nodes. In the thesis, we consider that a key space of DHT keys has size of 256 bits.

For the calculation of the distance, Kademlia uses bitwise exclusive OR (XOR). Formally, for two identifiers i_1 and i_2, the distance is calculated as $d(i_1, i_2) = n_1 \oplus n_2$.

The most important procedure nodes have to perform is to find the n closest nodes to a given key. In this thesis, we call these nodes *responsible peers*. Nonetheless, peers do not have the knowledge about all nodes in the network. That is why we need a routing algorithm to locate the responsible peers. Kademlia calls this procedure a *node lookup*.

For the *node lookup* procedure, every node keeps a list of nodes and their listening addresses with distance between 2^i and 2^{i+1} from itself. Kademlia calls these lists k-buckets. For small values of i, the buckets will be often empty since the probability that such peers exist is extremely small. Each bucket has a maximum of k nodes. Every node tries to keep k-buckets full and up to date by periodically sampling a random identifier in the bucket range and launching a *node lookup* procedure with the given identifier.

The *node lookup* is a recursive procedure. When a node n tries to locate responsible peers for a given key k, it picks up to α closest peers from the key k from its k-buckets list and puts them into a list of candidates. Then it asynchronously asks nodes from the list of candidates about the closest nodes from the key k. Each node returns up to α closest peers from its k-buckets list. Received nodes are added to the list of candidates and the search continues until the peer n finds the responsible peers.

As stated before, the *node lookup* procedure is used to keep k-bucket lists up to date. Also, it is used for storing and retrieving values from the DHT. When a peers wants to store a key-value pair (k, v) to the DHT, they use *node lookup* procedure to find l responsible peers. After that, they send value v to these l peers to store it. Similarly, when peers want to retrieve the value associated with the key k, they first

launch *node lookup* and find l responsible peers. After that, they ask these l peers for the value.

Note that we have described a slightly simplified version of the whole protocol to highlight the concept. In addition, Kademlia specifies a caching and a persistent mechanism to prevent losing information when responsible peers leave the network. However, these mechanisms do not play a core role in this thesis.

A nice secondary property is that apart from the distributed storage we can use DHT to search for the listening addresses of peers. Imagine that peer a wants to connect to peer b but it does not know the listening address of b. It only knows the identifier of b. In this case, peer a can use the *node lookup* procedure to find peer b. If peer b exists, it should be found because no other node is closer to b than b itself ($d(b, b) = b \oplus b = 0$).

Security and Privacy of Distributed Hash Tables

The DHT specification does not cover any form of access control. If one node in the network stores a value to the DHT, knowing the key is enough to query the associated value. Thus, privacy-aware networks need to design some form of access control on top of the DHT specification. Kieselmann and Wacker [27] proposed access-control mechanism that can be incorporated into the DHT itself. Nonetheless, their solution only support access control per individual peers, not groups.

Among the most common attacks against DHT [2, 50] are:

- **Eclipse Attack** - Eclipse attack is an attack targeted against a single victim. Usually, attackers try to control all peers that the victim connects to. This can be achieved by poisoning victim's routing table with malicious peers. As a result, the attacker essentially controls the whole communication of the victim. A mitigation of this attack is to protect the routing table from other peers' influence. Kademlia does this by favouring long-living nodes in the routing table.

- **Sybil Attack** - An attacker can generate many specially crafted peers to control all the *responsible peers* for a given key. As a result, the attacker can decide not to retrieve a value for the given key when somebody asks.

- **Churn Attack** - If attackers control a large amount of peers in the network, they can cause a malfunction of the network by disconnecting all the peers at once. This increases the churn rate and forces the network to consume resources to re-establish its structure.

- **Adversarial Routing** - Adversarial routing refers to a situation where the victim's message is routed through adversarial peers. In that case, the attacker

can route a victim to peers under their control and never let the victim find the true closest peers. A mitigation was proposed in [2] by using multiple disjoint paths for each query to minimise the chance of an attacker tampering with all of them.

- **Pollution attack** - In this attack, an attacker can store fake data under the given key in the DHT. This attack can be mitigated by enforcing that the keys for every value are calculated as part of its hash. Then this attack is essentially transformed into finding a collision in the hashing function.

2.4.6 Security of P2P Networks

Peer-to-peer networks are vulnerable to several types of attacks that are mostly caused by the decentralised and anonymous nature inherent to P2P networks. In this section, we talk about known general attack vectors [52, 16].

- **Man-in-the-Middle Attack** - A MitM attack refers to a situation when an attacker impersonates both ends of a communication channel. Both communicating parties have no knowledge about the ongoing situation and trust the communication channel. Depending on the security measures, the attacker can eavesdrop or even alter the messages.

  We can mitigate this by encrypting (assures confidentiality) and signing (assures integrity) the messages. However, if the communicating parties do not possess public key of the other party before the communication starts, we cannot verify that the received public key truly belongs to the benign peer or the attacker. Without a central authority, it is impossible to prevent that.

- **Sybil Attack** - An attacker generates multiple fake identities and deploy them to the network as fake peers. This way the attacker can control a substantial amount of the entire network. It was shown [15] that without a central authority, Sybil attacks cannot be prevented except under extreme and unrealistic assumptions.

  One way to make the attack more expensive for the attacker is to enforce a computational puzzle for every peer before joining the network [2]. The attacker then needs to spend computational resources for each fake identity. However, nothing prevents the attacker to pre-generate rainbow tables of fake identities with solved computational puzzle. To prevent the generation of rainbow tables, we have to link the puzzle to a time and implement expiration time. On the other hand, this approach brutally handicaps small benign devices that do not possess enough computational power to keep recomputing the puzzles.

- **Eclipse Attack** - An Eclipse attack targets either one or a small group of victims. Many types of Eclipse attacks exist, and they depend on the features of each network. However, a general concept is to control as many victim's connections as possible. The attacker can then respond with incorrect responses, route victim's messages to other peers the attacker controls, etc. The mitigation greatly depends on the architecture of each network.
- **Churn Attack** - An attacker can disrupt the structure of a network by substantially increasing the churn rate of the network. Usually this can be achieved if the attacker controls a majority of peers in the network and decides that all of them leave the network. The topology of the network breaks and the network must spend resources to re-establish the structure. Mitigation techniques differ and depend on the architecture of each network.
- **Bootstrapping Attack** - In this attack, attackers try to control the bootstrapping nodes which are used as entry-points into the network. If the attackers achieve that, they can essentially control all the connections that victims make. Mitigation depends on the architecture of the network. The general advice is to make a great effort to keep the bootstrapping nodes secure.
- **DHT related Attacks** - The attacks that relate to DHT have been already described in Subsection 2.4.5.
- **Impersonation Attack** - In this attack, an attacker tries to impersonate other peers in the network. A mitigation is to link a peer identifiers with public keys of a cryptographic key-pair. The private key is then used to sign all messages. This way, the attacker cannot impersonate any peers without owning their private keys.
- **Distributed Denial of Service (DDoS)** - A DDoS attack focuses on making victims' services unavailable either by completely disrupting them or draining all the victims' resources. In P2P networks, such a goal can be achieved by leaving/joining the network with many peers, performing traffic amplification attack or simply attacking the victim directly. To mitigate this attack vector we can deploy mechanisms that allocate resources in a secure way, such as for example, to control the maximum allowed messages sent in a given time period by one peer.
- **Trust Related Attacks** - There are many attacks associated with tempering of victim's trust about other peers [22]. Among those we can find Badmouthing, Unfair Praises, Inaccurate Recommendations, and many others. Since the trust problem is not in the scope of this thesis, we do not further analyse these attack vectors.

2.5 LibP2P

LibP2P [30] is an open-source project that provides a peer-to-peer networking stack. It contains protocols, specifications, and, more importantly, libraries in multiple languages to help developers build peer-to-peer networks.

LibP2P was derived from IPFS (InterPlanetary File System) [4] and focuses on implementing the core features of P2P networks such as NAT traversal, peer routing, content sharing, etc. It offers implementations in Go, Rust, and JavaScript programming languages. Currently, the biggest coverage is offered in Go because of its efficient built-in support for asynchronous networking.

LibP2P is a modular and extensible system. Users can easily substitute the implementation of specific features for their own. This allows building almost any distributed application.

In the following section, we present a subset of the features that LibP2P implements and are essential to our work [29]:

- **Peer Identity** - Every peer owns a cryptographic key-pair. Peers' identifiers are modelled as multihashes [35] of their public keys. Multihash is a special encoding that encodes to one structure a hash along with information about a hashing function that was used.

  Modelling peers' identifiers as hashes of public keys helps LibP2P to mitigate Man in the Middle and Impersonating attacks. Peers encrypt and sign all the messages with their underlying public and private key. The recipient can verify that the sender's public key used in the secure communication actually matches the sender's identifier.

  Also, modelling peers' identifiers as hashes of public keys essentially means that users cannot arbitrarily choose values of peer identifiers to for example control a peer that is closest to a given key in the DHT key space.

- **Transport** - LibP2P is built with the requirement to be transport agnostic. This way, developers can use any transport protocol they wish. LibP2P offers transport implementation for plain UDP, plain TCP, TLS, WebSockets, QUIC [23] and others.

- **Peer Routing** - LibP2P offers also an implementation of S/Kademlia [2] with some modifications [21]. The DHT can be used for peer routing functionality (Described in DHT Subsection 2.4.5) and as a standard distributed storage.

- **Addressing** - LibP2P uses a concept called *Multiaddresses* [34]. Multiaddresses encode multiple layers of addresses into one structure. For example, a multiaddress *'/ip4/1.2.3.4/udp/9000/quic 12D...oJ8*

encodes that we should use IPv4 with address 1.2.3.4, UDP transport protocol with port 9000 and expect QUIC protocol. The last part specifies an identifier of the peer that we want to dial. This is to provide an authentication of the peers we want to dial - they need to provide a correct signature using the corresponding private key.

Chapter 3

Related Work

3.1 Collaborative Intrusion Detection Networks

There has been a constantly growing trend in the number of cyber-threats over the past decades. Along with the number of attacks, the sophistication of malicious actors also increases [53]. Increasing threats drive research towards the field of collaborative cyber-defence.

Specifically, there has been a significant amount of work done in Collaborative Intrusion Detection Networks (CIDNs). A CIDN consists of a set of agents which use collective knowledge and experience to achieve improved intrusion detection. However, every CIDN we know of lacks some key feature with respect to our goal. In this section, we present similar projects and CIDNs that share similarities with our goal and discuss the differences.

DShield [49] is a centralised web repository where registered members can automatically send raw logs from different sources. Later, a group of people analyse the received logs and produces threat intelligence. Dshield is not an automated sharing system, but a central repository of data where members can manually download lists, and that focuses solely on the aggregation of raw logs. It is not peer-to-peer, it does not generate alerts or detections, it does not share data automatically, and it does not consider privacy, confidentiality, encryption, authentication or trust, or verification.

The Malware Information Sharing Platform and Threat Sharing [51] (MISP) is an open source software for receiving and sharing reports. MISP allows administrators to manually connect multiple MISP instances together. By connecting, the instances become trusted, thus MISP allows constructing a semi-centralised architecture. A difference with Iris is that MISP instances serve as middlemen between end-devices and are not fully decentralised or peer-to-peer.

Indra [24], on the other hand, offers a fully decentralised network with agents that monitor the network behaviour, and share data to strengthen its collaborative

defence. Indra allows peers to subscribe in groups based on attacks they are interested in. For example, some peers can join only the *SSH group* to receive only SSH related alerts. However, it operates only on a local area network and with fixed participants. Furthermore, Indra does not address the risk of compromised participants. Compromised nodes can send false alerts or even convince other participants to put any machine into their blacklist. Lastly, Indra authors proposed only ideas and did not conduct any experiment or implement a public proof of concept.

Similarly, Dovecot [22] proposes a fully decentralised P2P network that allows peers to ask other peers about specific Indicator of Compromise. Also, Dovecot implements a trust model that considers compromised or malicious participants. Nonetheless, Dovecot operates on a local area network and does not consider privacy and confidentiality of threat intelligence data.

Other CIDNs such as Domino [54] or ABDIAS [20] work in a hybrid mode. Hybrid networks consist of different types of agents with different responsibilities. For example, Domino consists of agents that are responsible only for monitoring the network and forwarding data to other types of agents for analysis. Note that Domino tried to solve the need of trusted groups because not all captured data can be disclosed publicly for privacy and security reasons. Because of that, all agents are allowed to talk only within its trusted group and another type of agents exists that is designed solely to anonymize and transfer the data between trusted groups. Iris, in contrast, is a pure P2P network and, as such, all peers share the same rights and responsibilities.

The first work that proposes a global and fully decentralised network was NetShield [7]. Netshield focuses on detecting epidemic worm outbreaks and uses Chord [40] DHT to store prevalence of each content block sent in the network. Every agent contributes to the DHT when the prevalence of a certain content block exceeds a local threshold. If a value stored under the given content block in the DHT exceeds a global threshold, a responsible peer triggers an alarm. However, NetShield assumes that all agents are honest and that is why the network does not mitigate any DHT attacks.

Another approach [38] utilises smart contracts [56] deployed to public blockchains [57] to offload computation of dishonest peers and process logs. Nonetheless, using blockchain smart-contracts is expensive and introduces a latency overhead while blockchain blocks are being created.

Hence, to our knowledge, no work focuses on designing a pure, fully decentralised, and permissionless peer-to-peer network for sharing confidential threat intelligence data.

3.2 Content Sharing in Peer-to-Peer Networks

The purpose of the first P2P networks was mainly to share content. The main challenges that content sharing networks try to solve are how to publicly keep track of the peers that offer certain files, how to efficiently download the files, and finally how to verify correctness of a provided file.

A first large P2P network scheme was called Napster [52] and allowed users to download mp3 files. However, the network operated centralised servers that kept track of the peers offering files and paired the peers with these providers to facilitate the download.

Currently, probably the most known file sharing protocol is BitTorrent [9]. BitTorrent addresses the inefficiency problems of inactive peers, which was a common problem in a similar network called Gnutella [5]. The BitTorrent protocol chunks files into smaller parts to allow parallel downloads. BitTorrent's disadvantage is that if users want to download a file, they need to first find a file called *tracker* that lists all the providers of the given file they want to download.

Modern content sharing networks such as IPFS [4] utilise Distributed Hash Table [45] to store information about shared files in a distributed way. Particularly, IPFS keeps track of peers sharing a file in the DHT under a key computed as a hash of the given file. Nonetheless, in order to find the list of file providers for the given file in the DHT, peers need to know the file hash. An advantage of this approach is that every peer can easily verify correctness of a downloaded file by computing the hash of the received content and verifying it with the DHT key which is an expected hash value.

Note that IPFS does not implement any form of access control for shared files. The reason is that DHT itself does not specify any access control mechanisms. Wacker et al. [27] proposed the use of resilient access control for any DHT to allow building privacy-aware distributed applications. However, their solution supports only assigning access-control per individual peer and not per trusted group.

Chapter 4

Design of Iris: P2P System

This chapter describes the general design of Iris P2P System and its integration into Slips [19]. First, we describe the features and overall goal of Iris. Next, we discuss the assumptions that we make about the environment. Furthermore, we show the general architecture of the system with all components and responsibilities. Last, we elaborate on the protocols that we have designed to achieve the aforementioned goals. Iris implementation details are presented in Chapter 6).

4.1 Goals of the System

The high level goal of Iris is to allow Slips instances to communicate directly with each other without any central authority and to securely exchange threat intelligence data. Threat intelligence data might contain confidential information, and that is why peers should be able to share specific data only to a trusted subset of peers.

In following Subsections, we further elaborate on the mandatory requirements:

4.1.1 Functional Requirements

- Peers shall be able to be members of trusted groups to allow message exchanges only within a subset of peers. We introduce a solution called *Organisations* for this problem in Section 4.6.
- Peers shall be able to alert the network or trusted groups of peers with an alert message about an IoC. This information can be used by other Slips instances to block the malicious IoC. For that, we designed a solution called Alert Protocol which is described in Section 4.7.
- Peers shall be able to share threat intelligence files with the network or trusted groups of peers. We propose a File Sharing Protocol in Section 4.8.

- Peers shall be able to ask other peers for their opinion about a given Indicator of Compromise (an IP address, a domain, etc.). To support this feature we propose a Network Opinion Protocol in Section 4.9.

4.1.2 Non-functional Requirements

We consider information security as the main non-functional requirement. In particular, Iris tries to address, as much as possible:

- **Confidentiality** - ensure that only authorised peers can access the data.
- **Integrity** - ensure that the exchanged data are not altered.
- **Availability** - ensure that the system is available.

4.2 Trust Model Assumption

Iris does not try to solve the issue of how to trust other peers and the data in the network, since it is a non-trivial problem. For that reason, there is a parallel diploma thesis done by Bc. Lukáš Forst [17] focusing on this particular issue in P2P networks. Both theses were designed in tight cooperation to ensure their mutual compatibility, yet both theses elaborate individually on different topics.

As a consequence, we assume the existence of a black-box trust model called *Fides* that Iris queries inside each peer to retrieve a value called *service trust* (defined in Definition 4.2.1). The *service trust* value is an estimation of how much a peer can be trusted to provide a good service. Iris uses this value in its design, for example, to favour more trusted peers when downloading a TI file. For more details about the computation of trust, we refer the reader to the Fides thesis [17].

Definition 4.2.1 (Service Trust)**.** Service Trust st_p denotes a belief about how much we trust that a given peer p will provide a good service. It is a real number between 0 and 1 where a value closer to 1 represents larger trust. On the other hand, a value closer to zero stands for no trust all.

$$st_p \in [0, 1]$$
$$p \in \{p_1, ..., p_n\}$$

Fides provides Iris with the option to report misbehaviour of any peer (such as providing incorrect files, not following the defined protocol, etc.). The reporting of peers should result in decreasing their *service trust*. That is why Iris does not need to remember the reputation or trust of peers.

Iris does not manipulate the application data shared between peers. It treats the content as a raw stream of bytes. Iris serves solely as a networking layer that is responsible for delivering the messages and the security of the system.

4.3 Architecture Overview

In total, there exists three major components in Iris, as shown in Figure 4.1:

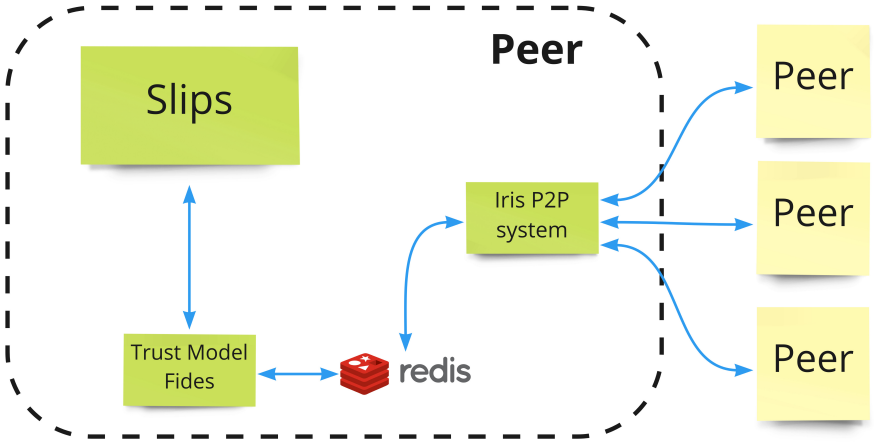


Figure 4.1: Diagram of the communication flow between components in the system

- **Slips [19]** - Slips is a modular IPS implemented in Python. The Slips instance monitors the behaviour on the local machine and in case of suspicious events might decide to ask the network for advice, share threat intelligence data or alert the network. Slips uses Redis as a database system.
- **Fides Trust Model [17]** - Fides is implemented as a Slips module in Python. If Slips wants to interact with the network, it asks Fides. Fides forwards the task through a Redis channel to Iris and subscribes for a reply in the Redis channel. Fides determines the credibility of the aggregated peers' replies and returns it back to Slips.
- **Iris P2P System** - Our networking stack that provides direct communication with other peers. Iris is responsible for secure communication, privacy of transferred data, and delivery of requests and corresponding responses. Iris is built on top of the LibP2P[29] project that provides libraries for P2P networking. Figure 4.2, shows an analysis of responsibilities between Iris and the LibP2P project.

4.4 Peers Joining the Network

The mechanism for new peers to join the network is problematic. All methods that we have described in the Background Section 2.4.3 either introduce more attack

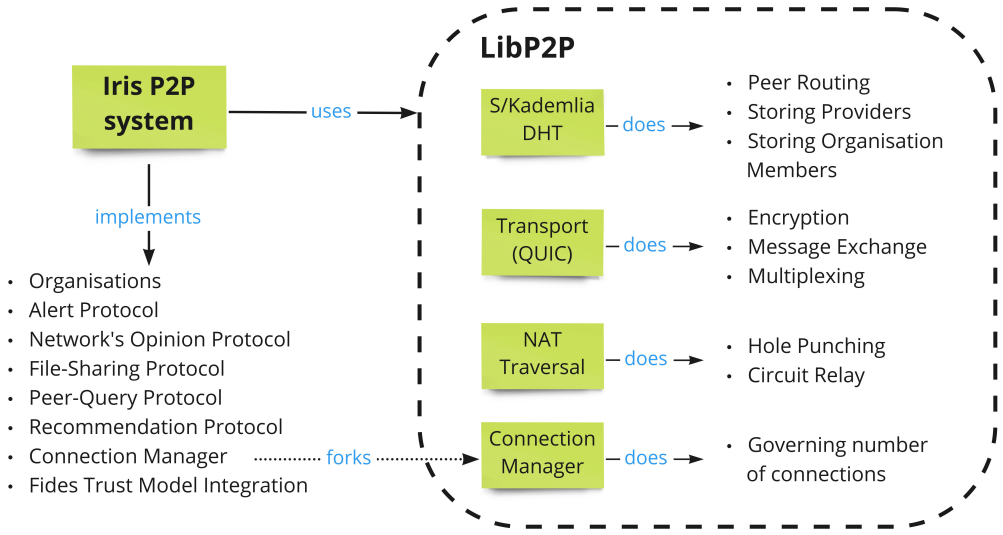


Figure 4.2: Diagram of responsibilities between Iris and LibP2P project

vectors or make the network hybrid or less decentralised. Even though it is possible that the best approach would be to use a mixture of methods, we only implemented bootstrapping nodes and the manual configuration of known peers and organisations.

4.5 Iris Usage of Distributed Hash Table

The Distributed Hash Table plays an essential role in Iris. We use S/Kademlia DHT implemented with some modifications by LibP2P project. The DHT in our system serves three purposes (the security aspect of all use cases is discussed later in this Chapter):

1. **Store providers of TI files**. The DHT stores the providers of every shared file in the system. Section 4.8 describes the details of the proposed File Sharing Protocol.

2. **Store members of organisations**. The DHT stores information about pre-trusted groups of peers. Section 4.6 describes details on the concept of *Organisations*.

3. **Peer routing**. Peer routing is a mechanism to locate peers in the network using only their identifiers. We have described the process in the Background Section 2.4.5. LibP2P project already supports this feature, so it is not implemented in Iris.

4.6 Organisations as Trusted Groups

Iris introduces a new concept for P2P networks called *Organisations*. An organisation represents a trusted group of peers in the P2P network. The main incentive to create them is that security practitioners on the Internet naturally form groups, where they know each other, trust each other to some degree, share common interests, and share data. It is a basic methodology to counteract adversaries. Therefore, having cryptographically-verified groups was necessary for a security P2P network.

There are two main motivations behind this concept:

- Organisations allow Iris to exchange data only with peers within a specific trusted group.
- Organisations provide the initial ground-truth of who to trust in the trust model. Note that Fides, the underlying trust model, takes organisations' memberships into consideration and uses this to compute the value of *service trust* for a given peer.

4.6.1 Definition of Organisation

Internally, Iris identifies an organisation in the same way as LibP2P defines identifiers of peers, which is by using a cryptographic key-pair. An identifier of an organisation is then a multihash of its *public* key.

Apart from its simplicity, identifying an organisation like this also ensures that peers' identifiers and organisations' identifiers share the same key space, which is an important fact in organisation member discovery and secure storage (Subsections 4.6.2 and 4.6.3, respectively).

Any peer p can become a member of an organisation o by having its ID digitally signed by the organisation's *private* key. Later, when the peers introduce themselves to other peers, they present also the signature and the organisation's ID. If the organisation matches any organisation in the receiver's trusted list, the receiver can verify the signature and consider given peer as a member of the given organisation.

Note that a peer does not have to be a member of an organisation o in order to trust peers that are members of o. A knowledge of the organisation's identifier is enough to verify other members.

4.6.2 Discovery of Organisations and Organisations' Members

Iris does not provide any way to trustworthily disclose organisations identifiers to users. Also, Iris does not provide users with a way to verify that an organisation's identifier truly belongs to the given organisation in the real world. For this, every organisation should choose any technique they wish to ensure trustworthiness of the

published ID, such as using a Central Authority from a Public Key Infrastructure to issue them a certificate, or use public social networks. It is then the users' responsibility to verify the correctness of a given organisation ID before launching a peer that trusts the given organisation.

After joining the network, it is in the new peer's interest to establish a connection with at least some peers from the trusted organisation. The reason is that if the peer trusts their organisations, there is a bigger probability that these peers will not be under the control of an adversary. It is the only heuristic the new peers have because otherwise they end up talking to complete strangers.

Since peers would like to find other members of the organisation they trust, Iris should provide a way to achieve that. A naive approach would be to traverse the whole network as a graph using depth-first or breadth-first search until the peer finds at least a minimum number of such peers. However, with a bigger sized network, this approach introduces unnecessary network overhead.

Therefore, Iris uses the DHT for automated discovery of organisation members. This is done as normally the DHT would be used to locate file keys, but since the id of an organisation shares the same key space, organisations can be found in the same way.

Organisations can also publish the identifiers of its peers on the Internet in any way they want. That way users can use publicly known peers in their configuration to connect directly to these trusted peers.

Storing Members in the DHT

Since we have defined the organisations' IDs from the same space as peers' IDs, we can use organisations' IDs as keys in the DHT. As the corresponding value for the key we store a list of peer IDs that belong to the given organisation.

To support this feature, we define two new methods that use the DHT:

1. *membership(o)* - using this method a peer advertises themselves as a member of organisation o in the DHT. To do so, the peer launches a *node lookup* procedure with the key o to find responsible peers for the key o. After that, the peer sends to all responsible peers a claim of being a member of o with a digital signature. The responsible peers verify the signature and if the signature is correct, they append the peer to an already existing list of members, or they create a new one if necessary.

2. *members(o)* - using this method a peer can find advertised members of an organisation o. Firstly, the peer finds all responsible peers for the o key. After that, it asks the responsible peers about the value of the corresponding key o, which contains the members.

For this method to work, whenever peers that are members of some organisations join the network, they shall advertise themselves as members of the organisations in the DHT using the *membership* method. Later, whenever any peer wants to connect to some pre-trusted peers, it can query the DHT using the *members* method.

4.6.3 Security of Storing Organisation Members in DHT

Since DHT plays an important role in our organisation mechanisms, we need to consider all the threats that come along with it. We must address two security risks: attackers controlling the responsible peers for an organisation, and poisoning the DHT with fake members of an organisation.

Attacker Controlling Responsible Peers for a Given Organisation

An attacker can try to control the information in the DHT by controlling the peers that are responsible for storing a list of organisation's members. This would have fatal consequences and could result in a lost communication between organisation members because the malicious peers could return anything instead of the list with organisation's members.

Remember that responsible peers are the ones which IDs' are closest to the stored key. Thus, the attacker would have to control the peers that are closest to the given key. However, it is enough for the attacker to be *closer than everyone else* in the network. And the list of currently closest peers in the network is fairly easy to acquire by simply launching the *node lookup* procedure.

The cost of this attack grows with a size of the network because if there is a large number of peers in the network, the attacker has to generate large number of peer IDs in order to find and control the closest ones. However, as has been shown in [37], that such attack may take only some seconds to find the closest peers in the IPFS network, which already has thousands of active peers. Most probably our network will be even smaller, that is why we have to consider this attack vector very seriously.

That being said, our goal to mitigate this attack is to prevent the attacker controlling the closest peers to the DHT key. But since the DHT key is actually an organisation ID and we generate the organisation ID the same way as peer ID, we essentially own the peer that has ID with distance zero from the organisation key because $d(x, x) = x \oplus x = 0$.

That is why, if we deploy a peer into the network with the same ID as the organisation ID, the attacker can never control all the closest peers. Utilising this technique, we can effectively mitigate this attack vector.

Also, if there is a peer in the network with the ID that equals to the organisation ID, we can automatically consider this peer non-adversarial and use them straight

away as a responsible peer. Otherwise it implies that the attacker controls the *private* key of the organisation and the whole trust mechanism of the organisation is breached.

Poisoning the DHT with Fake Organisation Members

An attacker could try to generate large number of sybil peers and falsely claim to be a member of any organisation to poison the list with members in the DHT. However, in order to poison the correct list of organisation members, the sybil peer needs to provide the true ID of the victim organisation (because the ID is used as a key in the DHT). And since the organisation ID is also a public key, the responsible peers can easily verify if the sybil peer owns a correct signature generated by the organisation's private key. Sybil peers will not be able to provide such signature and thus they will not be able to poison the DHT.

The only option is that the attacker controls the responsible peers to skip the verification process. However, if the attacker controls the responsible peers, we talk about the attack we have just described in previous Subsection.

4.7 Alert Protocol

The Alert Protocol is one of the contributions of Iris that provides alerting to other peers in the network about an IoC. The idea is that when Slips confidently detects an Indicator of Compromise (such as IP address, domain, etc.) it wants to warn other peers in the network. When a peer wants to block a new attack, the expected life of the IoC is measured in days at most (and hours at least), therefore we assume that peers want to receive the alert as soon as possible. Note that the content of alerts is irrelevant for Iris.

Peers might want to share the alerts only within a trusted organisation. There are several reasons for that. One of them is that alerts contain confidential data. Another one is to prevent attackers from eavesdropping on the alerts of their victims to adapt their behaviour. For that reason, Iris allows to specify authorised organisations of alert messages. If an alert is addressed only to a subset of organisations, peers spread the alert only to authorised peers along with information who is authorised to receive the alert for further propagation.

To understand how the Alert Protocol can spread different messages better, we have conducted experiments (Chapter 5) to see if we can optimise the configuration of spreading algorithms with the *service trust* values provided by Fides.

4.8 File Sharing Protocol

Another contribution of Iris is a File Sharing Protocol to share threat intelligence data. The goal is to notify peers in the network about a new available file in a reasonable time and to allow all authorised peers to download the file, therefore to design some form of a reading access control per organisations.

A naive approach would be to propagate the file content similarly as the alerts described in Section 4.7. However, that would introduce unnecessary network overhead, because not every peer wants to download every available file for various reasons - connection bandwidth limitation, no interest, etc.

The special need is that peers do not want to distribute the content of files to everyone in the network, since peers may not want or need that threat intelligence. Because of that, metadata of the shared file is first published, and the peers that want that threat intelligence file can ask for it in the P2P network.

Iris File Sharing Protocol stores providers of files in the DHT, similarly as IPFS [4]. Storing providers in the DHT is practical because peers can easily advertise themselves as providers just by writing a value into the DHT. Also, any peer can easily query the DHT to obtain an up-to-date list of providers for the given file. However, storing the providers in the DHT is not sufficient because we need to also somehow notify peers in the network about the existence of the file in the first place.

An overall diagram of how to share a file can be seen in Figure 4.3.

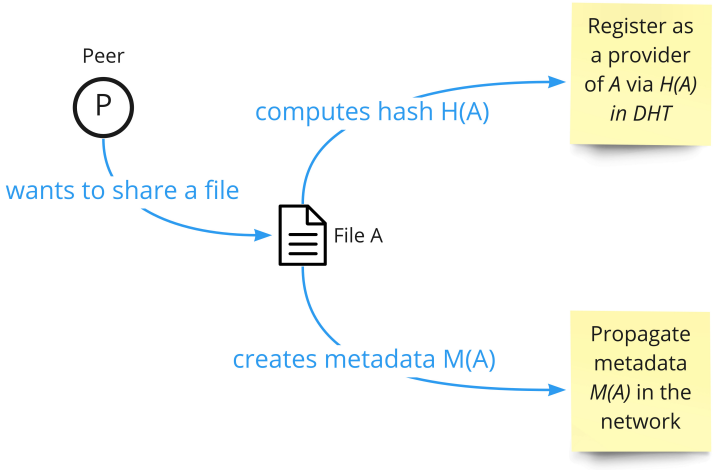


Figure 4.3: A high-level diagram of a peer sharing a file in File Sharing Protocol

4.8.1 Storing File Providers in the DHT

DHT stores information about the peers that share certain files, called file providers. The files themselves are not stored in the DHT, but the file hash is used as a key in the DHT. Thus, for every shared file with a hash f in the network, there shall be a list of provider peers l stored in the DHT as a key-value pair $(f : l)$.

Iris implements two methods using the DHT to support this functionality:

1. *provide(f)* - A peer can claim to be a provider of a file f by calling this method. As a result, the responsible peers for they key f append this peer to the list of providers of the f key (or create a new one if the list does not exist). Note that responsible peers cannot verify this claim as they may not be authorised to access the file. Figure 4.4 depicts this process.

2. *providers(f)* - A method to query the DHT for a list of providers for the given key k. Internally it launches the *node lookup* procedure to find responsible peers and ask them for a corresponding value - the list of providers.

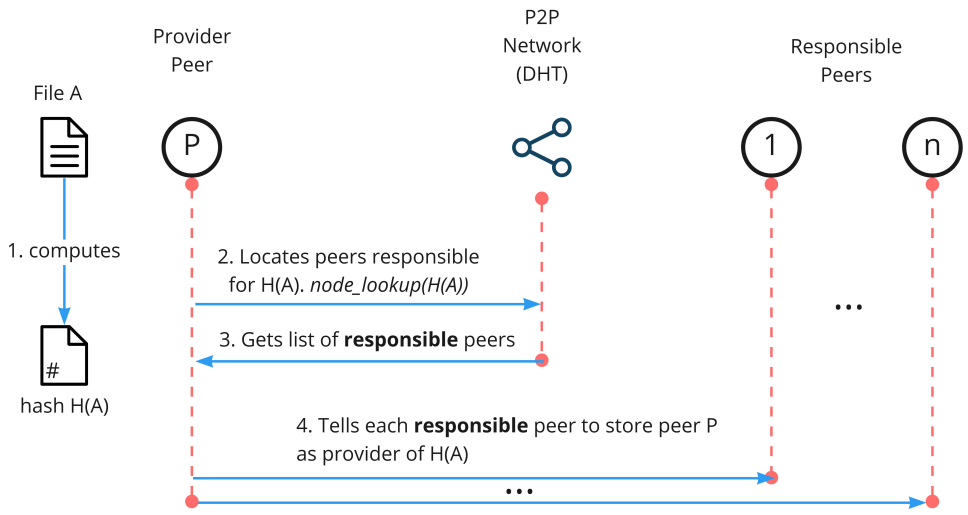


Figure 4.4: Diagram of how to store a new provider peer for a file in the DHT

4.8.2 Notification of Authorised Peers

Iris uses Epidemic Protocols (similarly as for Alerts in Section 4.7) to spread metadata of files to notify peers about their existence. An example can be seen in Figure 4.5a. The metadata contains a description of the file and the hash of the file that recipients

can use to query the DHT to find a list of providers of the file. This way, every peer can decide based on the received metadata file whether it wants to download the file or not.

Spreading algorithms depend on how the set of peers that receive a message are chosen. In the case of metadata files, should the metadata be shared with the whole network? We conclude that not. The reason is that by disclosing the threat intelligence metadata to unauthorised peers, we might face a side channel attack. A smart attacker could monitor the messages originating from its victims to learn about their defence mechanisms and adapt to their behaviour.

For this reason, Iris adds a list of authorised organisations into the metadata message and peers must only forward the metadata to members of authorised organisations. In other words, if only organisation O is authorised to download a file, then only the members of the organisation O should receive the metadata message. A diagram of sharing metadata only to authorised peers can be seen in Figure 4.5b. This requires every peer to have always at-least one open connection with members from their organisations.

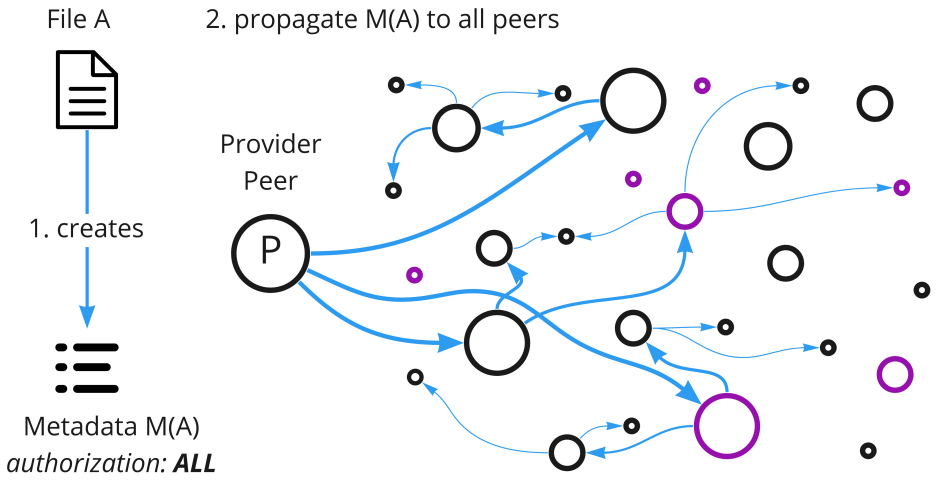
However, not all files share the same importance. For example, all authorised peers should promptly find out about a new file with high risk threat intelligence even at the cost of flooding the network with messages. On the other hand, less important threat intelligence files can take longer to spread without any cost. Therefore, we propose two severity levels of shared files. By setting the appropriate severity level, the network should guarantee to spread the metadata message to all authorised peers in a different way:

- **CRITICAL** - by setting a file to severity CRITICAL, the network should guarantee to spread file metadata to all authorised peers as fast as possible.
- **NORMAL** - by setting a file to severity NORMAL, the network should also guarantee to spread the file metadata to all authorised peers but try to minimise flooding of the network with messages.

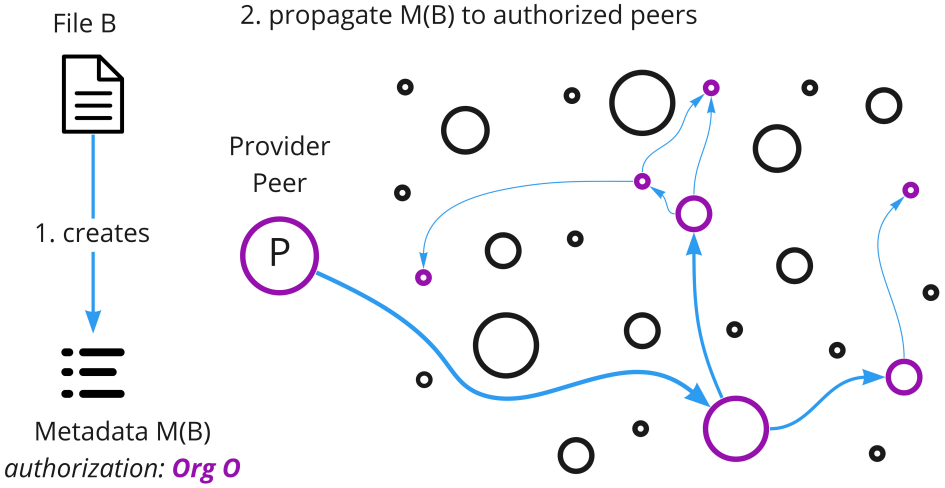
Separating threat intelligence files into two severity levels may be useful to use different spreading algorithms and thus optimise the use of network's resources. We conduct an experiment to answer this question in Chapter 5.

4.8.3 Reading Access Control

Even-though we share metadata of files only to authorised peers, it does not prevent unauthorised peers from knowing the IDs of the shared files. For example, we cannot guarantee that the responsible peers that store provider records in the DHT are authorised to access the given file. Most probably the responsible peers will be



(a) Example of an epidemic propagation of file metadata to all peers



(b) Example of an epidemic propagation of file metadata to authorised peers from one organisation

Figure 4.5: Examples of an epidemic propagation of file metadata

unauthorised to access the file. And we cannot prevent responsible peers from disclosing the file hash. After the possible disclosure of a file hash, anyone can query the providers in the DHT and try to download the file from them.

Thus, providers themselves have to verify that every peer that wants to download a file has authorisation to do so. Every peer can ask to be authorised by providing the organisation's signature. The *provider* peer verifies the signature and eventually decides to sends the file or not.

In the case that the shared file has no access control restrictions, providers do not demand any form of authorisation. Also, metadata of this file can be spread to all peers in the network.

4.8.4 Downloading Files

To download a file, peers query the DHT to get a list of providers for the given file. Peers then choose a provider from the list and try to download the file. If the file has access-control restrictions, peers also present their organisation signatures that authorise them to access the file. If the provider provides an incorrect file, the peer needs to try another provider from the list. The diagram of this process can be seen in Figure 4.6.

After downloading a file, peers can decide whether they want to be listed in the DHT as another providers of the file. By advertising themselves as new providers, they help an original provider because suddenly more peers allow to download the file from them.

4.8.5 Security of the File Sharing Protocol

There are two major security risks related to the use of DHT for file sharing: an attacker controlling the responsible peers for a file, and poisoning the DHT with fake providers of the file.

Attacker Controlling Responsible Peers for Given File

An attacker can try to control the information in the DHT by controlling the responsible peers. As a consequence, this would lead to a disruption of a service because peers would not be able to query the providers - thus to download the file. Remember that, to control the responsible peers, the attacker needs to own peers that are the closest to the given key in the DHT key space. The cost of this attack grows with the number of peers in the network. But as shown in [37], even in the IPFS network [4] an attacker can target one DHT key in a matter of minutes.

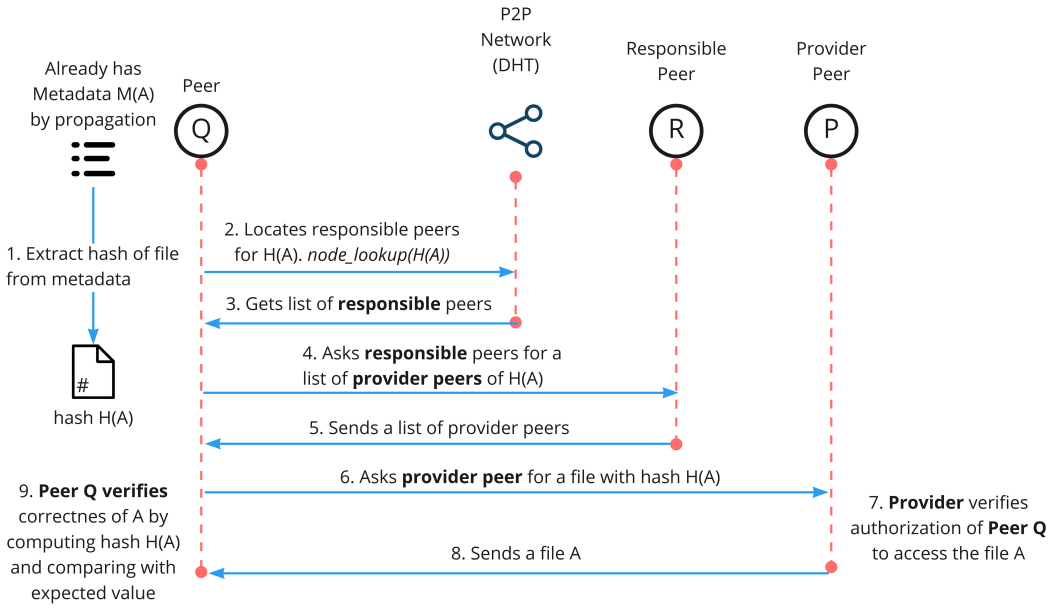


Figure 4.6: A diagram of a peer downloading a shared file

However, this attack assumes that the attacker knows about the file hash, otherwise it cannot attack the hash key in the DHT. On the other hand, we cannot prevent the attacker from discovering the file hash as the file hash is not secret information.

This attack can be mitigated a little bit by chunking files into smaller chunks (we further discuss this feature in Future Work Section 7.1). Nonetheless, our implementation does not implement chunking and thus is vulnerable to this attack vector.

Poisoning the DHT with Fake File Providers

An attacker might generate many fake peers that all claim to be providers of a certain file. And as shown in [37], the generation of fake peers in the IPFS P2P network is extremely easy. Peers responsible for storing the list of providers in the DHT may not be able to verify these claims as they may not be authorised to read the file and verify the hash.

As a result of this attack, when a victim peer asks for a list of providers of a specific file, it receives a list of peers, but the majority of them may be fake. The victim then needs to spend a non-trivial amount of time, filtering out the fake peers by trying to download the file. A fake file can be recognised by verifying that the hash of the given file does not equal to the ID of the file. However, peers have to download the whole fake file first in order to detect that the content is wrong. This essentially results in a DoS attack. A diagram of this attack can be seen in Figure 4.7.

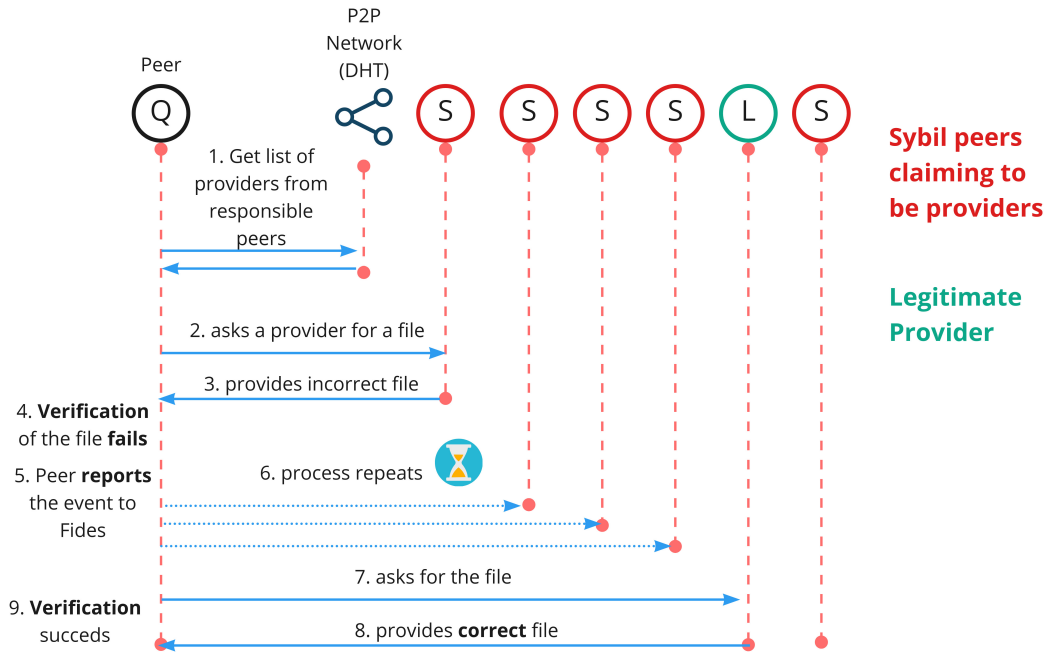


Figure 4.7: A diagram of downloading a file from a list of sybil providers controlled by an attacker

In our environment, Iris relies on *service trust* value provided by Fides trust model for choosing specific providers from the list of all providers to download the file. If these providers give us incorrect files, Iris reports them to Fides, and future interactions with them will be less trusted. If Fides works correctly, in the long run, the legitimate providers should possess a bigger *service trust* and thus we should be able to promptly find the correct providers. For more information about Fides, see Section 4.2.

4.9 Network Opinion Protocol

Iris implements a protocol to allow peers to ask other peers about their opinion on a specific IoC. The idea is that the Slips instance encounters a suspicious resource (e.g. an IP address, a domain, etc.) and needs to asks other peers for an opinion.

4.9.1 Design of Network Opinion Protocol

This protocol was designed so that a peer always asks more peers than just the ones it is connected to. However, a peer cannot ask the whole network. A peer doesn't

know the size of the network and the number of responses could be enormous. In the worst case scenario, a peer could even launch a DDOS attack against itself.

That is why this protocol was designed in a way that spreads the *request message* in an epidemic style, but containing the outbreak by automatically terminating it after some time. The termination is implemented in the Network Opinion Protocol messages by a special field called *time to live* (TTL) value. The peer that asks for an opinion, sets the initial TTL value of the request and sends the request to a set of peers. Every peer that receives the request, decrements the TTL value by one, and forwards the request to other peers. This propagation is done recursively until the TTL reaches zero. When the TTL reaches zero, the current peer stops further propagation.

Every peer that received the request (no matter if the peer propagated the request further to other peers or not) also forwards the request to a local Slips instance to acquire a local opinion. The peer signs the local opinion with its private key and encrypts the local opinion using the original requester's public key. This is very important for the overall confidentiality of the Iris P2P network. No message can be opened by any intermediary peer that is not the originator of the request.

After that, peers collect together the local encrypted opinion and opinions from responses of other peers. The accumulated opinions are returned as a response to the sender. This way, no peers can see the contents of opinions of other peers because every opinion is encrypted using original requester's public key.

Lastly, every peer that receives the request should make sure that it had not already processed the request before. If yes, the peer should not process the request again. Otherwise, peers could do the same work multiple times because they might be asked multiple times by different peers.

An overview of just described process that propagates the opinion request can be seen in Algorithm 1 and in Figure 4.8.

Nonetheless, two questions regarding the forwarding of opinion requests arise:

- **How many opinions does a peer want to acquire about a potential malicious IoC?** For a rigorous answer, we should find a heuristic that would provide us an approximation to the probability of two random peers dealing with a similar attack. Using this heuristic, we could decide how many opinions a peer needs in order for some of the opinions to be useful. However, the answer for this question is out of scope of this thesis and left for future work. For our current work, we approximate that an ideal number of total received opinions is around 100.

  In Iris, using the previously described process of opinion request propagation, the total number of recipients is calculated as the number of nodes in the spreading tree minus one (the original requester) - see Equation 4.2. For our

Algorithm 1: Propagation of opinion request through the network

```
1  Function getOpinions(rsc)():
2      req ← generate request for opinion on resource rsc
3      req.sig ← digitally sign the request
4      req.ttl ← initial value of TTL
5      recipients ← sample recipients for the request
6      opinions ← empty list
7      for r ∈ recipients do
8          msg ← encrypt req using public key of r
9          resp ← send msg to r
10         opinions ← decrypt opinions in resp using local private key
11     end
12     return opinions
13
14 Function receive(msg)():
15     req ← decrypt msg using local private key and verify signature
16     if req was already seen by local peer then
17         return ALREADY PROCESSED
18     end
19     opinion ← get and digitally sign local Slips opinion
20     resp ← encrypt opinion using original requester's public key
21     if req.ttl > 0 then
22         req.ttl ← req.ttl − 1
23         recipients ← sample recipients for the forwarded request
24         for r ∈ recipients do
25             msg ← encrypt req using public key of r
26             resp_r ← send msg to r
27             resp ← add items from resp_r
28         end
29     end
30     return resp
31
```

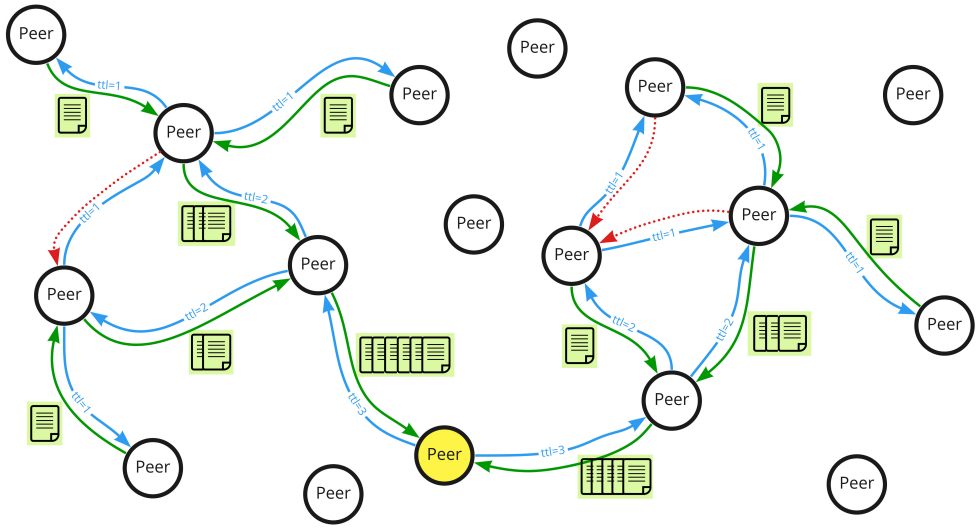


Figure 4.8: **An example of propagation of opinion request in Network Opinion Protocol.** An example of propagation of opinion request with initial TTL value=3. The request originates in the yellow peer. Blue edges represent the flow of opinion request and are labelled with current TTL value. Green edges represent successful replies with accumulated opinions. Red edges represent unsuccessful responses because a receiver already processed the request.

implementation, we use a strategy that forwards a request to 3 peers with an initial value of TTL=4. This results in a total number of recipients being up to 120 ($\sum_{i=0}^{4} = 3^i - 1 = 120$).

$$\sum_{i=0}^{TTL} = n^i - 1 \tag{4.2}$$

- **How should peers choose which other peers are the recipients of the opinion request?** One approach is to choose recipients completely randomly. Another approach is to use *service trust* value from Fides as a heuristic when sampling recipients from the list of candidates. Since *service trust* is defined exactly as the belief that peers would provide a good service, it is a rational value to use for choosing recipients.

  It is also important for peers not to ask every time the same peers about an opinion. We would like to allow a bit of variability that would slightly favour the more trusted peers. That is why Iris chooses recipients from candidates with a probability that is exponentially weighted with the candidates' service

trust. Iris does this by firstly transforming service trust values of all candidates using exponential function $f(st) = \frac{a^{st}-1}{a-1}$ with a constant value $a = 10$ (see the function plotted in Figure 4.9). After that, Iris normalises the transformed values which represent the probabilities of the candidates being chosen.

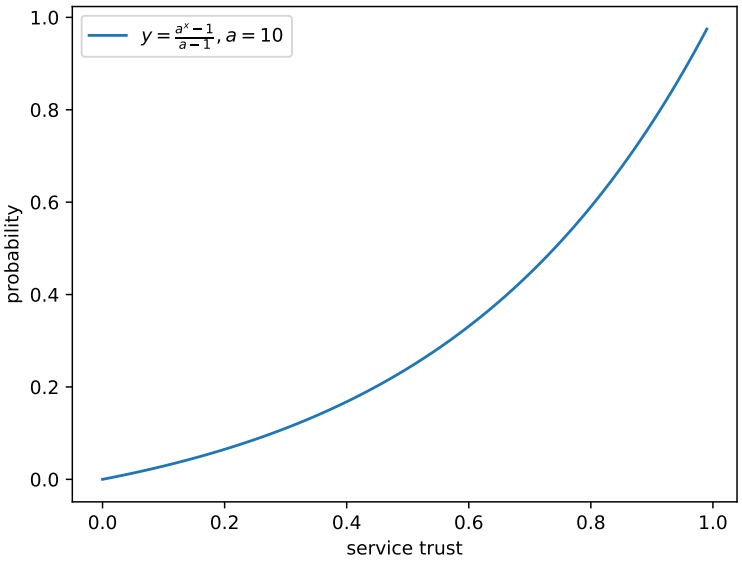


Figure 4.9: An exponential function $y = \frac{a^x-1}{a-1}$ with a constant value $a = 10$

4.9.2 Security of the Network Opinion Protocol

Encrypting and signing the messages provides confidentiality and integrity of the opinions. Every peer knows who the original requester was and cannot tamper with the forwarding request. Also, no intermediate peer can see into the opinions that are not addressed to him. This allows peers to respond with confidential data because nobody except the original requester can see the data. Also, every opinion is digitally signed by opinion's author and that is why the original requester knows who provided the opinions.

The only potential attacks done by adversarial peers using this protocol are based on lying to mislead the peer that requested the information. However, this is not an issue for Iris, but for the Fides trust model, and it is addressed in that work.

Chapter 5

Experiment on Optimal Epidemic Spreading Strategy

The Iris P2P system proposes new ways to deal with the security concerns of threat intelligence sharing. However, many of these ideas need to be verified and explored in simulated experiments in order to understand how a network of peers may behave under different conditions. We simulate and evaluate different spreading strategies in an epidemic sense. As a result of this experiment, we choose the optimal spreading strategy for each different condition and for the protocols for spreading Alerts (defined in Section 4.7) and for spreading file metadata with different levels of severity (defined in Section 4.8).

We decided to run one very large experiment with many parameters instead of a large number of small experiments. In this way it was possible to validate how the conditions relate to each other.

5.1 Goal

The goal of this experiment is to **evaluate** different **spreading strategies** based on how fast they spread the message in networks and how much they flood networks with messages. To define a spreading strategy, we first need to refer to the Background Chapter 2.4.4, where we state that for the optimal spreading, we need to answer five questions:

1. What algorithm of spreading to use?

   In our case, only the **Push** algorithm makes sense. The others assume that gossip is constantly updated in the system, and thus peers proactively ask other peers for updates. That is not true in our environment - it might happen that

no gossip is spread for a long time. In that case, the proactive update messages in **Pull** and **Push/Pull** algorithms are unnecessary.

2. To how many nodes should infected nodes spread the message at once?

   We will call this value a *spreading factor*. Finding the optimal spreading factor is one of the goals of the experiment.

3. How long should the infected nodes wait until they ask/spread again?

   We will call this value *spreading period*. Finding the optimal spreading period is one of the goals of the experiment.

4. How do infected nodes choose recipients for spreading from a list of candidates?

   Among choosing the recipients completely randomly, we can utilise the service trust of candidates to design a better heuristic. We propose 3 options:

   (a) Choose recipients with a uniform probability. Most of the Epidemic Protocols employ this option.

   (b) Sort recipients based on their service trust and choose first the most trusted ones. This technique assumes that peers with bigger service trust will more likely follow the protocol and thus contribute more into a dissemination of the message.

   (c) Choose recipients with probability that exponentially grows with service trust of each recipient. This technique was already described in Subsection 4.9.1 when we talked about choosing recipients of recipients in Network Opinion Protocol. It works firstly by transforming service trust values of all candidates using exponential function $f(st) = \frac{a^{st}-1}{a-1}$ with a constant value $a = 10$ (see the function plotted in Figure 4.9). After that, Iris normalises the transformed values which represent the probabilities of the candidates being chosen. This technique still favours the trusted peers but allows also to choose less trusted peers for a bigger variety. The reason is that the option (b) might result in flooding the trusted peers and not spreading the message to less trusted parts of the network which may also contain benign peers.

5. After how much time should the infected node move to a Removed state?

   We will call this value *spreading expiration*. Optimally, the spreading should stop after the gossip reaches the entire network. Nonetheless, our network is decentralised and does not provide complete information about status of all peers. For this reason, the spreading expiration should be set beforehand to a duration that guarantees the full convergence. Note that the spreading

expiration depends on the previous points because they essentially define the speed of spreading.

As a consequence, we define a **spreading strategy** as a triplet of:

- spreading factor $\in \mathbb{N}$
- spreading period $\in \mathbb{N}$
- algorithm to choose recipients - we define three options:
  1. choose recipients with uniform probability
  2. choose first the peers with the biggest service trust
  3. choose recipients with probability that exponentially grows with the service trust of recipients

Finally, after we evaluate the spreading strategies, we should be able to choose an optimal strategy for following types of messages:

- **Alerts** (viz Section 4.7): Alert messages should be spread rather quickly to their recipients. The reason is that the event that has caused the alert is usually relevant to the given moment and can help other peers to prepare for an incident.
- **File Metadata** (viz Section 4.8): Each metadata message has either *NORMAL* or *CRITICAL* severity. Each severity level should guarantee different spreading properties. Metadata messages with *CRITICAL* severity should spread faster to their recipients even for the cost of flooding the network with messages. On the other hand, metadata messages with *NORMAL* severity do not have to be spread as fast, and thus the peers should choose a spreading strategy that minimises the magnitude of flooding the network with messages.

5.2 Assumptions

Our experiment makes some important assumptions. First, we make assumptions about the Fides trust model. We assume that, in the long run, the average *service trust* of all *malicious peers* is **smaller** than the average *service trust* of all *benign peers*. This assumption essentially means that Fides behaves correctly.

Also, we think it is rational to assume that Fides does not provide to peers estimates of *service trust* values that 100% match the ground-truth values. We assume that an error of the estimation has a mean $\mu = 0$ and a standard deviation $\sigma = 0.25$.

Other assumptions are about a network itself. We assume that the churn rate of the network is practically zero. It means we assume that the network is static - no

peers join or leave the network. Real networks probably never have churn rate zero, but we think that our network will not be too dynamic. The reason is that once a peer joins, it is in his best interest to stay connected for the benefits of a collective defence.

Lastly, we assume that the underlying graph of our network is connected. However, the connectivity of the underlying graph is difficult to verify in the real world because we designed our network as an almost unstructured (DHT k-buckets form a small structure) and pure P2P network.

5.3 Methodology

In this section, we describe exactly how we plan to conduct and evaluate the experiment. Firstly, we describe how we generate the networks. Then we elaborate on how exactly our spreading mechanism works. Lastly, we define based on which values we compare the spreading strategies.

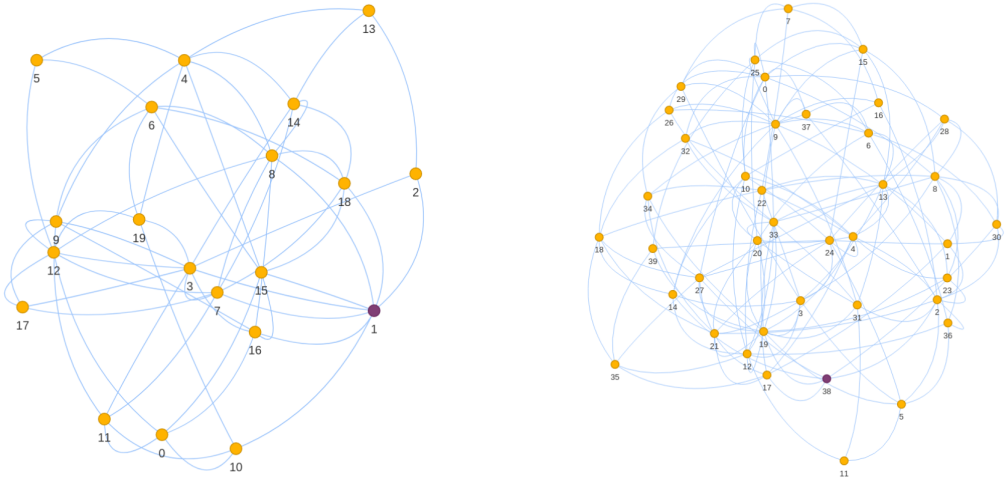
Note that we might use terms *graph* and *network* interchangeably.

5.3.1 Generating Networks

When peers want to send a message to all other peers in the network, the total number of peers in the network is unknown. That is why in our experiment, we generate graphs with number of nodes uniformly sampled from a range between 2-200. We have chosen 200 as an upper bound because we think that is a rational guess for the number of peers in our network in its initial stage of adoption. After that, for every peer, we sample edges representing random outbound connections. We sample a degree for every node's outbound connections from a Poisson distribution with $\lambda = 7$. Furthermore, we randomly choose one node that acts as a peer that starts spreading a gossip. This algorithm is described in Algorithm 2. Some example networks generated for the experiment with just described algorithm can be seen in Figure 5.1.

After generating a network, we have to select the *service trust* value for every node. In the experiment, we consider 2 types of peers - malicious and benign. For each graph, we test **4 scenarios** which are defined by ratio of malicious peers in the network. Possible ratios are 0%, 25%, 50% and 75%.

However, the behaviour of malicious and benign peers is the same. We do not try to model any specific attack scenario. The maliciousness of malicious peers is modelled by in average smaller *service trust* because we assume that Fides assigned them smaller *service trust* based on their past malicious behaviour. As stated in assumptions, we assume average *service trust* of malicious peers is smaller than average *service trust* of benign peers. We distinguish between malicious and benign



(a) A network with 20 peers

(b) A network with 40 peers

Figure 5.1: **An example of two networks generated for the experiment on optimal epidemic spreading strategies.** Purple nodes represent a randomly chosen starting nodes for spreading a message.

peers because we would like to see if specific spreading strategies favour only benign or malicious peers. This is explained further in the subsection that describes our evaluation technique.

In each scenario, we randomly choose the mean μ_m of service trust of all malicious peers. Then, we sample ground-truth service trusts for all malicious peers from normal distribution $sr_n = \mathcal{N}(\mu_m, 0.15)$. The same process is repeated for benign peers. We chose a standard deviation of 0.15 to allow situations where malicious peers have higher service trust than some benign peers because we think it is rational to expect such inaccuracy in our trust model.

Finally, we choose how every peer views its connected peers in terms of service trust. We realise Fides will most probably estimate service trust of other peers with some error from ground-truth value. That is why, for every scenario, we randomly choose standard deviation σ_{TM} from the set $\{0, 0.05, 0.15, 0.25\}$ that represents the error produced by the underlying trust model. Then, for every edge (n_1, n_2), we sample a service trust view of n_1 about n_2 from $\mathcal{N}(sr_2, \sigma_{TM})$ and vice-versa. This process is described in Algorithm 3.

5.3.2 Gossip Spreading

After describing how the networks were generated we only miss the algorithm of how to simulate a spreading strategy in a network.

Algorithm 2: Generate raw graphs

```
1  Function generate():
2      nodes ← generate random number of nodes between 2-200
3      edges ← empty set
4      for n_1 ∈ nodes do
5          degree ← sample from Poisson(7)
6          N_2 ← degree number of random nodes from nodes − {n_1}
7          for n_2 ∈ N_2 do
8              edges ← add (n_1, n_2)
9          end
10     end
11     start ← random node from nodes
12     return (nodes, edges, start)
13
```

A *spreading period* is defined using time. That is why we need to find a way of simulating time. We do that by running our simulation in **ticks**. One tick represents a time period in which peers can transfer one gossip to peers that they share an edge with. However, after we find an optimal spreading strategy, we need to convert the ticks back to time by measuring an average duration of one message transfer in the real network.

In the simulation, we treat the ground-truth *service trust* of every peer as a probability of successfully sending a message to its recipient. For example, if a peer p has a ground-truth *service trust* of 0.1, it successfully shares only 1 out of 10 messages even despite the fact that any other peer x might view the service trust of p as much higher or lower.

Every peer keeps a list of candidates for sharing the gossip. In the beginning, the list of candidates for every peer p_1 contains peer p_2 for every edge (p_1, p_2). Whenever a peer receives the gossip from a sender s, it removes s from the list of candidates because peer s already knows about the gossip. Also, whenever the peer successfully shares the gossip with a recipient r, it removes recipient r from the list of candidates because peer r already knows about the gossip.

Peers can only choose recipients of gossips from their list of candidates. If the list is empty, the peer stops spreading the gossip. An algorithm of how to choose recipients from the list of candidates is determined by currently tested spreading strategy.

In our simulation, we simulate spreading of only one message in the network. All peers begin in *Susceptible* state (see definitions of states in Background Section 2.4.4)

Algorithm 3: Generating graphs with service trust

```
Function generate(k):
    G ← generate k graphs  /* viz Algorithm 2 */
    N ← empty list
    for g ∈ G do
        /* for every possible malicious ratio */
        for mr ∈ {0%, 25%, 50%, 75%} do
            /* set ground-truth service trust to malicious peers */
            μ_m ← random from {0, 0.25, 0.5, 0.75}
            assign service trust from 𝒩(μ_m, 0.15) to mr nodes in g

            /* set ground-truth service trust to benign peers */
            μ_b ← random from {0.25, 0.5, 0.75, 1} but higher than μ_m
            assign service trust from 𝒩(μ_b, 0.15) to (1-mr) nodes in g

            /* set trust models' views of service trusts */
            σ_TM ← random from {0, 0.05, 0.15, 0.25}
            for n_1 ∈ g.nodes do
                for n_2 ∈ n_1.edges do
                    st_real ← ground-truth service trust of n_2
                    st_view ← sample from 𝒩(st_real, σ_TM)
                    set st_view as a trust view of n_1 about n_2
                end
            end
            N ← add g
        end
    end
    return N
```

except the starting peer that starts in *Infected* state. Peers move to *Infected* state after they receive the gossip. From the *Infected* state, peers can move to *Removed* state when their list of candidates is empty. The simulation ends when no peer is in *Susceptible* state or the maximum number of allowed ticks has elapsed.

5.3.3 Evaluation Technique

We conduct simulations of spreading strategies in two different environments - non-malicious and malicious. In both environments, we want to see which spreading strategies were successful in all networks. We consider a strategy successful in a network, if it manages to spread the gossip to all benign peers before the maximum allowed ticks elapse. If no strategy in a given environment succeeds in all networks, we focus on finding strategies that were successful in most networks.

Further, we essentially search for metrics that would show us how fast the most successful strategies manage to spread the gossip and how much they flood the networks with messages. Also, in the malicious environment, we measure these metrics both for the whole network and only for benign peers to see if some strategy produces strictly better results only for benign peers or vice-versa.

The speed of spreading the message is determined by a tick when all peers have already heard about the message. The intensity of flooding can be calculated by a number of repeated messages in the network. We consider a message repeated if it has been sent to a peer in either Infected or Removed state (in other words every peer that has already received the gossip before).

However, absolute number of repeated messages is not sufficient because it also greatly depends on the end tick of the simulation. Imagine two strategies s_1 and s_2. A total number of repeated messages both for s_1 and s_2 is 10 and 20, respectively. Strategy s_1 seems outperforming s_2. However, if strategy s_1 ended during 5th tick, it in an average produced 2 repeated messages per tick. If strategy s_2 ended during 100th tick, it produced in average 0.2 repeated messages. Thus in a long run, s_1 could flood the network significantly more compared to s_2. That is why we measure the average number of repeated messages per tick.

Furthermore, we record the ratio of repeated messages to all sent messages in the networks to see if some strategies produce smaller ratio of repeated messages.

To summarise, for the most successful strategies, we measure:

- **Repeated messages per tick** - An average number of repeated messages per one tick.
- **Ratio of repeated messages** - A ratio of repeated messages to all successfully sent messages in the system.
- **Average end tick** - An average end tick of finished spreading.

- **Worst case end tick** - A worst case end tick of finished spreading. This value should help us decide a *spreading expiration* value for the final strategy because it tells us how long it can eventually take until all peers receive the message.

5.4 Results

We present results of the experiment conducted with following values. We have generated in total 210 testing spreading strategies as permutations of:

- spreading factor - $\{1, 2, 3, 5, 7, 9, ALL\}$. ALL stands for all possible recipients in a candidate list.
- spreading period - $\{1, 2, 3, 5, 10, 20, 50, 100, 250, 500\}$. Each value represents $TICKS$ in a simulation.
- 3 algorithms for choosing recipients (described in Subsection 5.1)

All spreading strategies were simulated in totally 2,000 different networks which results in totally 420,000 simulations. We have chosen maximum allowed ticks in every simulation 10,000.

Note that for clarity, we show results with ticks converted back to time. We have measured that one transfer of a message using Iris P2P network takes approximately 300ms. The measurement was conducted between one peer deployed in CTU network while the other one was deployed in a server in Japan. Because of that, we have decided to convert 1 tick to 500ms to take into account also low-bandwidth and low CPU devices.

Firstly, we show results of spreading strategies simulating in networks without malicious peers. After that, we show results from environment with simulated malicious peers.

5.4.1 No Malicious Peers in Networks

In this scenario, all peers are considered benign. This still means that some messages can get lost because peers have different ground-truth value of *service trust* that represents the probability of successfully sending a message. Out of totally 210 strategies, 200 strategies have successfully spread the in all networks before maximum ticks elapsed. Across all successful strategies, an average ratio of repeated messages is $\mu = 0.74$ with standard deviation $\sigma = 0.028$ and maximum and minimum values 0.68, 0.78, respectively.

As we cannot clearly present results for all the 200 successful strategies, we filter only the most interested ones. In Table 5.1, we can see the five best and the five worst successful spreading strategies in terms of repeated messages per tick. For these 5 best

strategies we can see a total total number of sent messages in Figure 5.2. In contrast, in Table 5.2 we see the 5 best strategies in terms of speed of convergence. Value **A** in *choosing recipients* field in spreading strategies stands for choosing recipients with uniform probability. **B** represents choosing recipients based on their service trust in descending order. **C** stands for choosing recipients with a probability that grows exponentially with a view of recipient's service trust.

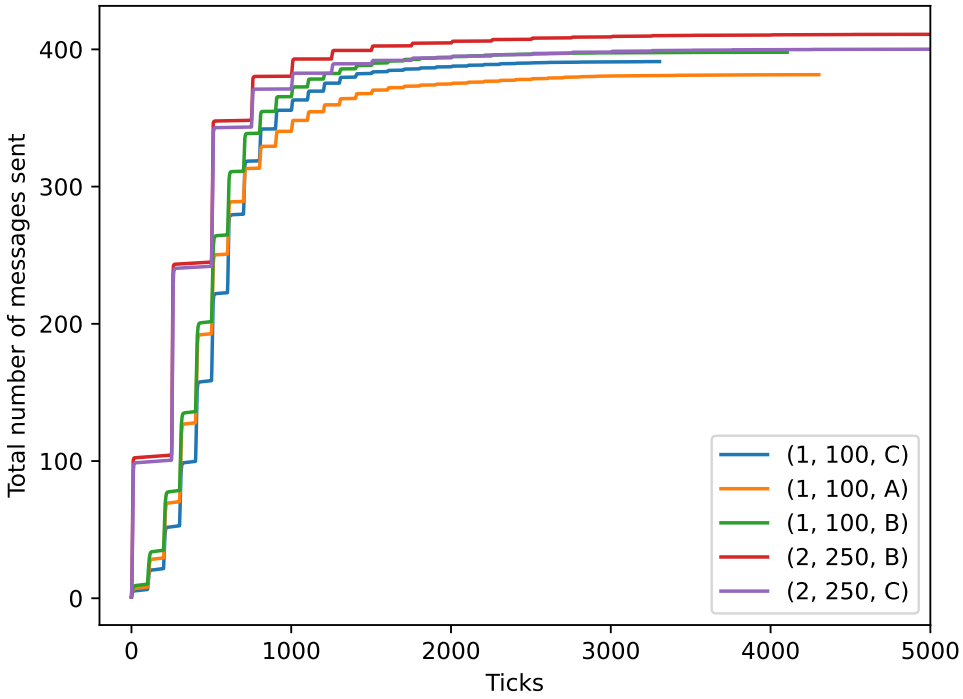


Figure 5.2: **A total number of messages sent for 5 spreading strategies with the lowest number of *repeated messages per tick* in an environment without malicious peers.** The figures' legends depict values of spreading strategies. The first one represents *spreading factor*. The second one represents *spreading period*. Lastly, third value represents an algorithm for choosing recipients. The value **A** stands for choosing recipients with uniform probability. **B** represents choosing recipients based on their service trust in descending order. **C** stands for choosing recipients with a probability that grows exponentially with a view of recipient's service trust.

spreading strategy			repeated messages per tick	ratio of repeated messages	average duration of spreading	worst case duration of spreading
factor	**period**	**choosing recipients**				
1	**100**	**C**	0.33	0.69	7.1min	45.86min
1	**100**	**A**	0.35	0.68	6.75min	29.21min
1	**100**	**B**	0.35	0.70	7.05min	45.08min
2	**250**	**B**	0.43	0.71	6.96min	41.73min
2	**250**	**C**	0.44	0.70	6.68min	56.32min
			...			
9	**1**	**A**	106.66	0.78	1.93s	6.5s
9	**1**	**B**	106.97	0.78	2.035s	11.0s
ALL	**1**	**B**	112.72	0.78	1.93s	10.5s
ALL	**1**	**C**	112.84	0.78	1.89s	9.0s
ALL	**1**	**A**	112.84	0.78	1.86s	9.5s

Table 5.1: **The five best and the five worst spreading strategies in terms of *repeated messages per tick* in an environment without malicious peers.** Value **A** in *choosing recipients* column stands for choosing recipients with uniform probability. **B** represents choosing recipients based on their service trust in descending order. **C** stands for choosing recipients with a probability that grows exponentially with a view of recipient's service trust. Value **ALL** in spreading strategy factor column stands for spreading to all available recipients in a candidate list.

5.4.2 Different Ratios of Malicious Peers in Networks

We have simulated spreading strategies in networks with 25%, 50% and 75% of malicious peers. In all three environments, no spreading strategy successfully spread the message in all networks. However, in all environments, the most successful spreading strategies were successful in 99% of all networks. We present the best strategies from these most successful strategies.

In Table 5.3, we can see an average value of *ratio of repeated messages sent to benign peers* along with standard deviation, minimum and maximum values across the most successful strategies.

In Table 5.4, we can see for each malicious ratio the best spreading strategies in terms of *repeated messages sent to benign peers per tick*. For these strategies, Figure 5.3 shows a total number of messages sent to benign peers. In Table 5.5, we can see for each malicious ratio the best spreading strategies in terms of *average end tick of spread to benign peers*. Value **A** in *choosing recipients* field in spreading strategies

stands for choosing recipients with uniform probability. **B** represents choosing recipients based on their service trust in descending order. **C** stands for choosing recipients with a probability that grows exponentially with a view of recipient's service trust.

5.5 Discussion

Regardless the number of malicious peers in the networks, spreading strategies with the fastest convergence of spreading are the ones with the biggest possible *spreading factor* and the smallest possible *spreading period* (as we can see in Tables 5.5 and 5.2 that show both malicious and non-malicious environments). Such result is logical because in these strategies peers simply send the message to all possible candidates as soon as possible. On the other hand, the strategies that produce the smallest *number of repeated messages per tick* are the slowest strategies - the ones with the smallest *spreading factor* and the biggest *spreading period* (as we can see in Tables 5.1 and 5.4). This makes also sense because bigger spreading period makes the simulation to take more ticks and that is why a value of *repeated messages per tick* decreases.

A significant discovery of this experiment is that we can decrease the *ratio of repeated messages* sent in the overall system and more importantly sent to benign peers by spreading the message slowly. We see that one of the the best strategies in terms of *repeated messages per tick* in the non-malicious environment produced a ratio of repeated messages 0.68 (viz Table 5.1). On the other hand, the best strategy in terms of speed of convergence produced a ratio of repeated messages 0.78 (viz Table 5.2). We observe a 10% difference. This fact holds strongly even in the environments with malicious peers. By comparing the same values in Tables 5.4 and 5.5, we see that in the environment with 25% of malicious peers, we can reduce the ratio of repeated messages sent to benign peers from 0.76 to 0.66 by spreading slowly. In the environment with 50% of malicious peers, we can reduce the ratio of repeated messages sent to benign peers from 0.74 to 0.63 and in the environment with 75% malicious peers from 0.72 to 0.60. The improvement is always at least 10%. This means that no matter the amount of malicious peers in the system, we can decrease a total number of sent repeated messages to benign peers by at-least 10% with spreading the message slowly. The reason is that by spreading the message slowly, we can avoid situations when peers send messages to each other simultaneously without a knowledge that the other peer already knows about the message.

If we look at the best spreading strategies in terms of *repeated messages per tick sent to benign peers* (Tables 5.1 and 5.4), we see that the top 3 strategies differ only in algorithms for choosing recipients (except for the environment with 75% of malicious peers) - strategies **(1,100,-)**. It means that in all environments except the one with 75% of malicious peers, algorithms for choosing recipients have no substantial effect in moderating the level of flooding the networks with messages. The change occurs

in the environment with 75% malicious peers where the best strategies in terms of *repeated messages per tick sent to benign peers* choose recipients with a probability that exponentially grows with candidates' service trust. It means that if we assume a highly adversarial network, we can actually benefit from using service trust as a heuristic to choose message recipients.

Moreover, service trust value has no effect if we want to share the message as fast as possible because all fastest strategies simply try to send the message to all available peers in the list of candidates. That is why the algorithm for choosing recipients is actually redundant.

In Table 5.3, we can see the distribution of *ratio of repeated messages sent to benign peers* across all strategies in malicious environments. We can see, that mean values decrease disproportionately with ratio of malicious peers in the network. A wrong interpretation would be to say that the more malicious peers the better. That is not correct. In Table 5.5, we can see that worst case duration of spreading grows proportionally with maliciousness of the network. The reason is that the networks become so untrustworthy that almost no message is successfully sent.

As we already stated, the fastest spreading strategies are the ones that send the message to all possible peers as fast as possible and the algorithm that chooses recipients is redundant. However, to decide the best strategies while optimising the *repeated messages per tick*, we have plotted the best strategies from Tables 5.1 and 5.4 in Figures 5.2 and 5.3. The figures show the total number of messages sent to benign peers over time - that is why the lower the y axis, the better. From the figures we conclude that strategy (1, 100, A) has the best performance in environments from 0% to 50% of malicious peers. In the environment with 75% of malicious peers, strategy (1, 50, C) has the best performance. If we use again the conversion that 1 tick equals 500ms, we get the best strategy **(1, 50s, A)** for non-to-medium malicious networks and **(1, 25s, C)** for highly adversarial networks. Let us remind that algorithm **A** stands for choosing recipients with uniform probability, while **C** represents choosing recipients with a probability that exponentially grows with candidates' service trust.

Another outcome of the experiment is that we can estimate how long it can eventually take to spread the message to all benign peers using certain strategies in a given environment. For the estimation we can utilise the *worst case duration* column in the result Tables. For example, in Tables 5.2 and 5.5 we can see that the fastest spreading strategies may take **from 10s** in non-malicious environment **up-to 40s** in the most malicious environment. On the other hand, if we focus on optimising *the number of sent repeated messages to benign peers per tick* by spreading slowly, we can see in Tables 5.1 and 5.4 that the best strategies may take **from 45min** in non-malicious environment **up-to 80min** in the most malicious environment.

It is also important to discuss if the simulated networks behave the same way as the real networks would behave. We have assumed that the simulated networks

are static - edges in the under-laying graph do not change. In reality, peer-to-peer networks can never guarantee such property. On the other hand, in dynamic networks, the number of nodes and edges may change. To design the best spreading strategies in such environment, we would have to be able to measure an up-time of peers to choose recipients that have a bigger probability of not leaving the network after we share the gossip with them.

Another important note is that in the experiment, we have not considered a malicious actor that would try to exploit the epidemic protocol by disseminating a large number of messages. Such action could result in flooding the network with substantial number of messages and potentially leading to DDoS of the entire network. A possible mitigation for such attack is to employ some form of rate-limiting per individual peers. Every peer would keep track of number of sent messages by all its neighbours (or number of bytes) and allow only a certain amount of them for each neighbour per time unit. Researching this idea is out of scope of this thesis and left as a future work.

Lastly, we could have designed values that define spreading strategies differently. For example, *spreading factor* and *spreading period* could be defined as ranges from which every peer randomly chooses values which define its spreading strategy. However, the number of possible combinations is large and that is why we have not chosen this approach in our experiment. Another different option how to choose a value of *spreading factor* is by for example setting *spreading factor* of every peer to a specific percentage of its all connections.

spreading strategy			repeated messages per tick	ratio of repeated messages	average duration of spreading	worst case duration of spreading
factor	**period**	**choosing recipients**				
ALL	**1**	**A**	112.84	0.78	1.86s	9.5s
ALL	**1**	**C**	112.84	0.78	1.89s	9.0s
ALL	**1**	**B**	112.72	0.78	1.93s	10.5s
9	**1**	**A**	106.66	0.78	1.93s	6.5s
9	**1**	**C**	106.25	0.78	1.98s	5.5s

(a) Best strategies based on *average duration of spreading* column

spreading strategy			repeated messages per tick	ratio of repeated messages	average duration of spreading	worst care duration of spreading
factor	**period**	**choosing recipients**				
9	**1**	**C**	106.25	0.78	1.98s	5.5s
9	**1**	**A**	106.66	0.78	1.93s	6.5s
7	**1**	**A**	95.50	0.77	2.085s	7.5s
5	**1**	**C**	79.90	0.77	2.445s	8.5s
ALL	**1**	**C**	112.84	0.78	1.89s	9.0s

(b) Best strategies based on *worst case duration of spreading* column

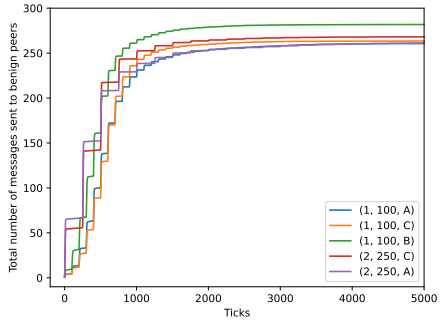
Table 5.2: **The five best spreading strategies in terms of speed of spreading the message to all peers in the network in an environment without malicious peers.** Value **A** in *choosing recipients* column stands for choosing recipients with uniform probability. **B** represents choosing recipients based on their service trust in descending order. **C** stands for choosing recipients with a probability that grows exponentially with a view of recipient's service trust. Value **ALL** in spreading strategy factor column stands for spreading to all available recipients in a candidate list.

ratio of malicious peers	**ratio of repeated messages sent to benign peers across strategies**			
	μ	σ	min	max
25%	0.73	0.03	0.66	0.77
50%	0.70	0.03	0.63	0.76
75%	0.65	0.04	0.56	0.72

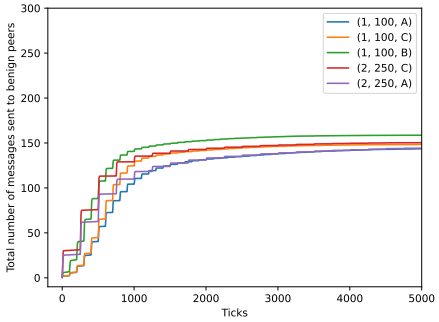
Table 5.3: **Distribution of *ratio of repeated messages sent to benign peers* across spreading strategies in malicious environments.** The table covers only strategies that were successful in 99% of networks (i.e. most successful strategies)

ratio of malicious peers	**spreading strategy**			**repeated messages per tick**		**ratio of repeated messages**		**average duration of spreading**		**worst case duration of spreading**	
	factor	**period**	**choosing recipients**	all peers	**benign peers**	all peers	**benign peers**	all peers	**benign peers**	all peers	**benign peers**
25%	**1**	**100**	**A**	0.30	0.21	0.68	0.66	7.92min	7.64min	> 85min	49.23min
	1	**100**	**C**	0.30	0.21	0.69	0.68	7.85min	7.21min	> 85min	39.18min
	1	**100**	**B**	0.33	0.25	0.70	0.70	7.47min	6.25min	> 85min	50.85min
	2	**250**	**C**	0.38	0.27	0.70	0.68	7.38min	6.62min	> 85min	56.27min
	2	**250**	**A**	0.41	0.29	0.69	0.67	7.71min	7.33min	> 85min	77.1min
50%	**1**	**100**	**A**	0.23	0.10	0.68	0.63	11.09min	10.01min	> 85min	77.57min
	1	**100**	**C**	0.24	0.11	0.68	0.65	10.28min	7.97min	> 85min	80.07min
	1	**100**	**B**	0.26	0.13	0.69	0.68	9.5min	6.29min	> 85min	56.75min
	2	**250**	**C**	0.30	0.13	0.69	0.65	9.85min	7.41min	> 85min	66.72min
	2	**250**	**A**	0.30	0.13	0.69	0.64	11.07min	9.85min	> 85min	70.88min
75%	**1**	**50**	**C**	0.36	0.08	0.68	0.60	9.0min	5.13min	> 85min	59.62min
	2	**100**	**C**	0.51	0.11	0.68	0.60	7.99min	4.11min	> 85min	77.53min
	3	**250**	**C**	0.46	0.11	0.69	0.62	11.37min	5.43min	> 85min	77.1min
	2	**100**	**B**	0.49	0.13	0.69	0.64	8.11min	3.34min	> 85min	57.52min
	1	**20**	**A**	0.85	0.17	0.68	0.56	5.1min	3.55min	> 85min	79.55min

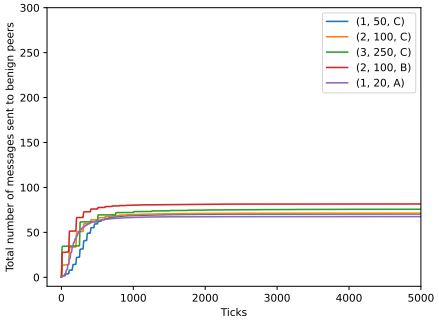
Table 5.4: **The best spreading strategies in terms of *repeated messages sent to benign peers per tick* in a malicious environment.** All depicted strategies were successful in spreading the message to all benign peers in the network in 99% of cases of all networks. Value **A** in *choosing recipients* column stands for choosing recipients with uniform probability. **B** represents choosing recipients based on their service trust in descending order. **C** stands for choosing recipients with a probability that grows exponentially with a view of recipient's service trust.



(a) ratio of malicious peers 25%



(b) ratio of malicious peers 50%



(c) ratio of malicious peers 75%

Figure 5.3: **A total number of messages sent to benign peers for the best spreading strategies with the lowest number of *repeated messages sent to benign peers per tick* in malicious environments.** All depicted strategies were successful in spreading the message to all benign peers in the network in 99% of cases of all networks. The figures' legends depict values of spreading strategies. The first one represents *spreading factor*. The second one represents *spreading period*. Lastly, third value represents an algorithm for choosing recipients. The value **A** stands for choosing recipients with uniform probability. **B** represents choosing recipients based on their service trust in descending order. **C** stands for choosing recipients with a probability that grows exponentially with a view of recipient's service trust.

ratio of malicious peers	spreading strategy			repeated messages per tick		ratio of repeated messages		average duration of spreading		worst case duration of spreading	
	factor	period	choosing recipients	all peers	**benign peers**	all peers	**benign peers**	all peers	**benign peers**	all peers	**benign peers**
25%	**9**	**1**	**B**	102.74	79.02	0.77	0.77	2.025s	1.845s	> 85min	8.0s
	9	**1**	**C**	103.81	76.19	0.77	0.76	1.95s	1.86s	> 85min	5.0s
	ALL	**2**	**B**	97.77	75.16	0.77	0.77	2.105s	1.925s	> 85min	7.5s
	ALL	**1**	**A**	102.40	74.36	0.77	0.76	1.995s	1.95s	> 85min	13.0s
	ALL	**1**	**C**	96.58	70.77	0.77	0.76	2.075s	1.98s	> 85min	6.5s
50%	**9**	**1**	**B**	79.74	41.06	0.76	0.76	2.635s	2.01s	> 85min	9.5s
	9	**2**	**B**	84.05	43.46	0.76	0.76	2.71s	2.02s	> 85min	28.0s
	9	**2**	**C**	84.29	40.27	0.75	0.74	2.495s	2.055s	> 85min	33.0s
	ALL	**2**	**B**	66.13	35.33	0.75	0.75	3.555s	2.195s	> 85min	15.5s
	ALL	**2**	**B**	72.22	37.11	0.76	0.76	2.8s	2.205s	> 85min	23.0s
75%	**ALL**	**1**	**B**	65.47	17.48	0.74	0.72	5.555s	2.17s	> 85min	44.0s
	9	**1**	**C**	64.62	15.57	0.74	0.70	5.6s	2.27s	> 85min	24.5s
	ALL	**2**	**B**	61.19	16.22	0.74	0.72	6.205s	2.285s	> 85min	36.0s
	9	**1**	**C**	61.33	14.67	0.74	0.70	5.205s	2.285s	> 85min	23.0s
	7	**1**	**B**	55.97	14.72	0.74	0.72	5.825s	2.365s	> 85min	26.0s

Table 5.5: **The best spreading strategies in terms of *average duration of spreading to benign peers* in malicious environments.** All depicted strategies were successful in spreading the message to all benign peers in the network in 99% of cases of all networks. Value **A** in *choosing recipients* column stands for choosing recipients with uniform probability. **B** represents choosing recipients based on their service trust in descending order. **C** stands for choosing recipients with a probability that grows exponentially with a view of recipient's service trust. Value **ALL** in spreading strategy factor column stands for spreading to all available recipients in a candidate list.

Chapter 6

Implementation

A significant part of this thesis consists of an implementation (> 5000 lines of code) of a working prototype of Iris which is intended to be integrated into Slips [19]. The reference implementation is written in Go using the LibP2P project and can be found online in Github [41].

In this chapter, firstly, we briefly describe generating organisations and running an Iris instance. Secondly, we show diagrams that describe integration of Slips, Fides and Iris. To access the full documentation of Iris, we refer the reader to the Github repository [41] that offers a detailed technical description of the entire project.

6.1 User Guide

6.1.1 OrgSig Tool

We have developed a small tool called *OrgSig*. OrgSig offers two features (see help message displayed in Listing 6.1) that help operating organisations:

- **Generate a new organisation** - Internally this means generate a new asymmetric key-pair, save corresponding private key to a file and output ID (public-key) of the new organisation.
- **Sign a peer using an Organisation** - Internally this means to sign a peer ID using the organisation's private-key.

Listing 6.1: Compilation and help output of Orgsig tool

```
> make orgsig
go build cmd/orgsig.go
>  ./orgsig --help
Running v0.0.1 orgsig
```

```
Usage of ./orgsig:
  -load-key-path string
        Path to a file with organisation private key. If not
           set, new private-key is generated.
  -peer-id string
        Public ID of a peer to sign. Flag --sign-peer must be
            set for this option to be valid.
  -save-key-path string
        If set, value will be used as a path to save
           organisation private-key.
  -sign-peer
        Flag to sign peer ID. Flag peer-id can be used to set
            peerID, otherwise, cli will ask. The signature
           will be printed to stdout.
```

6.1.2 Running an Iris Instance

To run a peer, a user needs a running Redis instance that serves as a communication channel between Redis and Fides (see the communication diagram between components in Figure 4.1). Also, to compile the code, Go ($\geq$ 1.17) is needed.

Every peer requires a configuration file given in *yaml* format. See a small example with most important parameters depicted in Listing 1. The example configuration specifies to generate new peer ID and start listening for incoming connections at an address 127.0.0.1:9000. It also specifies a Redis instance running at localhost with default port and Redis communication channel to exchange messages with *Fides*. Further, the configuration defines one trusted organisation. Lastly, configuration lists an address of one running peer and disables default bootstrapping nodes. To see a list of all configuration options, we refer reader to the project repository [41]. With the mentioned configuration saved in the *config.yaml* file, a peer can be simply started as:

```
> go run cmd/peercli.go --conf config.yaml
```

6.2 Flow of Events Between Slips, Fides and Iris P2P System

As we have described in Section 4.3 of the Design Chapter, there are in total three components: **Slips**, **Fides** and **Iris**. Slips is a brain of the system and uses Iris

```yaml
Identity:
  GenerateNewKey: true

Server:
  port: 9000
  Host: 127.0.0.1

Redis:
  Host: 127.0.0.1
  Tl2NlChannel: gp2p_tl2nl

Organisations:
  Trustworthy:
    - "12D3KooWErR8ZLhjAWYw4oj7gWLRPp99aupNU5HbFfVN9U12NBFZ"

PeerDiscovery:
  ListOfMultiAddresses:
    - "/ip4/127.0.0.1/udp/9001/quic 12D3KooWNxiCsZFyUFpLFNKDLEQDUK36my
       ifqufnnveK1jycMoJ8"
  DisableBootstrappingNodes: true
```

Listing 1: **An example of a *YAML* configuration file for a peer.** The configuration starts a peer that generates a new new random ID and listens for incoming connections at an address *127.0.0.1:9000*. The newly created peer trusts peers from a single organisation and connects directly to a peer at address *127.0.0.1:9001*. Also, the peer tries to connect to a Redis instance running on *127.0.0.1* with default port and channel *gp2p_tl2nl* to exchange messages with Fides.

as a tool to talk with other Slips instances. Fides is a trust model implemented in Python and integrated into Slips as a built-in module. On the other hand, Iris is implemented as a separate code-base in Go. A reason for not implementing Iris also in Python is that Go offers much better support for networking (for example built-in goroutines for asynchronous communication) and that LibP2P offers most specifications written in Go. That is why only Fides exchanges messages directly with Slips and Iris communicates only with Fides through a Redis channel (a diagram of the communication flow can be seen in Figure 4.1).

In this section, we show in more details the flow of events between these components for all implemented protocols.

6.2.1 File Sharing Protocol

The theory behind File Sharing Protocol is defined in Section 4.8. Figure 6.2 contains diagrams that depict file sharing procedure. Firstly, Figure 6.2a depicts events after Slips decides to share a file with peers in the network. Secondly, Figure 6.2b shows events after a peer receives a metadata file from the network. Lastly, Figure 6.1 depicts how downloading of a file from the network works.

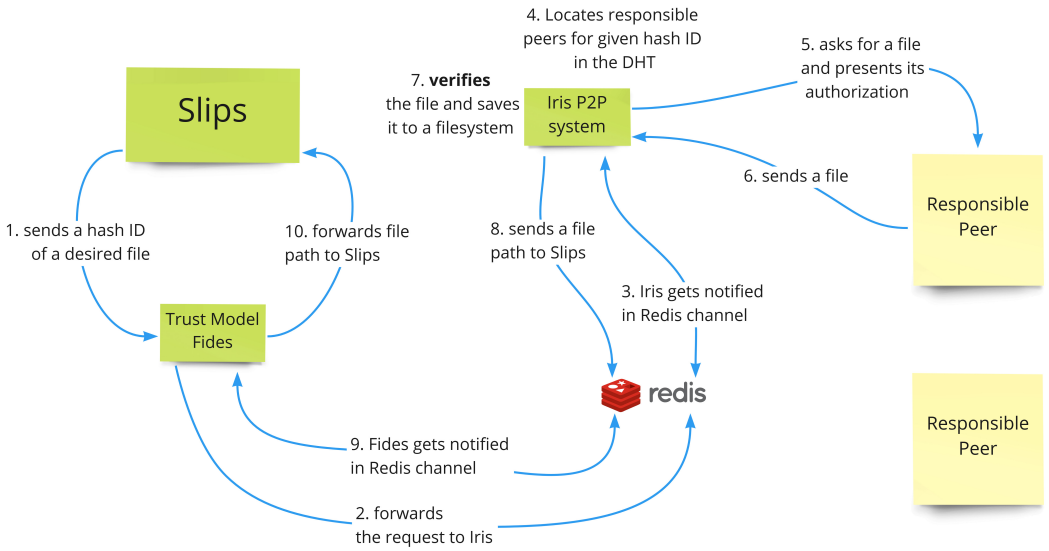


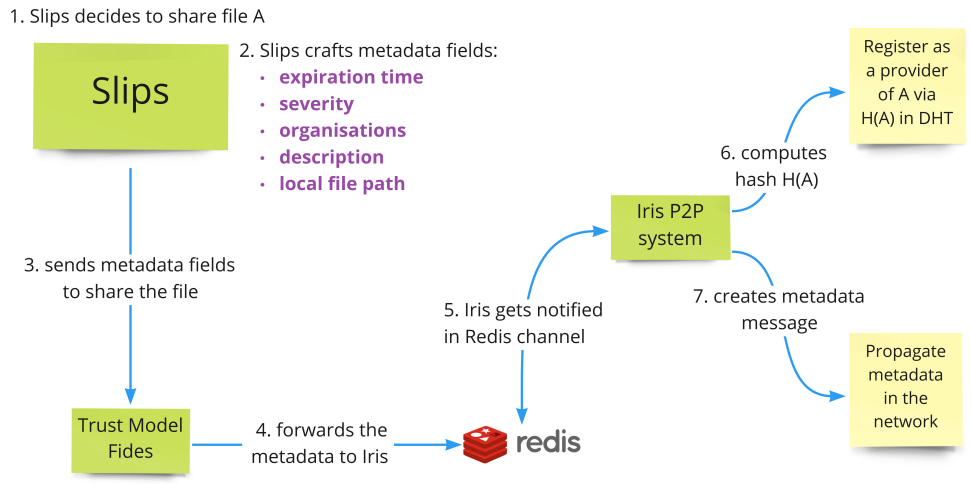
Figure 6.1: Flow of events between components in File Sharing Protocol after Slips decides to download a shared file

6.2.2 Alert Protocol

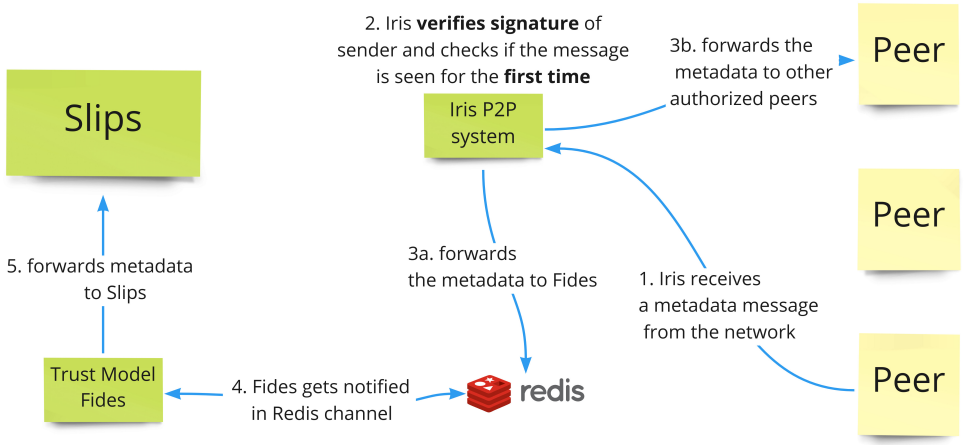
The theory behind Alert Protocol is defined in Section 4.7. In Figure 6.3, we can see diagrams that show how the alerts propagate through the components in the system. Firstly, Figure 6.3a shows events after Slips decides to alert the network. Secondly, Figure 6.3b depicts Iris receiving an alert form the network.

6.2.3 Network Opinion Protocol

The theory behind Network Opinion Protocol is defined in Section 4.9. In Figure 6.4 we can see diagrams that show how Slips ask other Slips instances for an opinion on a specific suspicious IoC. Firstly, Figure 6.4a shows communication events between components after Slips decides to ask other peers for an opinion. Secondly, Figure 6.4b depicts Iris receiving an opinion request from the network.

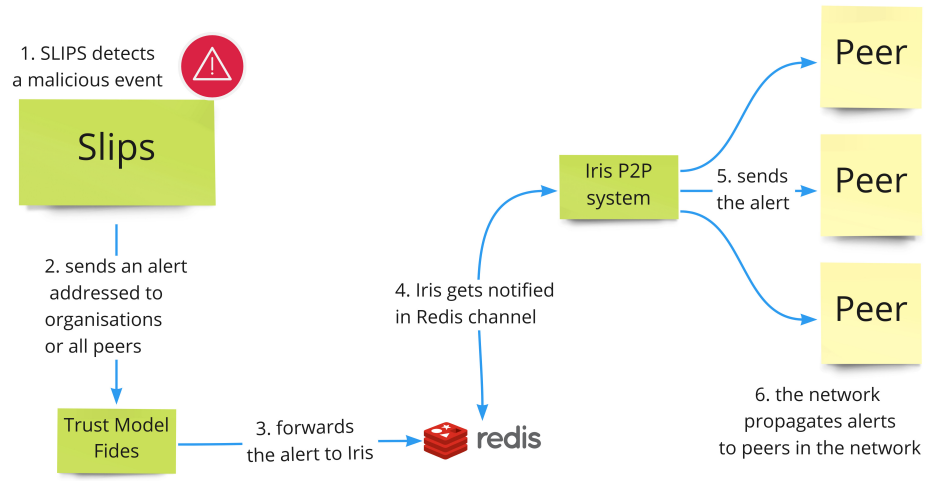


(a) Slips sharing a file

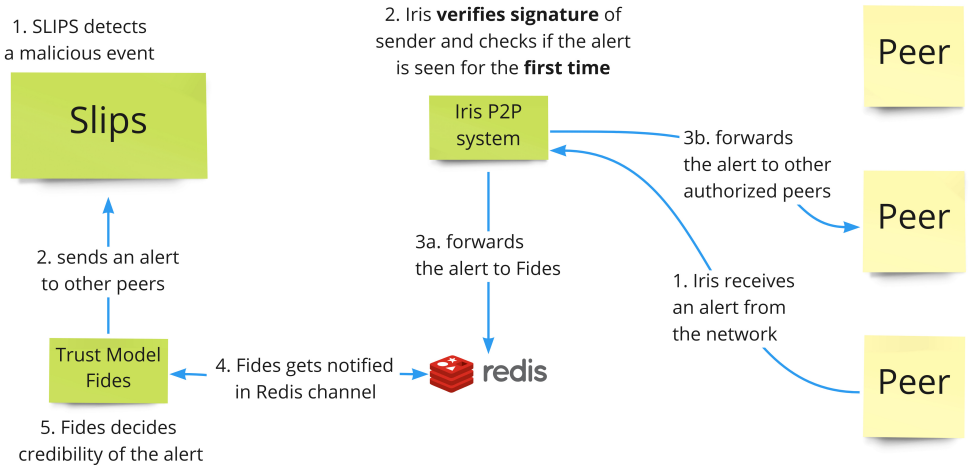


(b) Iris receiving a metadata message from the network

Figure 6.2: Flow of events between components in File Sharing Protocol after Slips decides to share a file

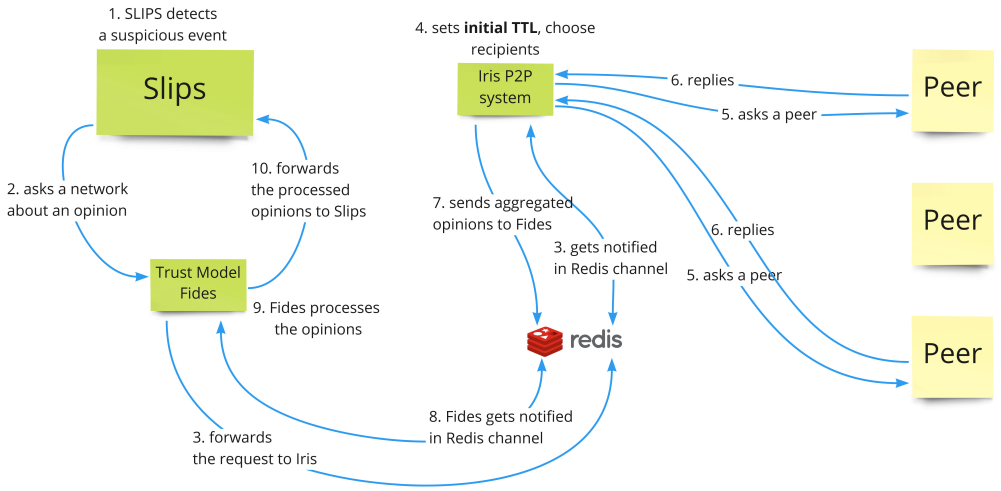


(a) Slips alerting the network

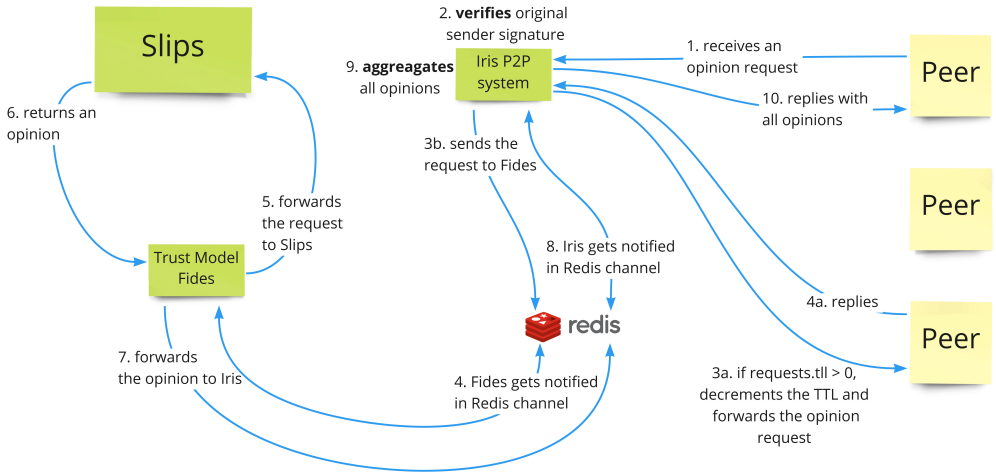


(b) Iris receiving an alert from the network

Figure 6.3: Flow of events between components in Alert Protocol



(a) Slips asking the network for an opinion



(b) Iris receiving an opinion request from the network

Figure 6.4: Flow of events between components in Network Opinion Protocol

Chapter 7

Conclusion

This thesis presented Iris, a global P2P system for collaborative security defence on the Internet with special emphasis on security and privacy. Iris allows peers to share threat intelligence data, alert other peers in the network about detected Indicator of Compromise (IoC), and ask other peers their opinion on a specific IoC. To address the possible confidentiality issues of sharing threat intelligence data with untrusted peers, Iris presents a novel concept called *organisations* that represent trusted groups in the P2P system. Organisations allow peers to address messages only to a specific subset of peers. Further, we have evaluated different spreading strategies based on epidemic protocols to optimise how to spread gossip messages. Last, we have implemented a working prototype of Iris in Go using the LibP2P project and integrated Iris into Slips [19] as a collaborative module.

Organisations are cryptographically-verified groups of peers that help establish connections between trusted peers. Organisations help peers form trusted connections in an otherwise completely adversarial environment which is a P2P network. Peers can be members of an arbitrary number of organisations or create their own organisations. Iris uses Distributed Hash Table (DHT) to store members of all organisations publicly. Our unique design of organisations allows owners of the organisations to have complete control over the organisations' data stored in the DHT. As a result, we mitigate most of the attack vectors that target specific values stored in the DHT.

As far as we know, Iris is the first global security P2P network for sharing threat intelligence that considers the confidentiality of sharing data and the security of the overall system. We face difficulties evaluating our proposed design because, to our knowledge, no similar project or theoretical proposal exists. To some extent, the community could use BitTorrent [9] or IPFS [4] to share threat intelligence in a decentralised manner but these candidates do not offer any built-in form of access control to share confidential data nor are designed to provide information on time. Thus, clients would need to require some additional form of authorisation or provide

the files encrypted, which leads to a problem of distributing keys for decryption. In addition, these non-security P2P networks do not automatically disseminate information about new files to peers in the network. Also, our solution provides more features in terms of collaborative defence than just File Sharing Protocol such as Alerting and Network Opinion Protocol.

The Alert Protocol provides dissemination of information as fast as possible within the P2P network. The idea is that a peer confidently detects a malicious IoC and wants to warn other peers. Iris also allows addressing the alerts just to members of specific organisations to avoid side-channel attacks of attackers eavesdropping on their victims. Also, the content of alerts might be confidential for privacy reasons. On the other hand, the Network Opinion Protocol allows peers to ask other peers about their opinion on a potential IoC. Using the Network Opinion Protocol, peers can ask for threat intelligence about resources that have not yet been labelled as malicious. Using this approach, Iris accelerates the detection of malicious entities because it allows using collective knowledge.

The thesis presented an experiment evaluating different spreading strategies in epidemic protocols to optimise the dissemination of critical and non-critical messages in the Iris P2P system. Results of the experiment show that we can decrease the number of sent repeated messages in the system by at least 10% if we decide to spread the message slowly. This fact holds strongly even in hostile environments. Iris utilises this fact when spreading non-critical metadata messages in our File Sharing Protocol. Also, the experiment shows that Iris can spread critical messages such as alert messages in the P2P network in less than a minute.

A significant part of the thesis is a working implementation of Iris with more than 5000 lines of code. The implementation is written in the Go programming language using the LibP2P project [30]. Besides the implementation itself, the thesis also provided complete integration of Iris into Slips [19] as collaborative module. The entire code-base is open-source and offered to the community [41].

Iris guarantees the confidentiality and integrity of shared data. All communication is digitally signed and *end-to-end* encrypted using keys that at the same time form peers' identities. In this way, an attacker cannot impersonate other peers without knowledge of the private key of the impersonating peer. If the attacker tries that, the digital signatures would not match the impersonating peer's ID. Similarly, the attacker cannot launch a Man-in-the-Middle (MitM) attack if the involved peers already know their respective IDs before the MitM attack happens. Moreover, the confidentiality and integrity properties hold even for the content shared within our File Sharing Protocol.

Even though we have paid special attention to security, we have not researched and mitigated all attack vectors that could potentially lead to disruption of the system's availability. Such attacks include a churn attack, exploiting our usage of Epidemic

Protocols to flood the network with messages or attacking the use of DHT in our File Sharing Protocol. For example, as was shown in [37], the generation of enormous rainbow tables of peer identifiers is relatively simple. Such databases can be later used to query responsible peers for given keys stored in the DHT. By launching this attack, an attacker controls the stored value in the DHT under the given key. Unfortunately, this attack vector comes inherently with the characteristic nature of the DHT, and even IPFS [4] is nowadays vulnerable to this kind of attack. Despite the fact that the mitigation of this attack is unknown, we propose mitigation ideas for further research in the Future Work Section 7.1.

We hope Iris can fill the void in the space of security and privacy-based solutions for decentralised sharing of threat intelligence that automatically connect global endpoints to improve the collaborative security. Since Iris is the first proposed solution of its kind, many directions for further research exist. In the last section of the thesis, we describe our ideas for future work.

7.1 Future Work

The field of peer-to-peer networks is vast. In the following paragraphs, we describe some ideas for further practical development of Iris and theoretical research in secure sharing of threat intelligence for fast and collaborative defence.

Our File Sharing Protocol could greatly benefit in terms of security and performance if the protocol chunks shared files and advertise each chunk individually through the DHT. An advantage of this approach is that peers downloading a file can simultaneously download different chunks from different peers; thus, it improves performance. Additionally, it allows downloading peers to perform *incremental verification*. Incremental verification means that peers can verify the correctness of each chunk, and because of that, they can detect malicious peers providing an incorrect file even before they download the entire content of the file. A question that arises is how to store and share a list of all chunks of one file to peers that want to download the file. IPFS does that using Merkle Trees [3]. Every peer that wants to download a given file receives a Merkle Tree of the given file. The Merkle Tree can be used to locate and verify each chunk and to construct the final file from individual downloaded chunks.

One problem Iris shares with IPFS is that responsible peers in the DHTs do not verify the claims of file providers that they truly own the given file. IPFS does not implement this mechanism for performance reasons because it means that responsible peers would have to download the entire file and verify its correctness. On the other hand, Iris cannot implement this mechanism because responsible peers might not have the authorisation to download the given file. To solve this problem, we propose researching the usage of zero-knowledge proofs. Zero-knowledge proofs are methods

to prove a correctness of a statement without revealing the statement. The idea is that responsible peers could require zero-knowledge proof from potential providers to verify that they truly own the file without actually downloading the entire file. Without the proof, responsible peers would not store the providers in the DHT. If this idea succeeds, we could completely mitigate the sybil attack in both Iris and IPFS.

Iris could increase the cost of sybil peers by making it harder to join the network. This could be achieved by enforcing a computational puzzle before joining the network [2]. Peers would have to spend computational resources to compute a puzzle before joining the network. The puzzle can be linked to a peer ID, and that is why the attacker would have to compute a puzzle for every sybil peer. However, this approach dramatically disadvantages benign peers without too much computational resources, such as for example small IoT devices. A solution could be to enforce the puzzle only for peers that claim to be providers of files in the DHT to at least mitigate the risk of sybil providers in our File Sharing Protocol.

So far, Iris does not support the expiration of peers' memberships in organisations. When an organisation issues a signature for a peer, the signature is valid forever. An idea for future work is to design an expiration mechanism after which peers lose membership and have to ask for a new signature. Similarly as certificates issued by certificate authorities expire.

Iris disseminates information in the network using the epidemic protocols. We think an attacker could exploit this fact and craft plenty of fake messages to flood the network, which could potentially lead to a DDoS attack and disruption of the network. At the moment, Iris does not implement any mitigation technique against this attack vector. One solution could be to research adaptive gossip protocols [6] and rate-limit the number of sent messages (or bytes) per peers.

A practical limitation of the current version of Iris is that peers that join the network later have no way of finding information about recently shared alerts or files. Once the dissemination of information ends, newcomers have no way to learn about these pieces of information. To solve this issue, peers could keep a small database of past messages and provide recent events to newcomers that issue a request for this information.

Lastly, our reference implementation only supports joining the network through bootstrapping nodes or configuring online peers manually. We think that variability in methods of joining the network improves the overall security because an attacker needs to attack all the methods to launch a Bootstrapping Attack. For this reason, we think that Iris could benefit from also employing DNS records to find initial peers. The list of peers returned by each DNS record can be dynamically updated; thus, this approach also contributes to the decentralised nature of the entire P2P network.

Bibliography

[1] Jean-Philippe Aumasson. *Serious Cryptography: A Practical Introduction to Modern Encryption*. USA: No Starch Press, 2017. ISBN: 1593278268.

[2] Ingmar Baumgart and Sergio Mies. "S/Kademlia: A practicable approach towards secure key-based routing". In: vol. 2. Jan. 2008, pp. 1–8. ISBN: 978-1-4244-1889-3. DOI: 10.1109/ICPADS.2007.4447808.

[3] Georg Becker and Ruhr-universität Bochum. *Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis*.

[4] Juan Benet. "IPFS - Content Addressed, Versioned, P2P File System". In: (July 2014).

[5] Fernando Bordignon and Gabriel Tolosa. "Gnutella: Distributed System for Information Storage and Searching Model Description". In: (Aug. 2001).

[6] Mike Burmester, Tri Van Le, and Alec Yasinsac. "Adaptive gossip protocols: Managing security and redundancy in dense ad hoc networks". In: *Ad Hoc Networks* 5.3 (2007), pp. 313–323. ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2005.11.007. URL: https://www.sciencedirect.com/science/article/pii/S1570870505001137.

[7] Min Cai et al. "Collaborative Internet worm containment". In: *IEEE Security Privacy* 3.3 (2005), pp. 25–33. DOI: 10.1109/MSP.2005.63.

[8] David Chismon and Martyn Ruks. "Threat intelligence: Collecting, analysing, evaluating". In: *MWR InfoSecurity Ltd* (2015).

[9] Bram Cohen. "Incentives build robustness in BitTorrent". In: *Workshop on Economics of PeertoPeer systems* 6 (June 2003).

[10] Giuseppe DeCandia et al. "Dynamo: Amazon's Highly Available Key-Value Store". In: *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*. SOSP '07. Stevenson, Washington, USA: Association for Computing Machinery, 2007, pp. 205–220. ISBN: 9781595935915. DOI: 10.1145/1294261.1294281. URL: https://doi.org/10.1145/1294261.1294281.

[11] Alan Demers et al. "Epidemic Algorithms for Replicated Database Maintenance". In: *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*. PODC '87. Vancouver, British Columbia, Canada: Association for Computing Machinery, 1987, pp. 1–12. ISBN: 089791239X. DOI: 10.1145/41840.41841. URL: https://doi.org/10.1145/41840.41841.

[12] Roger Dingledine. "Overhead from directory info: past, present, future". In: torproject.org, Feb. 2009. URL: https://research.torproject.org/techreports/overhead-directory-info-2009-02-1.pdf (visited on 05/02/2022).

[13] Roger Dingledine and Nick Mathewson. "Design of a blocking-resistant anonymity system". In: (Jan. 2006).

[14] Roger Dingledine, Nick Mathewson, and Paul Syverson. "Tor: The Second-Generation Onion Router". In: *13th USENIX Security Symposium (USENIX Security 04)*. San Diego, CA: USENIX Association, Aug. 2004. URL: https://www.usenix.org/conference/13th-usenix-security-symposium/tor-second-generation-onion-router.

[15] John R. Douceur. "The Sybil Attack". In: *Peer-to-Peer Systems*. Ed. by Peter Druschel, Frans Kaashoek, and Antony Rowstron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 251–260. ISBN: 978-3-540-45748-0.

[16] Md. Sadek Ferdous, Farida Chowdhury, and Md Moniruzzaman. "A Taxonomy of Attack Methods on Peer-to-Peer Network". In: Jan. 2007.

[17] Bc. Lukáš Forst. "Trust Model for Global Peer-To-Peer Intrusion Prevention System". MA thesis. May 2022. URL: https://www.stratosphereips.org/thesis-projects-list/2022/3/12/trust-model-for-global-peer-to-peer-intrusion-prevention-system.

[18] Sebastian Garcia. "Modelling the network behaviour of malware to block malicious patterns. the stratosphere project: a behavioural ips". In: *Virus Bulletin* (Oct. 2015), pp. 1–8.

[19] Sebastian Garcia. *SLIPS Github repository v0.8.5*. 2022. URL: https://github.com/stratosphereips/StratosphereLinuxIPS/tree/v0.8.5 (visited on 05/02/2022).

[20] Arjita Ghosh. "Agent-based distributed intrusion alert system /". In: (Jan. 2004).

[21] Jacob Heun. *Hardening the IPFS public DHT against eclipse attacks*. Oct. 2020. URL: https://blog.ipfs.io/2020-10-30-dht-hardening/.

[22] Ing. Dita Hollmannová. "Trust Models on Adversarial Distributed Security Agents". MA thesis. Aug. 2020. URL: https://dspace.cvut.cz/bitstream/handle/10467/90252/F3-DP-2020-Hollmannova-Dita-final.pdf?sequence=-1&isAllowed=y.

[23] Jana Iyengar and Martin Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. May 2021. DOI: 10.17487/RFC9000. URL: https://www.rfc-editor.org/info/rfc9000.

[24] R. Janakiraman, M. Waldvogel, and Qi Zhang. "Indra: a peer-to-peer approach to network intrusion detection and prevention". In: *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.* 2003, pp. 226–231. DOI: 10.1109/ENABL.2003.1231412.

[25] Xing Jin and S.-H. Gary Chan. "Unstructured Peer-to-Peer Network Architectures". In: *Handbook of Peer-to-Peer Networking*. Ed. by Xuemin Shen et al. Boston, MA: Springer US, 2010, pp. 117–142. ISBN: 978-0-387-09751-0. DOI: 10.1007/978-0-387-09751-0_5. URL: https://doi.org/10.1007/978-0-387-09751-0_5.

[26] Christopher Johnson et al. *Guide to Cyber Threat Information Sharing*. en. 2016. DOI: https://doi.org/10.6028/NIST.SP.800-150.

[27] Olga Kieselmann and Arno Wacker. "k-rAC - a Fine-Grained k-Resilient Access Control Scheme for Distributed Hash Tables". In: (Sept. 2016).

[28] Sebastian Klipper. "ISO/IEC 27005". In: *Information Security Risk Management: Risikomanagement mit ISO/IEC 27001, 27005 und 31010*. Wiesbaden: Vieweg+Teubner, 2011, pp. 63–97. ISBN: 978-3-8348-9870-8. DOI: 10.1007/978-3-8348-9870-8_3. URL: https://doi.org/10.1007/978-3-8348-9870-8_3.

[29] *LibP2P Documentation - Concepts*. URL: https://docs.libp2p.io/concepts/ (visited on 05/02/2022).

[30] *LibP2P Official Website*. URL: https://libp2p.io/ (visited on 05/02/2022).

[31] Petar Maymounkov and David Eres. "Kademlia: A Peer-to-peer Information System Based on the XOR Metric". In: vol. 2429. Apr. 2002. ISBN: 978-3-540-44179-3. DOI: 10.1007/3-540-45748-8_5.

[32] Rob McMillan. "Definition: Threat Intelligence". In: *Gartner* (2013).

[33] Alberto Montresor. "Gossip and Epidemic Protocols". In: 2017.

[34] *Multiaddresses - Github Repository*. URL: https://github.com/multiformats/multiaddr (visited on 05/02/2022).

[35] *Multihash - Github Repository*. URL: https://github.com/multiformats/multihash (visited on 05/02/2022).

[36] *P2P Network Documentation*. URL: https://developer.bitcoin.org/devguide/p2p_network.html (visited on 05/02/2022).

[37] Bernd Prünster, Alexander Marsalek, and Thomas Zefferer. "Total Eclipse of the Heart – Disrupting the InterPlanetary File System". In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/prunster.

[38] Guntur Putra et al. *Decentralised Trustworthy Collaborative Intrusion Detection System for IoT*. Oct. 2021.

[39] Sylvia Ratnasamy et al. "A Scalable Content-Addressable Network". In: *ACM SIGCOMM Computer Communication Review* 31 (Sept. 2001). DOI: 10.1145/383059.383072.

[40] Sylvia Ratnasamy et al. "A Scalable Content-Addressable Network". In: *ACM SIGCOMM Computer Communication Review* 31 (Sept. 2001). DOI: 10.1145/383059.383072.

[41] Martin Repa. *Iris - P2P System Github Repository*. 2022. URL: https://github.com/HappyStoic/iris (visited on 05/20/2022).

[42] Christian Rossow et al. "SoK: P2PWNED - Modeling and Evaluating the Resilience of Peer-to-Peer Botnets". In: *2013 IEEE Symposium on Security and Privacy*. 2013, pp. 97–111. DOI: 10.1109/SP.2013.17.

[43] Antony Rowstron. "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems". In: vol. 2218. Oct. 2002. ISBN: 978-3-540-42800-8. DOI: 10.1007/3-540-45518-3_18.

[44] Muhammad Saad et al. "Overview of Attack Surfaces in Blockchain". In: Mar. 2019, pp. 51–66. ISBN: 9781119519607. DOI: 10.1002/9781119519621.ch3.

[45] Emil Sit and Robert Morris. "Security Considerations for Peer-to-Peer Distributed Hash Tables". In: *Peer-to-Peer Systems*. Ed. by Peter Druschel, Frans Kaashoek, and Antony Rowstron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 261–269. ISBN: 978-3-540-45748-0.

[46] Rajeev Sobti and Geetha Ganesan. "Cryptographic Hash Functions: A Review". In: *International Journal of Computer Science Issues, ISSN (Online): 1694-0814* Vol 9 (Mar. 2012), pp. 461–479.

[47] Daniel Stutzbach and Reza Rejaie. "Understanding Churn in Peer-to-Peer Networks". In: *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*. IMC '06. Rio de Janeriro, Brazil: Association for Computing Machinery, 2006, pp. 189–202. ISBN: 1595935614. DOI: 10.1145/1177080.1177105. URL: https://doi.org/10.1145/1177080.1177105.

[48] tommy. *Introducing Bastet, Our New Directory Authority*. Nov. 2017. URL: https://blog.torproject.org/introducing-bastet-our-new-directory-authority/ (visited on 05/02/2022).

[49] J. Ullrich. *DShield*. URL: https://www.dshield.org/indexd.html (visited on 05/02/2022).

[50] Guido Urdaneta, Guillaume Pierre, and Maarten van Steen. "A survey of DHT security techniques". In: *ACM Comput. Surv.* 43 (Jan. 2011), p. 8. DOI: 10.1145/1883612.1883615.

[51] Cynthia Wagner et al. "MISP: The Design and Implementation of a Collaborative Threat Intelligence Sharing Platform". In: *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security*. WISCS '16. Vienna, Austria: Association for Computing Machinery, 2016, pp. 49–56. ISBN: 9781450345651. DOI: 10.1145/2994539.2994542. URL: https://doi.org/10.1145/2994539.2994542.

[52] Logan Washbourne. "A Survey of P2P Network Security". In: (Apr. 2015).

[53] Arctic Wolf. *The Fascinating Decade in Cybercrime: 2010 to 2020*. 2020. URL: https://arcticwolf.com/resources/blog/decade-of-cybercrime (visited on 05/02/2022).

[54] Vinod Yegneswaran, Paul Barford, and Somesh Jha. "Global intrusion detection in the DOMINO overlay system". In: (Jan. 2004).

[55] Ben Zhao et al. "Tapestry: A Resilient Global-Scale Overlay for Service Deployment". In: *IEEE Journal on Selected Areas in Communications* 22 (July 2003). DOI: 10.1109/JSAC.2003.818784.

[56] Zibin Zheng et al. "An overview on smart contracts: Challenges, advances and platforms". In: *Future Generation Computer Systems* 105 (2020), pp. 475–491. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2019.12.019. URL: https://www.sciencedirect.com/science/article/pii/S0167739X19316280.

[57] Zibin Zheng et al. "Blockchain challenges and opportunities: A survey". In: *International Journal of Web and Grid Services* 14.4 (2018), pp. 352–375.