GDAŃSK UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRONICS, TELECOMMUNICATIONS AND INFORMATICS

Student's name and surname: Mikołaj Nowak
ID: 184865
Cycle of studies: master's degree studies
Mode of study: full-time studies
Field of study: Informatics
Specialization: Computer Networks

MASTER'S THESIS

Title of the thesis: Analysis of technical solutions of anonymous communication networks (ACNs)

Title of the thesis (in Polish): Analiza rozwiązań technicznych wykorzystywanych w anonimizujących systemach sieciowych

Supervisor: dr inż. Krzysztof Gierłowski

STRESZCZENIE

Zapewnienie anonimowości komunikacji w różnorodnych systemach sieciowych stanowi wyzwanie techniczne, które wymaga rozwiązań zdolnych do realizacji tego celu przy jednoczesnym zachowaniu użyteczności systemu sieciowego. Niniejsza praca przedstawia analizę sieci anonimizujących (ACN) pod kątem ich użyteczności w różnych scenariuszach zastosowań. Głównym celem pracy jest wykazanie, że obecnie nie istnieje uniwersalna sieć ACN, która byłaby idealna dla każdego scenariusza użycia; każda z sieci radzi sobie inaczej w zależności od konkretnych wymagań, a wybór sieci powinien być determinowany przez zamierzone zastosowanie.

W pracy przedstawiono przegląd sieci anonimizujących, koncentrując się na ich aspektach technicznych i wyróżniając najważniejsze współczesne rozwiązania, jak również przegląd literatury powiązanej pod kątem analizy porównawczej sieci. Ponadto, zidentyfikowano potencjalne przypadki użycia oraz obszary zastosowań, sklasyfikowano je w oparciu o wymagania, które muszą być spełnione dla danego przypadku użycia, a także przedstawiono potencjalne zagrożenia dla sieci ACN.

Aby poprzeć tezę, że różne sieci anonimizujące są optymalne w różnych scenariuszach, przeprowadzono wielokryterialną analizę porównawczą, łączącą porównanie oparte na literaturze z pomiarami eksperymentalnymi. Umożliwiło to ocenę najważniejszych obecnie sieci ACN: Tor, I2P, Lokinet oraz Nym, pod kątem ich użyteczności i efektywności w różnych grupach zastosowań.

Wyniki analizy porównawczej jednoznacznie pokazują, że nie istnieje jedna uniwersalnie optymalna sieć anonimizująca; najlepszy wybór zależy od konkretnego przypadku użycia. Praca kończy się rekomendacjami dotyczącymi przyszłego rozwoju sieci ACN.

W celu wsparcia praktycznego zrozumienia wyników analizy porównawczej opracowano demonstrator edukacyjny. Umożliwia on studentom zrozumienie rezultatów przeprowadzonej analizy poprzez praktyczne ćwiczenia laboratoryjne oraz obserwację mocnych i słabych stron różnych sieci anonimizujących w odniesieniu do wybranych obszarów zastosowań.

Słowa kluczowe: Tor, I2P, Loki, Nym, sieci anonimizujące, ACN, analiza, przypadek użycia, obszar zastosowania, wykorzystanie

ABSTRACT

Ensuring the anonymity of communication in diverse network systems is a technical challenge that requires solutions capable of achieving this objective while maintaining the usability of the network system. This thesis presents an analysis of anonymous communication networks (ACNs) in terms of their usability in various use case scenarios. The main goal of the work is to demonstrate that currently there is no universal ACN that is ideal for every use case scenario; each network performs differently depending on the specific requirements and the choice of ACN should be determined by the intended application.

This work provides an overview of ACNs, focusing on their technical aspects and distinguishing the most prominent solutions today, as well as an overview of related work in terms of comparative analysis of ACNs. Furthermore, the work identifies potential use cases and application areas, categorises them based on the requirements that need to be fulfilled for a given use case, and presents potential threats for the ACNs.

To support the claim that different ACNs are optimal for different scenarios, a multi-criteria comparative analysis is performed, combining literature-based comparison with experimental measurements. This enables the evaluation of the most prominent ACNs today: Tor, I2P, Lokinet, and Nym, in terms of their usability and effectiveness across different use case groups.

The results of the comparative analysis clearly demonstrate that no single ACN is universally optimal; instead, the best choice depends on the specific use case. The work concludes with recommendations for the future development of ACNs.

To support a practical understanding of the findings of the comparative analysis, an educational demonstrator was developed. This demonstrator enables students to understand the results of the performed analysis through hands-on laboratory exercises and observe the strengths and weaknesses of various ACNs in terms of different application areas.

Keywords: Tor, I2P, Loki, Nym, anonymous communication network, ACN, analysis, use case, application area, usage

TABLE OF CONTENTS

ABBREVIATIONS

ACN – Anonymous Communication Network

AWS – Amazon Web Services

DH – Diffie-Hellman

DHT – Distributed hash table

DC – Dining cryptographers

DoS – Denial of Service

DPI – Deep Packet Inspection

ECC – Eliptic-curve cryptography

EFF – Electronic Frontier Foundation

GCP – Google Cloud Platform

HSDir – Hidden service directory

I2P – Invisible Internet Project

IIP – Invisible IRC Project

IOI – Item of interest

IoT – Internet of Things

IP – Introduction point or Internet Protocol

IRC – Internet Relay Chat

JAP – Java Anon Proxy

LLARP – Low Latency Anonymous Routing Protocol

NAT – Network Address Translation

ONR – Office of Naval Research

OP – Onion proxy

OR – Onion router

P2P – Peer-to-peer

PDU – Protocol data unit

PQC – Post-quantum cryptography

PT – Pluggable transport

RP – Rendezvous point

RTT – Round Trip Time

SI – Slice ID

SURB – Single-use Reply Block

VoIP – Voice over IP

VPN – Virtual Private Network

WWW – World Wide Web

1. INTRODUCTION

This chapter serves as a starting point for the thesis. It defines the motivation for taking up the topic of anonymous communication networks (ACNs) and outlines the main goals of the work. The chapter also specifies the scope of the analysis and presents the structure of the thesis. The intention is to give a clear orientation before moving on to the technical and analytical content that follows. This introduction helps to understand why the subject was chosen, what will be analysed, and how the thesis is organised.

1.1. Goal of the work

Ensuring the anonymity of communication in diverse network systems is a technical challenge that requires solutions capable of achieving this objective while maintaining the usability of the network system. The aim of this work is to analyse the types of technical solutions currently used in anonymising networks/anonymous communication networks (ACNs) in the context of their usability in specific usage scenarios.

1.2. Scope of the work

This work provides an overview of ACNs, emphasising their technical aspects, identifies potential use cases and application areas, categorises these use cases, presents potential threats, and offers a comparative analysis of existing anonymous communication networks in terms of usability in various scenarios. The work also determines the desired directions for the development of ACNs and includes the design and implementation of an educational demonstrator to verify and illustrate key elements of the analysis.

1.3. Structure of the work

This work begins with Introduction, which presents the objectives and structure of the thesis. The second chapter is Theoretical introduction, which explains all the theoretical concepts necessary to understand the paper. The third chapter is Overview of the anonymous communication networks. It provides an overview of the most prominent ACNs is provided, focusing on those most popular today and those that have significantly influenced current solutions. Then, in the fourth chapter Related work, the related work on the analysis of anonymous communication networks is presented. Following the related work, in the fifth chapter Use cases and application areas, the use cases for ACNs are proposed. The use cases are then organised into groups of similar requirements. The criteria for each group are selected and appropriately weighted. After presenting the use cases, in the sixth chapter Threats, attacks and limitations the relevant threats are discussed, including limitations and possible attacks. Subsequently, in the seventh chapter Multi-criteria comparative analysis, the technical comparison of the most popular

ACNs and their technical solutions is carried out in two stages. The first stage is the comparison based solely on the literature. Then, the experiments are performed and the results are used in the second stage, an empirical and experimental comparison. After comparisons, ACNs are evaluated in terms of usability for the identified use case groups and their requirements. Based on this evaluation, the best-suited ACN is identified for each use case group. Following the analysis, in the eighth chapter Future directions, the future directions of ACNs are described. After presenting future directions, in the ninth chapter Educational demonstrator, an educational demonstrator is designed to verify and illustrate the ACN analysis. Finally, the work concludes with Summary a summary of the findings and a discussion of the results. There are also appendices; Appendix A: Bash scripts for hidden services setup provides scripts for setting up the services that will be used to perform measurements, Appendix B: Python application for measuring RTT and jitter provides the application used for measurements, Appendix C: Educational demonstrator: Theory and practical exercises provides the laboratory instruction from the educational demonstrator along with the theory introduction for the laboratory, and Appendix D: Educational demonstrator: ACN simulator includes the code for the simulator that is used within the educational demonstrator.

1.4. Summary

In summary, this chapter sets the foundation for the thesis by clearly stating its objectives, establishing the boundaries of the analysis, and presenting the overall structure that will be followed in the next chapters. Setting these foundations is crucial for presenting the reasoning behind the work and to know what to expect in the thesis. The introduction clarifies the function of each part of the thesis and shows how the analysis will be approached in the following chapters.

2. THEORETICAL INTRODUCTION

This chapter introduces the basic concepts that are needed to understand future chapters. It collects definitions and technical background that are referenced throughout the thesis. The aim is to give the reader enough context to follow the overview and analysis of anonymous communication networks (ACNs). The chapter is organised as follows: it starts with the definitions of anonymity and privacy, then describes distributed hash tables and basic concepts from cryptography, and finally presents the network models, blockchain and cryptocurrency foundations, and an overview of protocols that are referenced in later sections. All these elements are important to understand the technical and analytical parts of the thesis.

2.1. Anonymity and privacy

The definition of anonymity used in this thesis is based on a paper titled "A terminology for talking about privacy by data minimization" [1]. According to the paper: "Anonymity of a subject means that the subject is not identifiable within a set of subjects, the anonymity set.". Another important property in anonymous communication systems is unlikability. According to the same paper: "Unlinkability of two or more items of interest (IOIs, e.g., subjects, messages, actions, ...) from an attacker's perspective means that within the system (comprising these and possibly other items), the attacker cannot sufficiently distinguish whether these IOIs are related or not." Pseudonymity refers to the use of pseudonyms as identifiers, where a pseudonym is an identifier of a subject other than one of their real names. Privacy is the right to keep personal matters and relationships secret, as well as the ability to determine when, how, and which information about oneself is revealed. Anonymity can be considered as a method to achieve privacy.

Why is anonymity and privacy needed?

For the same reason that in all well-prosper democratic countries there are anonymous elections. No one should be obliged to reveal information they do not wish to disclose; individuals should have the right to decide to whom and what information they reveal. In recent years, privacy has been increasingly compromised as personal data have become highly valuable to large corporations. Data are sometimes said to be the petroleum of the 21st century, highlighting their significant value in the modern economy. As with many technologies, anonymity can be used for both positive and negative purposes. Although negative uses often receive more attention in the media, this thesis will primarily focus on beneficial use cases to provide a balanced perspective. It should be noted that attempts to restrict anonymous networks primarily affect users with legitimate intentions, as malicious actors are often able to circumvent such restrictions and may use alternative solutions that do not require scalability.

Some organisations claim they do not retain the content of communications, but only metadata - such as who communicates with whom, when, where, and how often. NSA General Counsel Stewart Baker said: "metadata absolutely tells you everything about somebody's life. If you have enough metadata, you don't really need content." and General Michael Hayden, former NSA director, added: "We kill people based on metadata" [2]. Metadata, in fact, should not be neglected when it comes to the privacy topic. According to the European Data Protection Supervisor, privacy is a fundamental human right and a component of a sustainable democracy [3].

Anonymity by design and anonymity by policy

Sometimes service providers, Virtual Private Network (VPN) service providers in particular, advertise their services as anonymous, although in reality they can deanonymise users on demand; they just claim that they would not do it or that they keep logs of their user activity. However, history shows that such assurances are not always reliable and that users cannot be certain that anonymous service providers will not disclose or sell their information [4]. This approach is known as anonymity by policy. On the other hand, anonymity by design makes it impossible for the service provider to reveal the identity of the service users by creating a service or network in such a way. In other words, there is no need to trust any third party other than the code itself, which you can verify in most cases because they are usually open source. Proper privacy can only be achieved with anonymity by design.

Anonymous communication network

Anonymous Communication Network is a kind of network that allows anonymous uni- or bi-directional communication. They are usually implemented as overlay networks. An overlay network is a logical network that operates on top of an underlying physical network.

Clearnet, Deep Web, and Hidden Services

Clearnet refers to the publicly accessible portion of the Internet. It includes a wide variety of protocols and services, including, but not limited to, the World Wide Web. All resources on the clearnet are accessible without the use of ACNs. The **surface web** is a subset of the clearnet and denotes the part of the World Wide Web that is indexed by conventional search engines such as Google or Bing. It consists of publicly available web pages that can be accessed without authentication or authorisation.

The **deep web** refers to the segment of the World Wide Web that is not indexed by standard search engines. This includes resources such as private databases, academic repositories, webmail, and other pages that require authentication, authorisation or are otherwise hidden from public view.

The terms **darknet** and **hidden services** refer to a broad range of services and resources that are anonymous themselves, especially in terms of their location, and are accessible exclusively through specialised anonymous communication networks (ACNs). These services are not restricted to the World Wide Web. The **dark web** or **hidden web services** specifically describe web-based resources that are

only accessible through such networks and not through standard browsers or search engines. Although these concepts are often associated with negative media coverage, it is important to emphasise that such technologies also enable legitimate uses, including privacy protection, secure communication, and the promotion of free expression. For the sake of clarity, the term hidden services will be used throughout this thesis when referring to these resources.

2.2. Distributed Hash Tables

A Distributed Hash Table (DHT) is a decentralised system for storing key-value pairs, where keys are hashes of fixed length and evenly distributed across a large keyspace, allowing data to be stored across many machines instead of relying on a single central server. This distribution improves reliability, avoids single point failure, and eliminates the need to trust a single entity, with various strategies used to assign keys to specific machines. One of the first proposed DHTs was Chord [5]. It organises nodes and keys in a ring using consistent hashing, where each node maintains a set of pointers ("fingers") to efficiently locate keys in $O(log\ (n))$ hops. Keys are stored at their closest successor node, ensuring even distribution and scalable lookups across the network. Another popular DHT is Kademlia [6]. It uses an XOR-based distance metric to organise the nodes in keyspace, allowing each node to efficiently locate values near itself by recursively querying the nodes closer to the target, resulting in $O(log\ (n))$ lookup hops. The nodes keep more detailed information about the nearby nodes, structuring their routing tables as binary trees based on the bitwise distance.

2.3. Cryptography basics

Cryptography is a science of concealing the meaning of a message. This section introduces the basic cryptographic concepts and algorithms that will be referenced throughout the thesis.

Information security attributes

Information security, or InfoSec, is a practice of protecting information. Several attributes can be distinguished, slightly varying depending on the definition. From the point of view of this work, the most important will be four of them:

1. Confidentiality
2. Authenticity
3. Integrity
4. Non-repudiation

Confidentiality means that only authorised persons can access the data. In cryptographic systems, it is ensured by the possession of the decryption key. **Authenticity** relates to verifying that the identity of the entity performing a given action is as declared. **Integrity** ensures that the data have not been modified in an unauthorised way. **Non-repudiation** guarantees that parties involved in data exchange cannot deny their participation later.

Symmetric cryptography

In symmetric cryptography, one shared key is used on both sides of a communication to encrypt and decrypt messages. Symmetric cryptography ensures confidentiality, as (ideally) the key is needed in order to read the content of a message, and the key is only shared between two sides of the communication that are often referred to as Alice and Bob. Symmetric cryptography ciphers can be divided into two subcategories. The first is the **block ciphers**, where fixed-size blocks are encrypted. In case the data does not fit the block, a padding is used. The second is **stream ciphers**, where only one bit or byte is encrypted at a time.

Asymmetric cryptography

In asymmetric cryptography, or public-key cryptography, each entity uses a pair of keys instead of one shared symmetric key. The private key is a part that, as the name suggests, is specific to the entity exclusively and cannot be revealed to anyone else. However, the public key can be freely distributed. The message encrypted with the public key can only be decrypted with the private key from the same pair. Analogically, the message "encrypted" with the private key can be "decrypted" with the public key, although calling it encryption and decryption does not make much sense, as the public key is freely distributed; therefore, everyone can read the message. The more appropriate terms that are used in this example are signing and verifying, as if a given message can be decrypted with a certain public key, then certainly the one who signed (encrypted) it must be the owner of the private key from the same key pair. Encryption with asymmetric cryptography is in general much slower than with symmetric one; therefore, it is most often used for different purposes, such as the digital signature, which will be described later.

Elliptic-curve cryptography (ECC)

An approach to asymmetric cryptography using elliptic curves over finite fields has gained popularity in recent years, although it is not a new technique, as its origin dates back to 1985. ECC is believed to offer the same level of security as other widely used asymmetric cryptographic methods but with significantly shorter key lengths. One of the popular examples of a curve that can be efficiently used in this kind of cryptography is Curve25519.

Hash functions

Hash function algorithms allow for transformation of any input into a fixed-size output, called a hash, in a deterministic way, which means that each time a certain input produces the same output if the same hash function and its parameters are used. Another important feature of hash functions is its one-sidedness, in that the original input cannot be determined based on the produced hash. The hash function should also produce output that is uniformly distributed, and it should be hard to find two inputs that produce the same output - this situation is called a hash collision.

Digital signature

One of the most popular use cases of asymmetric cryptography is the digital signature. It is a mathematical scheme for verifying the authenticity, integrity, and non-repudiation of origin for the given message. In a simplified way, it works as given: The sender generates a private/public key pair and distributes the public key. He generates a hash of the message he has written and signs this hash with his private key. The receiver receives the message along with the signed hash. As the hash function is known, he creates a hash of the message himself with it, verifies the signed hash, and compares the two result hashes. If they are equal, then the signature is valid.

Quantum and post-quantum cryptography

Quantum cryptography is the science of using quantum mechanics in cryptography. Quantum cryptography algorithms are designed to work specifically on quantum computers. On the other hand, post-quantum cryptography (PQC) refers to the development of cryptographic algorithms that work on traditional machines but that are resistant to potential attacks that use quantum computers.

Popular cryptographic algorithms and hash functions

In the field of cryptography, several algorithms and hash functions are fundamental to the construction of secure systems. The following are the most significant ones relevant to ACNs:

- AES
- RSA
- ElGamal
- DSA
- ECDSA
- EdDSA
- Diffie-Hellman Key Exchange
- ECDH
- SHA

AES, or Advanced Encryption Standard, is the most popular block cipher today, considered a safe symmetric cryptography solution, especially with the longest possible 256-bit keys. Due to its popularity, it often has dedicated hardware support. **RSA** is one of the first asymmetric cryptography algorithms and still the most popular. The name RSA is an abbreviation from its creators' surnames: Rivest, Shamir, Adleman. Its security relies on the difficulty of the factoring problem, which means factoring the product of two large prime numbers. It can be used for both encryption and digital signature. **ElGamal** is the second most popular asymmetric cryptography algorithm after RSA. It is based on the discrete logarithm problem. It supports both encryption and digital signatures. **DSA**, or Digital Signature Algorithm, is an asymmetric cryptography algorithm used specifically for digital signatures, not for encryption. It is based on the discrete logarithm problem. **ECDSA**, meaning elliptic-curve

DSA, is a variant of DSA that uses elliptic-curve cryptography. **EdDSA**, meaning Edwards-curve DSA, is a variant of DSA utilising Schnorr signature based on twisted Edwards curves. Although the name might suggest correlation with ECDSA, both are completely different signature schemes. Ed25519 is a specific example of the EdDSA variant and uses the edwards25519 curve, which is related to the popular Montgomery curve Curve25519. **Diffie-Hellman Key Exchange**, or simply Diffie-Hellman (DH), is a key agreement algorithm that two parties can use to agree on a shared secret. The shared secret is converted into keying material, and the keying material is used as a symmetric encryption key. The Diffie-Hellman key exchange uses the properties of modular arithmetics. **ECDH** is an elliptic-curve version of the DH key agreement. Usually, Curve25519 is chosen as a curve because it is fast and not proprietary. **SHA**, or Secure Hash Algorithms, are a family of hash functions. Currently, there are four versions of SHA. **SHA-0** refers to the original hash function with 160-bit output that was published under the name SHA. **SHA-1** is an improved version of SHA-0, still using 160 bits. NSA designed it as part of the DSA algorithm. **SHA-2** is a family of two hash functions designed by NSA: SHA-256 and SHA-512. The postfix in the name determines the length of the output of the hash function (256 bits or 512 bits). There are also modified and truncated versions of these two standards: SHA-224, SHA-384, SHA-512/224, and SHA-512/256. **SHA-3** is the latest hash function of the SHA family, also known as Keccak. Although the hash lengths are the same as in the SHA-2 family, the inner workings of the algorithm are significantly different.

2.4. ISO/OSI reference model and TCP/IP suite

The two most popular networking models used today are ISO/OSI and TCP/IP. They both help us to analyse the end-to-end communication between two parties. They both consist of several layers: ISO/OSI consists of 7, while TCP/IP is of 4. Each of these layers has certain clear goals to fulfill. Each layer has certain protocols and protocol-specific units of information composed of user data and control information called protocol data units (PDUs).

The ISO/OSI model has the following layers:

1. Physical layer
2. Data link layer
   (a) Media Access Control sublayer
   (b) Logical/Data Link Control sublayer
3. Network layer
4. Transport layer
5. Session layer
6. Presentation layer
7. Application layer

Physical layer is the first and lowest layer, responsible for the transmission of unstructured data between a device and a transmission medium. The PDU of the physical layer is a bit.

Data link layer is the second layer, responsible for node-to-node data transfer between two directly connected devices. In this layer, two sublayers can be distinguished; **Media Access Control**, usually implemented on a hardware, manages access control, encapsulates and decapsulates data, adds header and trailer, while **Logical/Data Link Control (LLC/DLC)** is responsible for communication with physical and network layers and for flow control. The PDU of the data link layer is a frame. In real-world examples, technologies designated for the lowest layers usually cover both the physical and data link layer. Examples include Ethernet, the most popular wired local area network (LAN) technology today, or 802.11, sometimes referred to as Wi-Fi, the most popular wireless LAN technology.

Network layer is the third layer, responsible for addressing hosts, ensuring connectionless communication, routing, and relaying messages. It is also the first layer that is "sentient" of the network, which means that there is something outside the current machine or current link. The most popular protocol of this layer is the IP protocol in two versions: IPv4 and IPv6. The PDU of the network layer is a packet.

Transport layer is the fourth layer, responsible for interprocess communication. Its role is to address ports and control connection, flow and errors as well as message multiplexation and demultiplexation. There are two most popular protocols in this layer: TCP, responsible for reliable communication with the cost of delays, and UDP, responsible for unreliable communication, but with smaller delays. The PDU in the transport layer is a segment for TCP and a datagram for UDP.

Session layer is the fifth layer, responsible for maintaining and managing interprocess sessions. The PDU of the session layer is data.

Presentation layer is the sixth layer, sometimes referred to as the syntax layer, which is responsible for translating, coding, and decoding data between applications. The PDU of the presentation layer is also data.

Application layer is the seventh and last layer in the ISO/OSI model application layer that is responsible for providing an interface for communication between applications. Once again, the PDU of the application layer is data.

For the sake of simplicity, it can be said that in the TCP/IP model there are four layers that correspond to one or more ISO/OSI layers:

1. Link layer
2. Internet layer
3. Transport layer
4. Application layer

Link layer is the layer that serves as a combination of the first and second layers of ISO/OSI. **Internet layer** is the layer that corresponds to the network layer of the ISO / OSI model. **Transport layer** corresponds to the layer with the same name from the ISO/OSI model. **Application layer** combines the sessions, presentations and application layers of the ISO / OSI model.

The ISO/OSI was a model created from the ground up in the late 1970s and early 1980s as a

framework based on the set of tasks that needs to be fulfilled in order to have reliable communication and each layer in this model was assigned a specific role. One of the most criticised elements of the ISO/OSI model were layers 5 and 6 which were thought to be an exaggeration, as they can be easily integrated with the application layer. That is also why often, when referring to the application layer, the fifth, sixth, and seventh layers are grouped together. Regardless of the criticism, the model is still used as a reference model.

TCP/IP is an evolving model whose origins date back to ARPANET [7]. It has a simpler, more concise, and pragmatic approach compared to the ISO/OSI model, and it has specific protocol solutions. An important difference between ISO/OSI and TCP/IP models is the fact that in the ISO/OSI the communication is only possible between adjacent layers, while in TCP/IP any layer can refer to any other layer, including the same layer. An example can be the IP protocol utilising ICMP and vice versa.

This paper will focus on layers 3-7 from the ISO/OSI model (or layers 2-4 from the TCP/IP model). When referring to the seventh layer from the ISO/OSI model, fifth and sixth will also be included implicitly in the reference for the sake of simplicity.

2.5. Blockchain and cryptocurrencies

Blockchain is a shared, immutable, and distributed ledger that records information in a growing list of entries called blocks. Blocks are chained together using hash functions. Each new block depends on the integrity of the previous blocks in an unaltered stage and therefore the history of a blockchain cannot be altered. Blockchains are often used as the backbones of various cryptocurrencies, digital currencies based on cryptography, where transactions are written as records on the blockchain with an appropriate consensus mechanism. In today's world, cash usage is declining as countries shift toward cashless payments. This trend poses a major threat to privacy. Unlike cash transactions, which are private and typically known only to the two parties involved, cashless payments lack this privacy. Every cashless payment is visible to the government, tax authorities, banks, and possibly many more entities. This information can be used against the people who make such transactions. Cryptocurrencies offer a way to restore the privacy of transactions. They allow for private peer-to-peer transactions between users. Additionally, cryptocurrencies have excellent transfer properties, enabling reasonably quick transfers, regardless of the day of the week, if it is a national holiday or not. Unlike traditional bank transfers, cryptocurrency transactions cannot be blocked or restricted by financial institutions or government entities.

2.6. Network protocols overview

This section briefly describes the most important network protocols that are referenced throughout the thesis. These protocols are relevant to understanding the mechanisms behind anonymous communication networks.

- IP

- TCP
- UDP
- TLS
- SOCKS
- BitTorrent

IP is the network layer communication protocol in the ISO/OSI model and the Internet layer protocol in the TCP/IP are responsible for connectionless communication, relaying datagrams, and addressing hosts. It is the most popular protocol for this layer. IP is often associated with convergence as it serves as a universal protocol that supports both upper and lower layer protocols. The IP protocol exists in two versions: IPv4 and IPv6. The IPv4 was introduced first and it had some significant flaws, among which the most critical one is the address space - it was not predicted that the Internet would be so huge that 4 billions of potential addresses would not be enough; as is known today, it is definitely not. There are many actions taken to postpone the issues related to the exhaustion problem as much as possible like the introduction of classless addressing, public/private address distinction with NAT mechanism, but the only viable long-term solution is moving on to the newer version of the IP protocol, IPv6. It can have up to 2^{128} addresses.

TCP is one of the two most popular transport layer protocols responsible for interprocess connection-based communication. Provides connection, flow, and error control. TCP is used when reliability is more important than potential delays.

UDP is second of the two most popular transport layer protocols responsible for connectionless interprocess communication. It lacks error control and flow control. It does not guarantee the delivery of a datagram or its correct order; however, these mechanisms can be implemented in the higher layers.

TLS, or Transport Layer Security, is a protocol widely used to ensure confidentiality and authentication for client-server applications. The predecessor of the TLS protocol was SSL; therefore, TLS is sometimes still referred to as SSL for historical reasons.

The **SOCKS** protocol allows network packets to be exchanged between the client and the server with an intermediary proxy server between. Currently, the latest version of this protocol is SOCKS5.

BitTorrent is a peer-to-peer protocol for file sharing. Its important feature is decentralisation, meaning that there is no possibility to take down one central server as it was for some other peer-to-peer file-sharing solutions like Napster. Users utilise BitTorrent clients, the programs that allow them to share files via BitTorrent. The communication is optionally assisted by the BitTorrent trackers that provide available files and find peer users, known as seeds. The other option, depending on the BitTorrent client implementation, is, for example, a distributed hash table as an alternative for the trackers. In BitTorrent files are divided into pieces and the file that holds a list of these pieces is called Torrent. After downloading a piece, the user starts sharing it with other peers, becoming one of the sources of this piece (seed). Each piece is associated with its hash to provide integrity.

2.7. Summary

This chapter provided the technical background and definitions required in the rest of the thesis. The discussed topics include anonymity and privacy, distributed hash tables, cryptography basics, network models, blockchain and cryptocurrencies, and a short summary of the most important network protocols. By collecting these fundamental concepts in one place, this chapter ensures that the reader has the necessary knowledge for understanding the design and evaluation of anonymous communication networks in the next chapters. Its function is to make the thesis self-contained and to make later technical discussions easier to understand.

3. OVERVIEW OF THE ANONYMOUS COMMUNICATION NETWORKS

This chapter reviews the history and evolution of Anonymous Communication Networks (ACNs), highlighting key milestones, technical developments, and influential systems that have shaped the field. The aim here is to provide the necessary background and context for understanding how modern ACNs have developed, what problems they were designed to solve, and why specific technical solutions were introduced. By presenting the main network families along with notable systems built on these designs, this chapter sets the foundation for all later analysis.

This overview is intended not only to familiarise the reader with the essential concepts and terminology, but also to clarify the motivation behind each architectural choice. Ultimately, this chapter serves as a reference point for the rest of the thesis, making it possible to understand the specific technical solutions that will be compared in detail later in the thesis.

The chapter begins with a historical overview and then presents the main technical approaches: single-hop proxies, Mix-nets, DC-nets, anonymous publication systems, onion routing, each with description of representative systems and their technical solutions. The chapter concludes with a classification and summary.

3.1. History of ACNs

The first paper that described the idea of ACN is considered David Chaum's master's thesis on Mix networks [8], written in 1979, published in 1981. To this day, most of the anonymous communication systems utilise ideas that were first described by Chaum. The Mix-nets design was ahead of its times, not only in terms of demand for anonymous communication, but also in terms of the computational power needed to repeatedly perform complex public-key cryptographic operations. It took more than a decade for the first anonymous communication networks to be successfully deployed.

The first ACN solutions were remailers, proxy servers that received messages and passed them accordingly to the instructions that were attached to the message. The first widely used ACN-like solution was the Penet remailer [9], deployed in 1993. However, it did not yet utilise Chaum's concept; it was a simple single-hop pseudonymous remailer, also known as a Type 0 remailer, and did not ensure anonymity by design. It served as a proxy that stripped metadata from the message and forwarded it to the recipient, allowing for replies. The Penet had several security flaws, including the lack of encryption and the logging of user activity, and was eventually discontinued.

Around the same time, although initially less known, the Cypherpunk remailer was introduced, otherwise known as a Type I remailer. It was associated with the Cypherpunk movement, a movement that, since the late 1980s, promoted cryptography and privacy for everyone, rather than limiting them to military use, as had previously been the case. It placed strong emphasis on encryption and lack of logging, and enabled users to chain their messages through multiple servers. It was the first real-life

implementation of Chaum's concepts, although only partially.

Another important implementation was Mixmaster, also known as a Type II remailer, created around 1995. It implemented the most important Mix-nets concepts: layered encryption, batching, and mixing, and gained significant recognition. Over the years, many Mix network-based solutions have been introduced, addressing issues related to the previous designs.

Mix-nets are not the only design that was proposed. In 1988 David Chaum published a paper about DC-nets [10] - an alternative approach for achieving anonymity for both sender and receiver. Similarly to Mix-nets, it provided a foundation for further research, and many later designs based their principles of operation on DC-nets. Even though there have been attempts of deploying ACNs based on DC-nets, none of the ACNs used today utilise this approach due to several limitations.

Another breakthrough design was onion routing [11, 12], a low-latency, connection-based, bidirectional communication network that, similar to the Mix-nets, utilises layered encryption. The work on it began in 1995 and was funded by the Office of Naval Research (ONR), the science and technology agency for the U.S. Navy. The first paper on onion routing was published in 1996; however, the final version was published in 1997, and a year later it appeared in the *IEEE Journal on Selected Areas in Communications*. Most ACNs used today are based on onion routing architecture.

One of the examples is Tor - the most popular ACN today, first deployed in 2002 as a direct successor to the original onion routing idea that was meant to be used by common people. The design was described in the 2004 paper [13]. During the years, it evolved and introduced many new features that can be seen today. Another prominent project that is based on a slightly modified onion routing is I2P - the modification is named garlic routing. I2P is a peer-to-peer network, and every user also acts as a router. Theoretically, the work on I2P started in 2001; however, the architecture was completely redesigned in 2003. Similarly to Tor, it is constantly evolving.

One of the newer onion routing-based ACNs that can be used today is Lokinet, first described in 2018 [14]. Lokinet implements a custom protocol called LLARP, which stands for Low Latency Anonymous Routing Protocol, and, contrary to Tor and I2P, it is placed on the network layer and can support any IP-based protocol.

Although onion routing dominated the scene of ACNs, they have some drawbacks, particularly vulnerability for traffic analysis. Even though Mix-nets are more resistant to this kind of attack, they suffer from significant latency. Loopix, proposed by Piotrowska et al. in 2017 [15], is an acceptable latency Mix-net architecture that led to a new rise in research on Mix-nets. One of the results of this increase is the Nym network [16], first described in 2021. The official launch of NymVPN, a service that allows the utilisation of the Nym network, was carried out in early 2025.

3.2. Single-hop proxies

The principle of operation of single-hop proxies is straightforward: strip the metadata related to the sender before forwarding it to the recipient. The major advantage of them is their low latency

due to simplicity and the fact that there is only one node between the sender and the receiver. However, such a single point of failure is not a desirable characteristic of an anonymous system, regardless of the claims of not keeping sensitive user data, as it imposes anonymity by policy rather than anonymity by design. As this architecture does not provide the anonymity by design property, it will not be treated as a proper anonymous communication network, however, due to its historical meaning it is mentioned here.

Notable examples from this category include Penet [9] and Anonymizer [17]. In addition, many VPN services commonly used today can also be considered examples of this category.

3.2.1. Penet

Penet was the first widely known single-hop proxy and the most prominent example of the family of pseudonym-based solutions, called pseudonymous remailers or nym servers. Sometimes, they are also referred to as Type 0 remailers. Their common characteristic is that they remove sender-specific information that could potentially identify senders, assign them a pseudonym, and forward the message to the recipient with the possibility of reply. Penet was created in 1993 by Johan "Julf" Helsingius from Finland. The creation was a result of a discussion on a Finnish Usenet newsgroup about whether people should be required to use their real information for online communication. Johan believed that if such accountability were enforced on Internet users, they would find a way around it. To prove his point, he created Penet.

Remailers serve as an intermediary between sender and receiver. Penet, which served as such a remailer, removed metadata that could potentially identify the sender and then forwarded the message without sensitive data to the receiver. It also served as an e-mail box that allowed users to assign their pseudonymous identities to their messages and receive messages sent to their anonymous addresses.

Penet had some vulnerabilities that could potentially compromise its users. For example, it sends the messages in plaintext. The lack of confidentiality of these messages could have led to the disclosure of sensitive information in the presence of an eavesdropper. Penet also kept a list of its users with their real email addresses and its mapping to the anonymous ones. It posed a threat in case of breaking into the server. Penet was closed due to legal issues in 1996.

3.3. Mix-nets

The first system that provided anonymity by design was described by David L. Chaum in his master's degree thesis, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms" in February 1979. Mix networks use public-key cryptography to hide both the sender and receiver of an electronic mail, as well as the content of their communication. They do it by encrypting a message with multiple layers of public key encryption and passing them through multiple nodes called mixes. Each node changes the message order, delays them, adds random information and padding in order to minimise the risk of traffic analysis and associating the sender with a given message; in other words,

to obfuscate correlation between the input and the output. It also removes the corresponding layer of encryption and passes it on to the next node on the path. The simplified Mix-net architecture is presented in Figure 3.1; each mix within the Mix-net decrypts a message that was previously encrypted with its public key, permutes the messages, and then forwards them to the next node or the final destination.

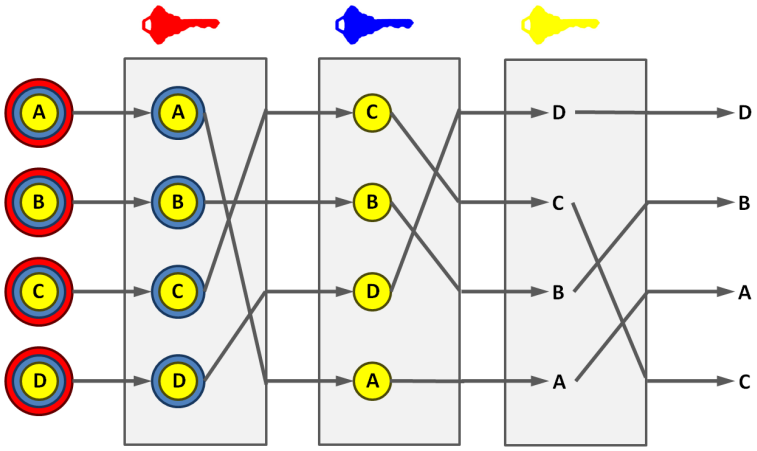

Fig. 3.1. Simplified Mix-net architecture. By Primepq, Wikimedia Commons [18], licensed under CC BY-SA 3.0.

In essence, it can be said that the sender knows all nodes in the path, each node knows only its neighbours, and the receiver knows only the final (exit) node. So, the identity of the recipient is only known to him, the sender, and the exit node; and the identity of the sender is only known to the first (entry) node.

The sender can send his message anonymously while the recipient can respond to the untraceable return address constructed using a real address, two random strings, and public-key cryptography. Communication participants exchange these addresses through multiple nodes, but only the addressee can decrypt them.

In 1964 [19] Paul Baran, a Polish Jewish American engineer, proposed a solution to the traffic analysis problem; the downside of this solution was that it required a trusted common authority. Chaum's Mix-net network does not need a trusted third party to achieve resilience to traffic analysis.

Mix-nets also propose digital pseudonyms in order to identify entities online. They are nothing more than public keys that can be used for signature verifications performed by an anonymous entity. The authority creates a list of pseudonyms and can decide which are trustworthy and which are not. The pseudonyms contained in a letter from each entity to the authority are processed together in a single batch and create together a complete roster. These letters can also be protected to provide an additional level of security. A person cannot be traced by his pseudonym in the final list; therefore, it allows verifying and communicating securely with assured anonymity.

One of the major disadvantages of the Mix-net is high latency correlated with public key cryptography, delays, and other aspects that are supposed to provide anonymity; therefore, anonymity

for more latency-sensitive purposes must use other solutions.

3.3.1. Early related designs

Various anonymous communication systems utilised concepts from the original Mix-nets design, introducing certain modifications in order to address certain aspects of anonymous communication. Many of these systems were associated with large latencies associated with the Mix-nets architecture that made it impossible to use them in more time-sensitive scenarios such as web browsing.

For these systems, it can be mentioned, among others, Babel, an anonymous remailer described in 1996 [20], which aimed to quantify the dimensions of anonymity, provide a way of measuring it and enhance the security of the network. Another example was a paper about Stop-And-Go MIXes (SG-MIXes) [21] that introduced random delays in order to be more resilient to traffic analysis and remove the requirement of collecting a fixed-size batch of messages. ISDN-MIXes [22] is another Mix-net-based design from 1991 for untraceable communication for services such as telephony. One of its major advantages is the very small bandwidth overhead.

The first Mix-net-based solutions that were successfully implemented and gained significant recognition were anonymous remailers. In 2000, the Web MIXes system [23] was proposed as one of the first described solutions for Internet browsing using Mix-nets. The project was successfully implemented and gained recognition.

After initial Mix-net designs, where mixes were treated as specific servers to whom clients connect, propositions to make peer-to-peer designs occurred where each user also relayed traffic for others, acting as a mix in the network. Within this category, there are solutions such as MorphMix [24] or Tarzan [25].

3.3.2. Anonymous remailers

Anonymous remailers were the first real-life implementations of Chaum's ideas that gained significant recognition. There are three types of anonymous remailers, or four, if pseudonymous remailers that were described before are counted.

Cypherpunks remailers

The first anonymous remailers, also known as Type I remailers, were initially created by the Cypherpunk movement. Cypherpunk remailers can only be used for one-way communication, as there is no possibility for the recipient of the message to reply.

Cypherpunk remailers, as the whole Cypherpunk movement, attaches huge importance to the cryptographic aspect of communication; therefore, messages were encrypted in order to provide confidentiality, usually with the software from the PGP family.

Another important concept introduced with Type I remailers was chaining, which means that instead of one remailer that will be more than intermediate in the connection between the sender and

the receiver, eliminate the single point of failure issue and increase the anonymity of the sender, since the second and every next remailer in the path does not know anything about the sender's identity and the first remailer does not need to know the identity of the sender.

Contrary to the Penet remailer, Type I remailers do not keep a list of users or any activity logs. One of the greatest weaknesses of cypherpunk remailers is their vulnerability to traffic analysis, and therefore to correlating messages with the sender by the order of them.

Cypherpunk remailers gave the foundation for other types of anonymous remailers.

Mixmaster

Mixmaster, also known as Type II remailer, is another type of anonymous remailer that, similarly to Type I, allows for one-way anonymous communication, unless, of course, the sender himself includes a reply address in the message itself. It was created by Lance Cottrell around 1995 and later maintained by Len Sassaman and Peter Palfrader. Mixmaster operates on fixed-size packets, and one of the greatest improvements in comparison to the Type I remailer is message reordering (mixing) that prevents the possibility of traffic analysis. Mixmaster sends messages through intermediary nodes via the SMTP protocol.

Mixminion

Mixminion, also known as Type III remailer, further addresses the issues related to its predecessors. Similarly to Type II remailers, it uses fixed-size packets that are reordered and forwarded with appropriate encryption through a chain of servers called mixes with their public keys. A single mix cannot correlate the sender with the recipient.

In order to reply to messages, Mixminion uses single-use reply blocks, also called SURBs, that are a way of anonymous responses. They are hidden partial paths included in the message that give the possibility of responding without revealing the sender's identity. Each mix in the chain can unwrap its layer and encrypt the message.

The initial release of Mixminion was in December 2002, and one year later the paper describing the design was published [26].

3.3.3. JAP

JAP is an anonymous communication network known by many names: Web MIXes, Java Anon Proxy, JAP, JonDonym, JonDo. Despite the naming confusion, all of them refer to a proxy system designed to provide anonymity on the Web. The system was first described in 2000 in a paper "Web MIXes: A system for anonymous and unobservable Internet access" [23]. Later, the project was deployed as part of the AN.ON project of the Technische Universität Dresden, the Universität Regensburg, and Privacy Commissioner of the state of Schleswig-Holstein under the name Java Anon Proxy (JAP), which was directly taken from one of the system components. In 2007, the project diverged into

for-profit JonDonym (AN.ON substitute), and the proxy itself got the name JonDo (JAP substitute), and the entity responsible for the commercial branch development was JonDos GmbH. Regardless of these administrative and name changes, the system will be treated as one within this paper. It initially supported only HTTP and HTTPS traffic, although it was later expanded for other protocols, and had several interesting technical solutions that made it suitable for web browsing even though it utilised the high-latency architecture of Mix-nets.

Key components of the WebMIXes include:

- Java Anon Proxy client
- MIXes cascades
- Cache Proxy
- Information Service

Java Anon Proxy client is the client software written in Java, in order to achieve platform independence. It connects to the first MIX of the MIX network via the Internet. It periodically sets up dedicated bidirectional channels for data transfer. If a user wants to send a message, it is first transformed into the MIX format and passed through the MIX cascade. If a user wants to send a message smaller than a given size, it is padded to avoid volume correlation. On the other hand, if he wants to send a large message or streaming data, the message is split into several time slices and it is sent slice by slice with an appropriate Slice ID (SI). The idea of slicing was first introduced in the previously mentioned ISDN-MIXes. If there is nothing to send from a given client, a dummy message is sent. An adversary that wants to correlate a message with the specific sender is unable to do so as with equal probability it can be any of the active clients in a given moment and he is unable to tell the difference between the real message slice and a dummy message as they are both encrypted. As the channels are bidirectional, it is also responsible for handling the responses. Messages sent via JAP are also filtered with respect to potentially compromising content such as cookies and JavaScript. On top of that, JAP also uses the Information Service (info-service) in order to measure anonymity. Each user has a limited bandwidth and the number of time slices that are used concurrently. Each time he wants to send a slice, he has to provide a ticket. The ticket-based authentication system uses the blind signature algorithm invented by David Chaum [27] in order to preserve users' privacy. This authentication mechanism aims to prevent flooding and Denial of Service (DoS) attacks.

MIXes cascades are dedicated servers connected to each other through the Internet, although the authors of the article suggest that ideally they should be connected with dedicated high-speed connections instead, however, it would be too complex and expensive. The chain of MIXes that mediates between the sender and the receiver is called MIX-cascade. Similarly to the original Mix-net design, each MIX node strips a layer of encryption, creates a batch of messages, reorders the messages, and forwards the data to the next MIX. The last one in the cascade sends it to the cache-proxy. MIXes, similarly to JAP, also send dummy traffic to each other.

Cache Proxy is an element responsible for sending the data from the last MIX to a specific

web server requested by the user. It sends back the response through the same bidirectional channel, although in reverse order. It also loads all the objects embedded in the requested HTML page. This idea was first introduced in the Crowds project. Moreover, as the name suggests, it is responsible for caching the responses, which helps to reduce the number of calls to the Internet.

Information Service is a service that provides information about MIXes' addresses and public keys, current traffic and MIXes availability and the level of anonymity provided which is calculated taking into account the number of currently active users, which is directly associated with the possibility of an intersection attack. Each user can set a threshold of anonymity he wishes to maintain, and once the level falls below that threshold, he will be warned.

The WebMIXes project underlined the importance of the usability perspective and ease of configuration and use, as well as the portability of the JAP software so that it could have been run on as many diverse devices as possible. It also provides a built-in integration with web proxies that helps to work behind restrictive proxies, firewalls, et cetera. in order to provide better accessibility.

In JonDonym users also had the possibility of choosing Mix Cascades in order to avoid organisations that they do not trust. In the past, it used to be one of the more recognised ACNs, although as it is not functioning today, it cannot be considered within the comparative analysis.

3.3.4. Design considerations

During the evolution of Mix-nets, several important design considerations emerged, including the development of different mixing and batching strategies, message formats, and network topologies. These aspects played a crucial role in shaping the functionality and security of modern Mix-nets.

Mixing and batching strategies

Several mixing and batching strategies were introduced in various papers and projects. One of the first attempts to provide a formal framework for modelling mixing networks based on the mixing and batching strategy was proposed in 2003 [28]. It was based on the 2002 paper [29] in which certain types of mix networks were distinguished and categorised along with attacks on them. Those strategies included:

- threshold mix
- timed mix
- threshold or timed mix
- threshold and timed mix
- threshold pool mix
- timed pool mix
- timed dynamic-pool mix (Cottrell Mix)

Threshold mix is a strategy in which the mix collects a certain number of messages and then forwards them. In **timed mix** mixes forwards messages after a specific time period. With **threshold or**

timed mix, the mix either collects a certain number of messages and then forwards them or forwards messages after a specific time period. In case of **threshold and timed mix**, the mix collects a certain number of messages and forwards them after a specific time period. In the **threshold pool mix** when $n + f$ messages accumulate in the mix, where n is a threshold and f is a pool size, n randomly chosen messages are forwarded while the f messages are kept in the pool for the next iteration. In **timed pool mix** every specific period of time messages are sent, however a fixed number of random messages (a pool) are kept in the mix. Lastly, the **timed dynamic pool mix (Cottrell mix)** is similar to the timed pool mix combined with the threshold pool mix; however, only a certain fraction of the surplus messages are forwarded, after pool subtraction, every specific period of time.

Message format

Over the years, specific message formats for Mix-nets have been proposed to improve security, for example, by hiding metadata and routing information that helps break the linkability between incoming and outgoing messages or protecting against attacks such as the tagging attack where the attacker modifies messages by embedding tags that he wants to detect in other parts of the network in order to get routing information. Minx [30] was one of the first propositions to address these issues. The format uses RSA for the standard Mix-net encryption along with anonymous reply blocks, AES with IGE mode that helps to detect message manipulation, and, as the IGE mode propagates errors, destroys the modified message by changing the content into garbage. At the same time, the format imposed a low computational and communication overhead. Minx has a vulnerability that was discovered in a 2008 paper [31]. The paper, besides describing the vulnerability and a chosen ciphertext attack that exploits it, provides a fix for the Minx format that would address this issue by replacing explicit session key and next hop header fields by utilising hash functions.

Yet another message format, Sphinx, was proposed in 2009 [32]. The goal was to provide the most compact and yet fully secure scheme. Sphinx addressed issues of the Minx design while keeping all the desirable properties of it. One of the changes was using ElGamal instead of RSA. Sphinx also includes replay protection and formal proofs of security, which Minx lacked.

Topologies

There are several possible topologies in the Mix-nets that appeared throughout the years. These topologies were distinguished and analysed in terms of anonymity and overhead in 2010 [33]. The first topology that appeared in the original Chaum 1981 paper was a cascade topology in which the mixes create a simple path where each node is connected only with its predecessor and successor (excluding the first and last nodes). Due to the unlinkability property of Mix-nets, the cascade can be fixed, and there is no need of permutations of paths as in the onion routing that will be described later. The drawback of this topology is its poor scalability and availability, as each mix in the path is a potential point of failure and there is a high risk of bottlenecks. Instead of a single cascade, there is also a

possibility of a topology where there are many cascades - it is called multi-cascade topology.

One of the alternate topologies is a fully connected network, where each node is connected to any other node in the network. It fixes the availability issue as there are many potential routes from one mix to the other. It does not fix the issue of scalability, as adding a new node implicates the necessity of adding a connection to every other node in the network. The drawback of this topology is the fact that messages cannot be freely mixed as there are several possible directions for a message from the node.

Another possible topology, mostly used in the latest Mix-nets, is a stratified topology. Each node is assigned to a certain layer and is connected to every other node from the previous layer and with every other node in the next layer; however, the direction of the messages in the network is fixed. With this topology, scalability and availability are improved, and all messages in the node can be mixed together regardless of their origin and destination, contrary to the free route design.

3.3.5. Loopix

The development of Mix-net-based systems continues to this day, although it is definitely less popular than onion routing-based solutions. One of the newer propositions in the MIX world is the Loopix system, proposed in the 2017 paper [15]. It used ideas from Stop-and-Go MIXes, creating a simplified version of it called the Poisson mix. The messages do not use synchronous rounds, as with some different Mix-net designs; they are asynchronously and nondeterministically delayed using an exponential distribution. Messages arriving at a given entity within the network follow the Poisson distribution. As a solution for a smaller user base and increased security, cover traffic is introduced between entities in the network, including loop traffic. Loop traffic, whose concept is similar to the 2003 paper [34], where a user sends messages to himself, helps to detect active attacks and blocking of messages and, along with drop cover traffic, aims to hide communication patterns. Loopix provides a modifiable trade-off between anonymity and usability by changing message delays and the real to cover traffic ratio. Loopix uses providers to manage access and store offline messages. These are semi-trusted nodes that users connect through. The Loopix mixes create a stratified topology, which means that they are grouped in layers, and each node is connected to every node from the previous layer and every node from the next layer. When a user wants to send a message, he randomly chooses a node from each layer. The messages sent on the Loopix network follow the Sphinx format.

Loopix opened a new chapter in mix networks as it serves as a basis for other Mix-nets that are created to this day, Nym [16] being an example of such a network.

3.3.6. Nym

The Nym network was described in 2021 [16]. The network is based on the Mix-nets architecture and heavily utilises concepts from the Loopix network and can be considered an implementation of Loopix, enhanced with economic incentives. Mixes on the network are rewarded with NYM tokens

using a mechanism called "proof of mixing". The tokens provide economic incentives which are not present in other volunteer-based networks and aim to encourage more people to host mix nodes. One of the main goals of the network is to introduce a global adversary-proof solution that is not present in solutions such as Tor. In addition to a decentralised Mix-net, Nym includes an anonymous credential cryptosystem.

Nym network architecture consists of the following elements:

- mix nodes
- gateways
- service providers
- Nym blockchain
- nympool
- validators

Mix nodes are nodes that create a network called Mix-net that relays, decrypts its specific layer of encryption, reorders and delays messages asynchronously as in the Loopix network. **Gateways** are entry points for the Nym network. Users can choose whether they want to always use the same gateway or not. Similarly to the Loopix design, they are also able to store messages for users who are offline. **Service providers** are entities that provide certain services through the Nym network. They receive cover traffic that is indistinguishable from real user traffic, which enhances user privacy. **Nym blockchain** is a decentralised blockchain structure, maintained by validators, that distributes network-wide information regarding public node reachability and public keys, configuration parameters and other publicly available data that are necessary for proper functioning of the Nym network. **Nympool** is a shared pool of funds supporting the Nym network. Periodically validators algorithmically distribute rewards from the nympool to the nodes (validators, gateways, mixes) and to the service providers that redeem users' service credentials. **Validators** maintain the Nym blockchain, including nympool, as well as issue bandwidth and service credentials that prove the allowance of sending data through the Mix-net and the right to access a certain service, respectively.

The flow of users connecting to the service provider in the Nym network works as follows: A user deposits his NYM tokens, earned by operating a node, purchased or received from a service provider, in the nympool and obtains credentials with the help of the validators. These credentials cannot be directly linked to their use due to blind signatures. With the bandwidth credential, the user can access the gateway of his choice. The gateway validates the token on the Nym blockchain and submits a commitment of the credential to the validators in order to avoid double-spending. The user and the gateway establish a shared secret for data encryption and a pseudonym associated with the credential that allows for tracking bandwidth consumption as well as for offline storage. From now on, until the bandwidth credential expiration or consumption, the user can use the Nym Mix-net through the gateway. Although the gateway has knowledge of when the user is accessing the network, it is not aware of where his messages are sent. The message from the user is sent to the recipient's gateway,

from which the recipient can pull the messages. The recipient is either a service provider, a validator, or another user. If the recipient is a service provider that provides a paid service, a service credential is needed to access the service. The logic behind the handling of the service credentials is similar to that of the bandwidth credential.

3.3.7. Other designs

In recent years, other Mix-net designs have appeared. Katzenpost [35] being one such example, aims to fix certain issues associated with the Loopix design. It puts a great emphasis on implementing post-quantum cryptography algorithms into the mix network, as well as changing a point of failure in the form of a single service provider into scattered distribution across multiple providers, similarly to a distributed hash table form. The project is still in development. Recently, the design was described in the Echomix paper [36], where Katzenpost is defined as the Echomix implementation.

Besides Loopix-based designs, there are also unrelated recent approaches to the topic of Mix-nets, including Vuvuzela [37] that aimed to minimise metadata related to the communication process, Riffle [38] that aimed to decrease bandwidth and computation costs for the sake of file sharing and microblogging, or cMix [39] that reduces public key cryptography-related latency and the overhead for the client by utilising precomputation. The cMix uses a threshold and a timed mixing strategy, which may potentially still introduce significant latency, especially for small numbers of users. Moreover, fixed cascade topology suffers from the issues described in the topology subsection, for example, reduced availability. The paper suggests a multi-cascade approach as a potential answer for this issue.

3.4. DC-nets

DC-nets are anonymous communication networks based on the dining cryptographers (DC) problem described by David Chaum in his 1988 paper [10]. Despite the name, the problem is not related to the dining philosophers problem. It considers the problem of secure multiparty computation of the boolean XOR function.

The problem can be described as follows: assume that three cryptographers are dining together. The waiter informs them that their meal has already been paid for by someone but does not reveal the identity of the payer. It is assumed to be one of the dining cryptographers or the National Security Agency (NSA). In order to determine whether it was one of them who paid or the NSA, they have to perform a certain two-stage protocol.

In the first stage, every pair of cryptographers establishes a one-bit secret. For three dining cryptographers, there are three such secrets: AB, AC, BC. In the second stage, each cryptographer who did not pay announces the result of an XOR operation on the secrets he shares with his two neighbours. On the other hand, if a certain cryptographer paid, he announces the opposite of this XOR operation.

After three announcements are revealed, it can be determined whether it was one of the cryptographers who paid or was it NSA. This can be done by performing another XOR operation on

these three bits. If the result is one, it means that it was one of the cryptographers who paid for the meal. If the result is zero, it means that none of the cryptographers paid; therefore, the NSA paid.

Figure 3.2 illustrates two scenarios in the dining cryptographers protocol: the case where none of the cryptographers pays for the meal and the case in which cryptographer *A* pays.

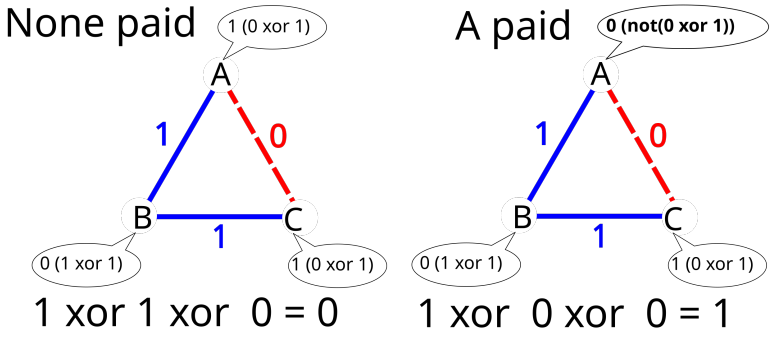

Fig. 3.2. Illustration of the dining cryptographers problem. By PauAmma, Wikimedia Commons [40], licensed under CC BY-SA 3.0

There are several limitations to the dining cryptographers problem. Firstly, only one cryptographer can pay for the meal. Secondly, each cryptographer is trustworthy, which means he always presents information that is true. Thirdly, each cryptographer must share a secret with every other participant, which means that there will be a total of $n * (n-1)/2$ connections, creating a complete graph.

The proposed protocol to solve the dining cryptographers problem can be easily converted to anonymous communication of longer messages. While sending messages bit-by-bit might be problematic, it can be replaced with fixed-size blocks, and the sender, instead of performing a negation of the XOR operation he performed on the secrets he shares with his neighbours, performs an XOR of these secrets with his message. The message is broadcasted by design; however, certain improvements can be made in order to provide confidentiality and authenticity - public-key cryptography can be utilised where each participant can have a public key associated with a pseudonym. The sender can include a digital signature in his message to prove the authenticity of the message. Moreover, in order to provide confidentiality (or secrecy), he can encrypt this message with the public key of the recipient to whom he wants his message to be transferred. Each network member will receive the encrypted message; however, only the recipient, the owner of the private key, will be able to decrypt it.

Chaum also proposes other potential improvements to the design, such as slot reservation. The original DC-Nets design was intended to demonstrate the dining cryptographers problem and sketch potential implementations of them. However, they cannot be used in the original form in real use cases due to the issues associated with this simple design. Similarly to Chaum's Mix-Nets, his DC-Nets introduced a new branch of ACNs, creating a basis for future work of other researchers.

3.4.1. Related designs

Among systems or propositions based on DC networks, which similarly to Mix-net-based solutions were meant to address some issues of the original design, it is worth mentioning The Dining Cryptographers in the Disco protocol [41] and CliqueNet [42], a peer-to-peer, practical, self-organising, resilient, scalable, and anonymous communication network that was proposed in late 2001 and aimed at fighting surveillance from solutions like Carnivore [43]. CliqueNet evolved into the Herbivore [44] network; however, the goals remained mostly the same. Another example of DC-net-based design with efficiency improvements and multicast focus was described in the XOR-Trees paper [45]. A notable paper on improvements to the DC-net design - among others in terms of jamming detection, efficient fault recovery, and performance - was proposed in "Dining Cryptographers Revisited" [46]. Dissent (Dining-cryptographers Shuffled-Send Network) [47] aimed to address disruptions caused by DoS or Sybil attacks by introducing a slot distribution mechanism through responsible shuffling. Its follow-up paper was titled "Dissent in Numbers: Making Strong Anonymity Scale" [48] addressed scalability issues with the original Dissent design through the introduction of client-server architecture, where each client trusts that at least one server is honest. Riposte [49] further enhances this concept with Private Information Retrieval via Distributed Point Function, which reduces the high bandwidth cost associated with DC-nets-based designs. At the time of writing this thesis, there is no widely deployed and fully operational DC-net-based solution.

3.5. Anonymous publication systems

Anonymous publication systems are a category of systems that have the specific purpose of anonymous publishing of content, which is usually joined with the goal of censorship-resistance. They can be either created on top of an existing anonymous communication network or have dedicated mechanisms. As these networks are not universal in terms of networking and have a specific purpose, they will not be considered within the comparison; although they are worth mentioning, as they had an important role in the development of ACNs that are known today.

One of the first proposals for such a system was The Eternity Service [50]. It was a proposal for a storage system with long-term availability of files in it. It assumed utilising redundancy and scattering and emphasised the importance of the anonymity of servers that were serving content. For anonymous communication utilising Chaum's Mix-nets or DC-nets was proposed. Adam Back, a British cryptographer, implemented the Eternity Service [51] that was inspired by the original article and its goal, although the technical design was different.

Another proposal for anonymous WWW publishing was described in the paper "TAZ servers and the Rewebber network: Enabling anonymous publishing on the world wide web" [52]. The authors underlined the importance of anonymous publishers throughout the history of publication. They provided a more practical solution to the issue of anonymous publishing. Rewebber acts as a proxy when a certain file is accessed and the user sees only the Rewebber domain, the path is encrypted, therefore no

information about the TAZ server location is revealed. The Rewebber forwards the request to one of the TAZ servers. The data in the system are stored on TAZ servers in a scattered way. The Rewebber network can provide availability in the event of one of the TAZ servers' failure.

A similar system was proposed by The Free Haven project [53]. It aimed to provide anonymity for publishers, readers, and senders, accountability without sacrificing anonymity, persistence, and robustness of storage, and flexibility of peers joining and leaving the network.

One more example of a system in this category was the Publius [54]. Nine goals of the network were: censorship resistance, tamper evidence (unauthorised changes can be traced), anonymous source, updatability, deniability (third parties that participate in the publishing are not responsible for the content itself), fault tolerance, persistence, extensibility, and free availability. The system was implemented and opened for a two-month trial; however, it did not gain enough prominence to continue development.

A further example is Tangler [55], a censorship-resistant system that employs a storage mechanism where newly published documents depend on the previously published documents, similarly to the blockchain where each block depends on the previous. Publication of new documents is associated with replication of previously published ones. The authors name this dependency an entanglement.

Today, the most recognisable systems from this category are Freenet and GNUnet; although GNUnet greatly evolved compared to the original file storage functionality.

3.5.1. Freenet

Freenet was created in 1999 as a university project of Ian Clarke, who studied at the University of Edinburgh. The paper describing the network was published in 2001 [56]. The goal was to create a fully decentralised, peer-to-peer system for censorship-resistant and private communication and publishing. Freenet was further developed and maintained by a group of internet freedom activists. It has gained great popularity, as it has been downloaded more than two million times since the release of the system.

In March 2023, Locutus was renamed to Freenet, and the project that up until that point worked under the name Freenet was renamed to Hyphanet. For the sake of clarity, this paper will use the names Freenet and Hyphanet interchangeably and Locutus when talking about the new network.

The main goal of Hyphanet is to provide a way to anonymously share files, browse, and publish Hyphanet websites, and chat on forums without censorship. The project emphasised the importance of freedom of speech and the fact that if some institution has the ability to control the information that people have access to, then the institution can influence people's opinion.

Each participant contributes a portion of their disk space and bandwidth to the network, forming a distributed data store. When a user wishes to publish content, the data is encrypted, divided into chunks, and inserted into Hyphanet along with a cryptographic key. These chunks are distributed across participating nodes based on availability and demand, without any central authority. To retrieve content, a user requests the appropriate key, which is propagated through the network via a series of

nodes; each node checks its local cache and forwards the request closer to the data's likely location. As data is retrieved and passed back through the network, nodes cache content fragments, optimising future access, and improving resilience against node loss or censorship. A typical request is presented in 3.3.

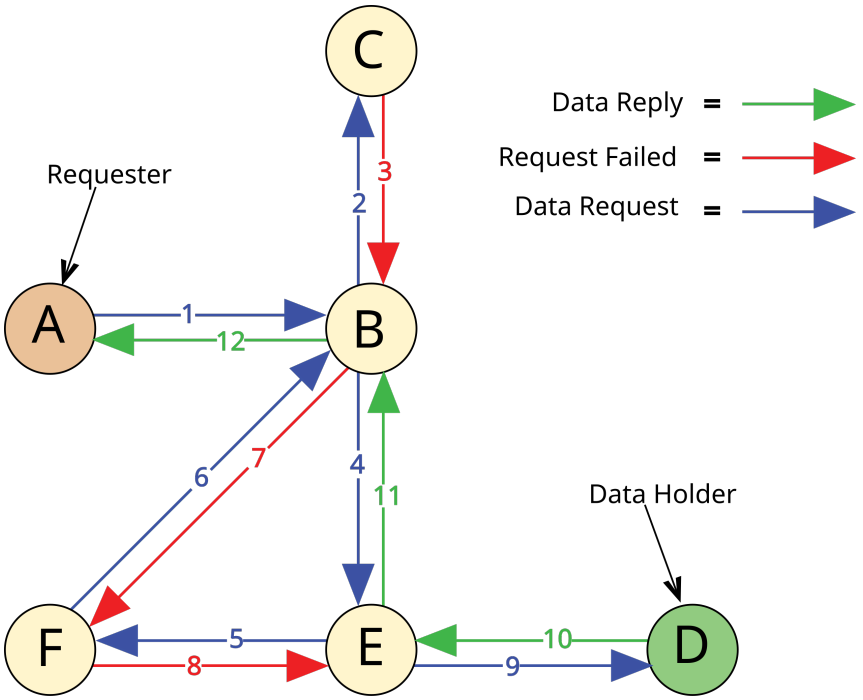

Fig. 3.3. Request sequence in the Hyphanet (Freenet) network. By XcepticZP, Wikimedia Commons [57], licensed under CC BY-SA 3.0.

Hyphanet supports both *opennet* and *darknet* operating modes. In opennet, nodes automatically connect to random peers. In darknet mode, connections are established manually between trusted friends, providing additional resistance to traffic analysis and surveillance.

Routing in Hyphanet is adaptive; nodes maintain routing tables that indicate which peers are most likely to have certain keys, and these tables are refined through continual use. The combination of strong encryption, distributed storage, and dynamic routing mechanisms ensures anonymity for both publishers and retrievers, while making it extremely difficult for adversaries to censor content or trace information flow within the network.

Hyphanet therefore aims to allow two or more people to share information freely, without government or other entity control. At the same time, Hyphanet assumed that it is impossible to have freedom of speech without being anonymous because it is easier to punish those who proclaim things that are not appealing to the censor rather than preventing them from doing so. Hyphanet also proposed solutions to the problem of trust in the anonymous system.

The goal of Locutus [58] is slightly different from the goal of Hyphanet; it aims to provide a decentralised alternative to the Internet in terms of computing, not only in terms of storage, as it was with Hyphanet.

It poses a threat of data exploitation or censorship and therefore can violate individual freedom. Locutus is proposed as a solution to these problems.

It works as a shared computing platform. While Hyphanet can be compared to a shared disk, Locutus can be compared to a shared computer.

Locutus can be reached using standard web browsers as well as with an API. It proposes interesting solutions regarding many problems of today's Internet, like users' privacy, spam, or DDoS attacks. Locutus is designed to be a platform on top of which developers can build highly interoperable, seamlessly integrated, scalable and cryptographically secured decentralised services that can be an alternative to the centralised ones that are present now.

3.5.2. GNUnet

GNUnet started in 2001 and its main goal then was to improve Freenet file sharing. The initial name for the system was GNET as GNU had not yet approved the project back then. The principle of operation was described in the paper with the same name [59], describing a reliable and anonymous distributed backup system. After approval a few months after the release, the project was renamed to GNUnet as it had been intended since inception. The communication mechanism in the GNET on the lowest layer was similar to SSH, although it was based on UDP, not TCP, as it did not need guarantees of proper order or packet loss and preferred to utilise lower-overhead solution. The project also described a reputation system that affected the ability of a node to interact with the network to prevent various attacks and protect the network. Instead of a global credibility system, each node had an "opinion" on other nodes that it contacted that decreased with every quest and increased for the valid replies. Queries were encrypted and hashed by the client so that the intermediaries could not decrypt the content. The hashes could have been combined with logical operators in order to increase searchability over systems like Freenet. Users can decide whether to use keywords of shorter length and improve usability or of longer length and increase security. The data stored on the GNET nodes were split into chunks, and each chunk was encrypted by default and stored separately with hashes as chunk identifiers. In an alternative configuration, the data can be shared in plaintext and encrypted only in transfer or, in the maximum-security configuration, XORed with a one-time pad, which results in doubling traffic/storage requirements.

The project has greatly evolved through the years, and now GNUnet is a framework for P2P networking in a decentralised and privacy-preserving manner. It aims to serve as an alternative for the traditional TCP/IP stack, creating many of the networking solutions from the ground up, addressing their major flaws from the point of view of privacy, freedom and decentralisation and at the same time providing excellent plug-in-based compatibility with existing infrastructure. As anonymity is no longer the main focus of the project, it will not be considered within the comparison.

3.6. Onion routing

While Chaum's Mix-nets provided a satisfactory level of anonymity, most of the solutions based on them introduced significant latencies and could not be used for latency-vulnerable use cases. One of the first proposals that aimed to address this issue was the ISDN-MIXes network, although it was mainly focused on the ISDN network. A different and more universal approach was PipeNet [60], a protocol for low-latency anonymous communication. It was based on a network of nodes, identified by their public keys, that communicated asynchronously with each other over secure links, creating a mesh network topology. The main drawback of the network was that it assumed fully utilised links, which are expensive.

Another proposition was the UDP-based Freedom System [61]. The primary goal of Freedom was to create a system with the strongest privacy, ease of use, and integration. Freedom used a circuit-based solution named the telescope encryption. The solution that turned out to be the most used for low-latency anonymity communication was onion routing.

In 1995 the U.S. Naval Research Lab (NRL) researchers started working on the first prototypes of onion routing. The first paper on Onion Routing was released in 1996; however, the final version was first published in 1997 and a year later, in 1998, it was published in the IEEE Journal on Selected Areas in Communication [11].

Onion routing enables anonymous connections that are highly resistant to surveillance, tracking, and traffic analysis. Onion routing provides a near real-time bidirectional connection similar to the TCP socket connections. Regular Internet applications can access anonymous connections via proxies, which additionally can remove identifying information from the data stream, and therefore make the communication anonymous.

It is the major difference between the Mix-net and the Onion routing as the mixes are not connection-based. It has several implications; for example, onion routing does not batch the traffic and the message reordering is highly limited, which may make it more vulnerable to certain traffic analysis attacks compared to the classic Mix-nets. However, the versatility of onion routing coupled with a better user experience through smaller latencies made this solution a dominant one today. Applications that initialise communication make it through a few machines called onion routers. Onion routing was designed to be application independent; therefore, a wide variety of applications could have been used. The onion routing network allows the initiator and the responder to establish a connection between them. These connections are called anonymous socket connections, or simpler, anonymous connections. They hide who is connected to whom and why. If one wishes to go a step further and provide anonymity to the entities themselves, identifying information must be removed from the data stream before it is sent over the anonymous connection.

The basic configuration of the onion routing topology assumes that there is an onion router that sits on the firewall of the sensitive site. There is an assumption that connections within the sensitive site are protected by, for example, physical security, therefore it can also be called a secure

site. Sensitive sites can be only on the initiator end or on both ends of the connection. The onion router at the edge of the sensitive site serves as an interface between machines within the secure site (behind the firewall) and the external network like the Internet. To avoid traffic analysis, the onion router that sits on the edge of the initiator should route its traffic through other onion routers within the onion network.

If both sites control onion routers, then they can successfully hide their communication from eavesdroppers. On the other hand, if the responder, for example, the web server, has an unprotected connection between him and the last onion router, then the data stream from the initiator must be anonymised. Otherwise, analysis of the unprotected link can reveal the identity of the initiator.

The onion routers within the network are connected through permanent socket connections. Anonymous connections over the network are multiplexed over these permanent connections. For any given anonymous connection, the route in the path is precisely defined at the connection initiation; however, each router in this path is only aware of the previous and the next hop in the route. The data transmitted through the onion network is modified at every onion router in order to prevent data tracking and ensuring that compromised onion routers cannot collaborate by correlating the data stream they observe.

Onion routing connections by design have three phases: connection setup, data movement, and connection teardown. In the first phase, an initiating application establishes a socket connection to the proxy specific to an application on an onion router, which intermediates its connection to the onion routing network. The proxy then determines a path through the onion network and creates a layered data structure known as an onion and sends it through the network. Each layer specifies the next hop in the route; when an onion router receives the onion, it removes the layer specific to itself, identifies the next hop, and forwards the remaining onion to the next onion router. According to the design, each onion router should pad the onion to maintain a fixed size. After sending the onion, the proxy on the initiator's site can send data through the established anonymous connection. The last onion router in the path forwards data to the proxy on the same machine, called the responder's proxy which forwards data to the responder. Each onion layer includes properties of the connection at each point in the path, for example, cryptographic algorithms that will be used in the data movement phase or key seed material necessary to generate symmetric keys that will be used to encrypt the data sent forward (direction in which the onion travels) or backward (opposite to the onion travel direction) along the anonymous connection. 1999 design assumes usage of different algorithms and keys for the data that moves backward compared to the one moving forward. Symmetric cryptography is used for this purpose as it has significantly less computational overhead than expensive public key cryptography, therefore, it is more suitable for near real-time connections.

As mentioned, after establishing the connection by sending the onion in the initialisation phase, the connection is ready to perform the data movement phase. Sender's onion router will add a layer of encryption for each onion router in the path. Each onion router in the route will remove one layer

of encryption and the last onion router will have a plaintext, which it will pass to the receiver. For the response, the reverse order of these layers is used. That is, the last router in the path will add the first layer of encryption, and the first router will add the last one. Then the sender will receive the response encrypted with multiple layers, which he can remove in order to obtain the plaintext.

The 1999 paper describes that all information, including the onion from the initialisation phase, network control information, as well as the data that is being sent in the data movement phase, is sent in uniform-sized cells. All cells that the onion router receives within a fixed time interval are mixed together in order to reduce the correlation of the network data. For the purpose of foiling external observers, padding and bandwidth limiting can be used. Due to the fact that each onion and each data phase cell look different from each onion router, it is significantly harder for attackers to perform traffic analysis.

Although the name "onion routing" may suggest that it has something to do with the third (network) layer of the ISO/OSI model, the operation logic occurs in the seventh (application) layer. It is an example of an overlay network, so it refers only to the logical connections, not the physical ones. Moreover, there is a significant flexibility when it comes to the protocols of the lower layers. The authors describe anonymous connections through onion routing as protocol-independent.

Onion routing turned out to be a breakthrough in the anonymous communication network field and many designs to this day implement this concept. The best known example of such a network is Tor, the most popular ACN today, which will be described in a dedicated section.

Onion routing introduced an idea of reply onions, which idea is similar to Mix-nets' reply blocks. It is a way to connect with the originator in other ways than using an initial bidirectional connection with him, for example, for the sake of asynchronous communication.

3.6.1. Tor

History

In the beginning of the 2000s Roger Dingledine and Paul Syverson, shortly later joined by Nick Mathewson, worked on an implementation of onion routing that could be used for everyone, not just the US Navy. It was designed to work in a decentralised network, run by people and institutions with varied interests and levels of trust. Roger Dingledine named the Tor project, which was an abbreviation for The Onion Routing, to differentiate between other onion routing-related projects that were developed at that time.

In 2002 Tor network was deployed and its code was released under a free and open software license in order to provide more transparency to the project.

In 2004 a paper "Tor: The Second-Generation Onion Router" [13] was published, which described how Tor works. In the same year, the Electronic Frontier Foundation (EFF) began to fund the work on Tor.

In 2006 Tor Project Inc, a non-profit organisation was created in order to maintain development

of the Tor network. Today, Tor is funded from several sources, including volunteers, foundations, and institutions.

Tor stood the test of time and turned out to be an exceptionally useful, free, and easily accessible tool against censorship, surveillance, and much more. Today, the Tor network has thousands of voluntary run relays and millions of users, and it is undoubtedly the most popular anonymous communication network in use.

Modifications compared to the original onion routing design

Tor introduces a few modifications to the original onion routing design. Costly, slow and compromise-prone public key cryptography with long-living keys was replaced with short-living symmetric keys, achieving a perfect forward secrecy, as it will be impossible to decrypt captured traffic once the keys are deleted with the circuit teardown. Separate, dedicated proxies for every application were replaced with a universal SOCKS proxy that supports most TCP-based applications, and for the HTTP and HTTPS Tor relies on Torbutton, an add-on for Firefox, and hardened version of the Mozilla Firefox, creating a Tor Browser Bundle. The ideas of mixing, padding, and traffic shaping were rejected in favour of performance improvements and experience quality. Many TCP streams are multiplexed within a single circuit, which improves both efficiency and anonymity, as building many circuits can present a threat to anonymity, which will be discussed later. Tor proposed a leaky-pipe circuit topology, meaning that a client can communicate with all nodes in the circuit, which in theory may make traffic analysis more difficult as the data are not always going through the entire circuit. Congestion control was achieved through end-to-end acknowledgments that detect congestion. Problematic state flooding was replaced with another mechanism, directory servers; certain nodes, considered as more trusted, provide a list of known routers and their state in order for the user to be able to build a circuit. Initially, such a request was made in HTTP; however, it was later exploited by censoring regimes that were able to block the Tor network for users, and therefore the connection between the user and the directory servers started to use HTTPS. More information about the history of blocking Tor across the world will be described in a dedicated section. Tor nodes can advertise exit policies that describe hosts and ports to which a node is allowed to connect. End-to-end acknowledgments were introduced, enhancing security and preventing certain types of attack, for example, tagging attack. Reply onions were replaced with a more sophisticated mechanism of hidden services that allows a connection to a server whose location is not known. In order to connect with a hidden service, clients negotiate rendezvous points.

Initially Tor did not try to hide the fact that a certain user is using the network - it changed over time when regimes started to censor Tor and it was necessary to fight back - the censorship resistance became another goal of the Tor network. Today, Tor circumvents censorship with bridges that are not publicly listed relays, along with pluggable transports, a way of masking traffic.

The visualisation in Figure 3.4 illustrates onion routing using the example of Tor. The bidirectional connections between Tor nodes, as well as the link between the Tor client and the entry

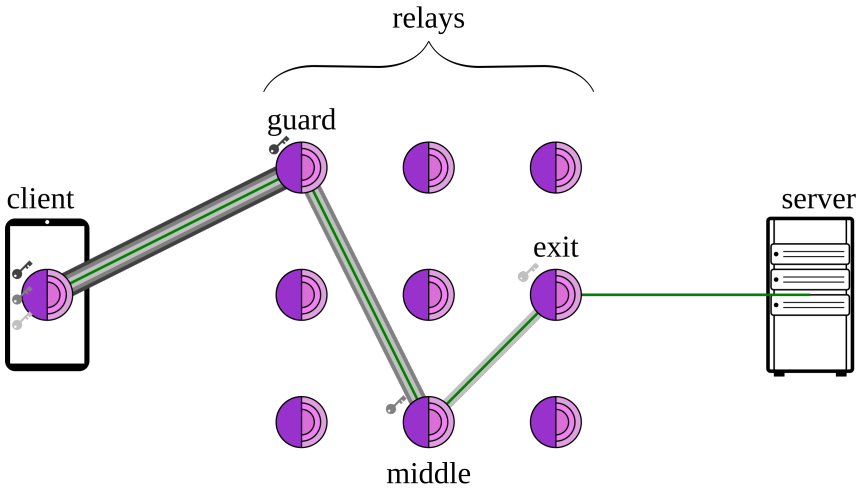

Fig. 3.4. Tor circuit diagram by Justin Tracey (Wikimedia: Tga.D), originally published in [62]. Available on Wikimedia Commons [63], licensed under CC BY-SA 4.0.

guard, are encrypted using symmetric cryptography of Tor; however, the link between the exit node and the destination server is not.

Design and architecture overview

Tor does not protect against a strong global passive adversary and, therefore, it is not secure against end-to-end traffic correlation attacks. It can protect against local passive adversaries that observe a certain fraction of a network, against generation, modification, deletion, and delay of traffic or against operating or compromising some number of onion routers. The main reason for sacrificing stronger anonymity is to achieve a compromise between anonymity, usability, and efficiency.

The project places great emphasis on the deployability by enabling inexpensive and easy setup, portability by supporting various operating systems, expandability for future improvements, and simplicity for ease of understanding and analysis. The main goal of these actions was to increase usability: the more people will use Tor, the larger anonymity set will be, and therefore the harder it will be to identify a single user.

Tor uses a client-server architecture. Each onion router (OR) in the onion network runs a non-root process which allows it to maintain a TLS connectivity with every onion router in the network. Users run a process called an onion proxy (OP) that allows them to establish circuits in the network, manage user connections, and fetch directories. Multiple TCP streams from onion proxies are multiplexed in a single circuit. Onion routers maintain two types of keys: short-term onion keys and long-term identity keys. Identity keys are used for TLS certificate signatures, signing a structure called a router descriptor, which is a summary of router details such as address, bandwidth, policies, keys, et cetera. On top of that, directory servers use identity keys to sign directories. Onion keys, on the other hand, are used in circuit setup and in symmetric ephemeral key negotiation. TLS protocol also establishes a short-term key during onion routers' intercommunication. Short-term keys are rotated periodically and, in order

to reduce a potential key compromise impact, independently.

Initially, Tor design assumed random choice of nodes in the circuit. This approach turned out to be problematic for various reasons. The first one was the fact that it created bottlenecks as each node had the same number of circuits as any other node and therefore the low-bandwidth ones were overloaded, the high-bandwidth ones were underutilised. The solution for this was to weight a node based on this bandwidth in a way that the high-bandwidth nodes get more circuits than the low-bandwidth ones, as well as to balance based on the capabilities of the node as nodes that can be only the middle node would be chosen three times less frequent compared to a node that can be either the entry, middle, or exit node, which would also lead to bottlenecks. To prevent abuse by a potential attacker, who would advertise an impossibly high bandwidth in order to have a dangerously high probability of choosing him as a node in the circuit, Tor uses measured bandwidth instead of an advertised one. Another improvement made with nodes' choice was related to guard nodes. It is assumed that it is sufficient for the attacker to deanonymise the user if the attacker is controlling both the entry and the exit node. Considering the fact that the circuits are created rather frequently, the probability of deanonymisation in a longer period of time was exceptionally high. Guard nodes are an answer to this issue - instead of choosing three new nodes each time the circuit is created, the entry node is kept for a longer period of time. With this approach, the likelihood of deanonymisation is not increasing as rapidly with time, unless the malicious node is chosen for the guard, then the probability is even higher. Moreover, to decrease the likelihood of controlling both entry and exit nodes at the same time, additional diversity requirements were introduced for nodes in the circuit that prevent the creation of a circuit with two routers with the same network range or family.

Cells, circuits and streams

Tor uses fixed-size cells of 512 bytes for communication over TLS connections between onion routers and onion proxies. The cell includes the identifier of a given connection, the command field that specifies the purpose of a cell's payload, and the payload itself. There are two types of cells: control cells, interpreted by the receiving node, and relay cells, responsible for the end-to-end data stream. Relay cells have additional header fields compared to control cells, including: TCP stream identifier, end-to-end checksum, payload length, and a specific relay command. The relay header is encrypted along with the payload of the relay cell and decrypted along the circuit.

In Tor, a circuit can be shared among multiple TCP streams. Moreover, to avoid correlation between the Onion Proxy and the stream, a new circuit is being established periodically in the background, when the previous circuit is being used. Circuits that do not have open streams are closed. Every minute Onion Proxy considers switching the circuit to the new one.

The Onion Proxy negotiates symmetric keys with every node in the circuit incrementally. It will first establish an encrypted connection with the first (guard) node, then through the first node with the middle relay, and eventually, through the guard and the middle relay, with the third (exit)

node. Establishment is done via certain cell commands and the Diffie-Hellman key-exchange algorithm is used. It is important to mention that there is one-side entity authentication here, meaning that the Onion Proxy will know to which Onion Router it connects to, but not the other way around. Two symmetric keys are derived from each established key: one for the forward traffic and the second one for the backward one.

After establishing the circuit, the relay cells can be sent. After receiving such a cell, the Onion Router checks the correlated circuit and decrypts the header and payload with the session key. It checks the integrity of the message by verifying the digest. In order to gain computational optimisation, hash computation is often omitted, and the first two bytes of the hash are zero. After verifying the message integrity, it either processes the message or checks the circuit ID (circID) and OR for the next step of the circuit and replaces circID. If the exit node receives an unrecognised relay, it raises an error and tears down the circuit. When Onion Proxy receives the relay response cell it unwraps the relay header and the payload with the session keys that it shares with every Onion Router in the circuit, checking the integrity with every unwrapping. The initial Tor TLS handshake was a straightforward mechanism that included providing supported cryptographic algorithms and parameters by the initiator and choosing them by the responder, as well as providing a two-element certificate chain that proved the authenticity of the Tor node. This mechanism had to be later modified due to censoring Tor using deep packet inspection in a way that would make it appear like the usual HTTPS web traffic. Streams utilise SOCKS protocol in order to be able to connect the Onion Proxy to the given address and port. OP either creates a new proxy or uses the newest created open circuit. From that circuit it chooses one node to be an exit node, depending on the exit policy, usually the last node.

Due to several flaws with SOCKS in the initial Tor design that could have led, for example, to a DNS leak, usage of privacy-aware HTTP proxies was suggested. Later, it was addressed by the Tor Browser bundle.

Similarly to closing TCP streams, Tor uses a two-step handshake for normal closing, which is especially useful for half-closed connections, and a one-step handshake for errors.

Figure 3.5 presents a sequence diagram that describes the typical circuit configuration process in standard Tor usage. Initially, the user requests a signed list of relays (the directory document) from the directory servers and receives this information in response. Using the directory document, the user selects three relays to form a circuit. The circuit is then established using the telescope approach, where encrypted channels are incrementally opened through each relay, one at a time. Once the circuit is fully established, it enables secure bidirectional communication between the user and destination until the circuit teardown.

QoS

Tor includes QoS mechanisms: congestion control, fairness, and rate limiting. **Congestion control** in Tor is achieved at several levels. Firstly, it uses TCP along with its mechanisms of congestion

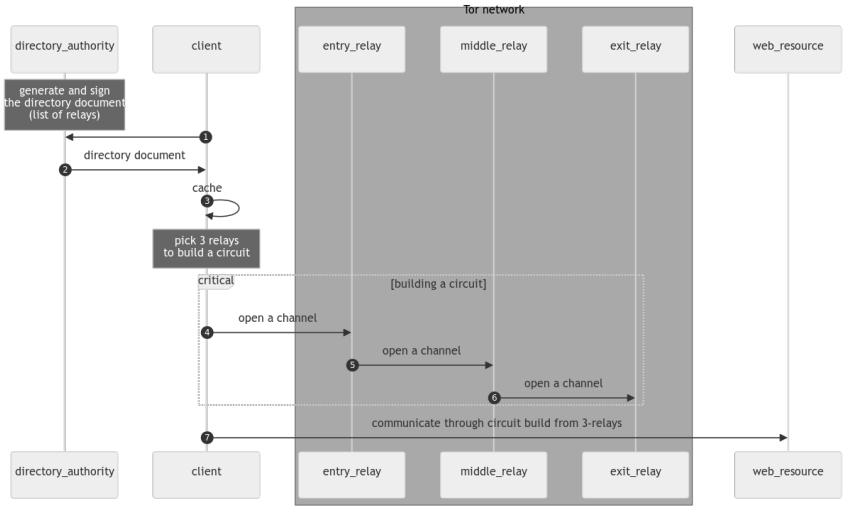

Fig. 3.5. Tor circuit setup process. Source: The Tor Project specification [64], licensed under CC BY 4.0.

control and in-order guarantees. Secondly, circuit-level throttling to control the bandwidth within the circuit, where each onion router keeps a track of two windows for incoming and outgoing traffic. Thirdly, an end-to-end stream-level throttling is introduced, similar to the circuit one, however, utilising a single window. To limit bandwidth usage and provide **fairness and rate limiting**, Tor uses a token bucket algorithm, which helps to achieve a long-term average rate of bandwidth but at the same time allows for infrequent sudden spikes. Moreover, streams can be categorised based on the frequency of cells' appearance in order to provide a better latency for interactive streams. The risk of an end-to-end attack in this case is not an additional threat as Tor is already vulnerable to these types of attack.

Onion Services

Tor can provide anonymity not only for the sender, which was described above, but also for the server itself (responder anonymity). It can be achieved thanks to Tor's onion services, previously known as hidden services or location hidden services. The server anonymity can have many benefits, which will be described in the use cases and application areas section.

The anonymous bidirectional connection is established as follows: First, the onion service creates a long-term key pair that will identify the service. Secondly, it chooses several onion routers as its introduction points (IPs), establishes a circuit with each of the IPs, providing each of them his public key, and advertises them in the lookup service which has a form of a distributed hash table, a hidden service directory (HSDir). Then it signs the advertisement, also known as the hidden service description, with its private key. Optionally, a part of the descriptor that includes the introduction points can be encrypted with a key that is shared between the onion service and the authorised users. With this approach, the unauthorised users will not only not be able to connect to the service, but will also be unable to discover it in the first place. If a user wants to communicate with the onion service, he looks up in the DHT for one of the introduction points of the hidden service, chooses an OR as his rendezvous point (RP), establishes a connection with both the RP and the IP, and then announces

the RP to the OS through the IP, along with a rendezvous cookie, the first part of a DH handshake, and optionally with additional information, encrypting everything with the onion service's public key. Once onion service receives this message it can connect with the rendezvous point, of course through intermediate nodes, and pass the rendezvous cookie, joined by the second part of the DH handshake and a hash of the session key they are now sharing. Despite the fact that the RP connects the circuits of the user and the onion service, it is not aware of the communication between them. Neither are the IPs of onion services. Once the user sends the control cell along the circuit, anonymous bidirectional communication can be performed. Such a separation of functionalities on multiple routers provides smear resistance, while several introduction points provide robustness and higher availability. There is also a possibility of access control associated with optional authorisation tokens that a user can provide.

From 2015, .onion domains are recognised as special use domains by the IESG and can obtain TLS certificates [65]. There are two companies that support issuing X.509 certificates for the .onion top level domain. The first is DigiCert; they support Extended Validation (EV) TLS certificates, wildcards (which is an exception as EV in general does not support wildcards [66]) and validity period up to 15 months. The second one is HARICA - they support Domain Validation (DV) certificates. Although it may sound redundant, as onion services already seem to provide benefits of TLS in terms of end-to-end encryption and authentication, there can be several benefits from using TLS certificates. Firstly, the mixing of HTTP and HTTPS can cause a leak of sensitive information [67]. Browsers enable additional security and privacy improvements when accessing HTTPS websites related to cookies and sensitive data. Moreover, certain software solutions require TLS by default and allow only HTTPS. Additionally, users are taught to look for HTTPS when connecting to a website in order to make sure that it is a safe connection, and changing their habits might be challenging. Furthermore, if the Tor client is located in a different place than the web server itself, the connection between the Tor client and web server needs to be additionally encrypted; in the case of using HTTPS for the onion service, the problem is solved.

Finally, an important and practical feature of onion services is that they work reliably even when the server is behind a Network Address Translation (NAT) or firewall. This is possible due to the way onion services use hole punching techniques: the service initiates all outbound connections to introduction and rendezvous points, meaning it does not need to accept inbound connections directly. As a result, no port forwarding or firewall configuration is necessary, which significantly simplifies the deployment and increases accessibility for less technical users.

Bridges

Tor Bridges are Tor relays that are not publicly listed as standard nodes. They often offer additional functionalities, such as traffic obfuscation. Bridges were introduced in response to the censorship of traditional nodes or directory authorities that are easy to block due to their public availability, and the Tor project in advance planned for this. The idea was to simply encourage users from less censored parts of the world to offer themselves as private relays that will not be publicly listed

for the people in censored countries. Bridges can be distributed directly from the Tor website [68], via e-mails from bridges@torproject.org upon request (currently it is required to send the request from a Gmail or Riseup email address) or by other means; any person can set up a bridge relay and privately share it with whom he wants.

Pluggable transports

Pluggable transports are modules that obfuscate traffic, so anyone monitoring the network activity of a user running Tor with a pluggable transport will see traffic that does not look like typical Tor usage. There are several types of pluggable transports (PTs):

1. Obfsproxy
2. Meek
3. Snowflake
4. WebTunnel

Obfsproxy is the first pluggable transport, created in 2012, encrypts the traffic so that no specific headers are recognisable. In this case censors can either block all unrecognisable traffic, which will result in a massive number of false positive cases, or allow it, which is usually done. **Meek** is a pluggable transport created in 2014, based on domain fronting through Amazon Web Services (AWS), Microsoft Azure or Google Cloud Platform (GCP). The basic idea behind it is to route user traffic to a cloud provider and from within the cloud reach the Tor network. **Snowflake** is yet another pluggable transport that works as a JavaScript code executed in the volunteers' browsers. The client connects to the Snowflake proxy of the volunteer through WebRTC that obfuscates his traffic, making it appear as, for example, a video call. The major advantage of this kind of pluggable transport is the simplicity of deployment without the need of dedicated software other than browser extension, although there are also dedicated standalone snowflake proxies aimed at long-term connections. The more people join as a snowflake proxy, the more difficult it will become to censor traffic without a large number of false positives. **WebTunnel**, officially announced in 2024, is a new kind of bridge that aims to circumvent censorship in heavily censored countries. Using an HTTPS proxy, similar to HTTPT [69]. Contrary to Obfsproxy, it aims to mimic a real web server, rather than appearing as unrecognisable traffic.

3.6.2. I2P

History

The idea for I2P started in October 2001 when Lence James, known under the pseudonym 0x90, created IIP - the Invisible IRC Project as a way for instant communication with Freenet users regarding Freenet issues and for Freenet keys exchange. In 2003 it was decided that IRC was not sufficient for the project to serve its purpose. Instead of such an approach, a universal anonymisation layer was needed for a wider variety of protocols. From then on, IIP was also referred to as InvisibleNet.

In the same year an anonymous developer jrandom joined the project and his ambition was to

expand it. He wanted to redesign the system with inspiration from Tor and Freenet projects, but at the same time many things specific to the I2P were introduced. Strong emphasis was placed on protection from organisations with enormous resources. In the same year, jrandom took control over the project, and it was renamed to the name that exists today, Invisible Internet Project, or I2P for short. Two years later, in 2005, another prominent anonymous developer joined the project - zzz, shortly after joined by yet another developer with a pseudonym Complication. In 2007 zzz with Complication took control over the project after jrandom unexpectedly quit, which caused turbulence for the project development.

Over the years, several developers joined and left the project, and many improvements were introduced. The project is to this date fully based on volunteers' work. The project used to accept donations but they were discontinued. Today, I2P is considered one of the most popular anonymous communication networks; currently, it has around 55,000 routers, according to I2P metrics [70, 71]. The number of users may be slightly larger since users from restricted countries may not act as routers.

Design and architecture overview

I2P is an overlay peer-to-peer network with full encryption and resilience against censorship, eavesdropping, and recognition. Every user, unless it is unsafe for him, contributes to the network as a router, increasing the network anonymity, creating a peer-to-peer network. Also, being a router or, in other words, participating in the I2P network is not necessarily something that should be kept as a secret, while information about connections with endpoints associated with specific applications called destinations is.

I2P utilises tunnels, encryption (both router-to-router and end-to-end), provides forward secrecy for all connections and self-authenticated network addresses.

The network consists, among others, of routers/peers communicating with each other, unidirectional tunnels (one for incoming traffic, one for outgoing traffic, contrary to the Tor bidirectional tunnels) and a distributed hash table as an internal database for information distribution. These and other elements of the I2P architecture will be described in detail in this section.

In order to connect to the I2P network, the user needs to install a Java client consisting of a static website template, email, and a BitTorrent client. As mentioned above, it also works as a router.

I2P offers a complete suite of network protocols and serves as a platform on top of which applications can be built. These applications include newsgroups, email, messaging, file sharing, web browsing, blogging, or distributed data storage. One of the main advantages of the I2P network is its scalability. The more people use the network, the faster it is as there are more routers in the network and more traffic is being passed by them. It is also becoming more anonymous with an increasing number of users as there is much more traffic to pass with for a particular router. Due to the peer-to-peer nature, creating exit points would impose a threat of legal issues for the users, given illegal actions of some other user that would have them as the exit point. Users might not be willing to take this risk, which would definitely reduce the number of users willing to use the network. Due to this fact, along

with security issues that are associated with exit points, I2P does not support exit points by default; however, it is possible to create such a proxy (outproxy) on a higher layer.

Routers and destinations

The I2P router is an entity that participates in routing the traffic of users. It has the same role as the nodes in the Tor network; however, contrary to Tor, each user that uses the I2P also routes traffic to others. Although being a router or running the I2P software is currently not a secret, it is possible that in the future, when I2P will gain more popularity and governments will try to censor it, it will become necessary to add a masking layer to circumvent the censorship. When a user first connects to the network, he connects to special-purpose routers called reseed hosts, that provide an initial set of nodes for a new router to connect with.

Destinations are cryptographic identifiers of applications within the network. The fact that a user has reached a given destination is a secret. In addition, the user who reached the destination remains anonymous.

Network database

The network database, netDb, is a distributed hash table (DHT) database based on the Kademlia algorithm that is responsible for the sharing of metadata on the network. The database is maintained by a subset of routers called floodfill routers. They are normal routers with high bandwidth, which constitute about 6% of the network. There are two types of metadata.

The first is **leaseSets**. It gives the instructions necessary to reach the particular destinations. LeaseSets also include a number of leases, where each lease specifies a tunnel gateway that is necessary to reach a particular destination. Lease information includes a pair of public keys for message encryption, tunnel expiration time, and recipient inbound gateway. LeaseSets are distributed through outbound tunnels to avoid correlation between leaseSets and their router and to remain the router's identity. There is a possibility of private encrypted leaseSets for the purpose of reaching services that must not be publicly available. Such encrypted leaseSets are not published on the netDb and require appropriate decryption keys.

The second is **routerInfo**; it gives the instructions to reach the specific router. These instructions include a unique ID in the form of a cryptographic public key, transport addresses and ports, timestamps, and other options that can be utilised by the user, everything digitally signed and the signature is also attached. RouterInfo is distributed by the routers directly to netDb, as this information should not be a secret.

Figure 3.6 illustrates the use of inbound and outbound tunnels for Alice and Bob within the I2P network. For simplicity, Charlie's inbound tunnel and Dave's outbound tunnel are omitted from the diagram. The network database contains the information necessary to allow participants to discover and reach each other's inbound tunnels.

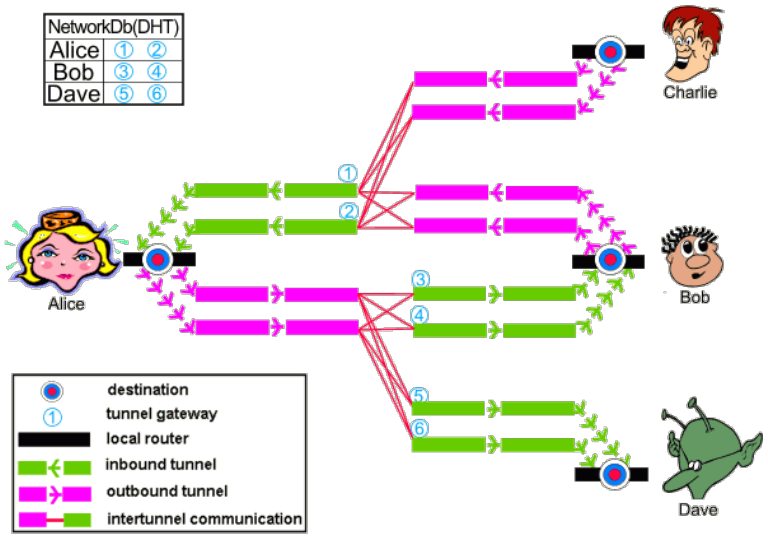

Fig. 3.6. Simplified I2P network topology. Source: I2P documentation [72], licensed under CC BY-SA 4.0.

Tunnels and garlic routing

A tunnel is a short-living, direct path between two points underlaid by several routers between, where layered encryption is utilised. Each router can decrypt (or "peel") the encryption layer and forward the message to the next router. The first router on the path, the starting point of the tunnel, is called a gateway. The tunnel ends with an endpoint. Tunnels are only one-way, and for responses an additional tunnel is needed. That is why there are two types of tunnels in I2P that are required for two-way communication: inbound and outbound. Inbound tunnels, as the name suggests, bring messages to the tunnel initiator and outbound tunnels send messages from the tunnel initiator. The sender's outbound endpoint will pass the message to the recipient's inbound gateway, given the appropriate instructions to forward the message. Using two one-way tunnels instead of one two-way tunnel makes traffic analysis harder for the adversary.

In order to build inbound and outbound tunnels, a user needs to call netDb and fetch a list of peers that can be used as hops in these tunnels. Then users can gradually, router after router, build the tunnel through construction messages. The user sends the message to the first hop, then to the second hop with intermediation of the first router, then third through the first and second until the tunnel is successfully formed. When the sender wants to send a message, he first needs to find the receiver leaseSets in netDb to retrieve the receiver inbound tunnel gateways. The sender then has to choose one of his outbound tunnels and sends the message through this tunnel along with the information to pass the message to one of the receiver's inbound tunnel gateways. In summary, the path of the message is as follows: First, it is sent through the sender's outbound tunnel, and then, thanks to appropriate instructions, it is forwarded to the receiver's inbound tunnel. In case the sender would like to receive a response, he will need to send his destination within the message. The sender can also optionally include

his leaseSets in the message so that the receiver does not need to look it up in the netDb. Despite layered encryption in tunnels that was described before, additional end-to-end encryption needs to be used to protect messages that leave outbound tunnels and enter inbound ones. End-to-end encryption is performed with public-key cryptography.

Messages can be grouped together and form what is called a garlic. Each message in the garlic can be named a garlic clove. Garlic cloves can accumulate in the inbound gateway, pass through the tunnel, and split at the endpoint of the inbound tunnel. That mechanism is possible due to unidirectional tunnels and would not be possible in classic onion routing bidirectional tunnels. It is often referred to as garlic routing. Garlic contains information about the tunnel that will be used for message sending, making route establishment much faster.

Although I2P initially proposed the introduction of batching, mixing, throttling, padding, and delaying traffic, it is currently not implemented as parallel support of two types of traffic can be problematic. If it will be in the future, the network should then be considered as a hybrid solution between Mix-Nets and onion routing.

Routers are chosen for the tunnel based on various metrics, including their bandwidth, capacity measured by successfully built tunnels, which is the direct indicator of their reliability, family diversity, as well as manually adjustable individual preferences.

Figure 3.7 illustrates the multiple layers of cryptography applied to messages in the I2P network, effectively providing three layers of encryption: on the link level, tunnel level, and end-to-end.

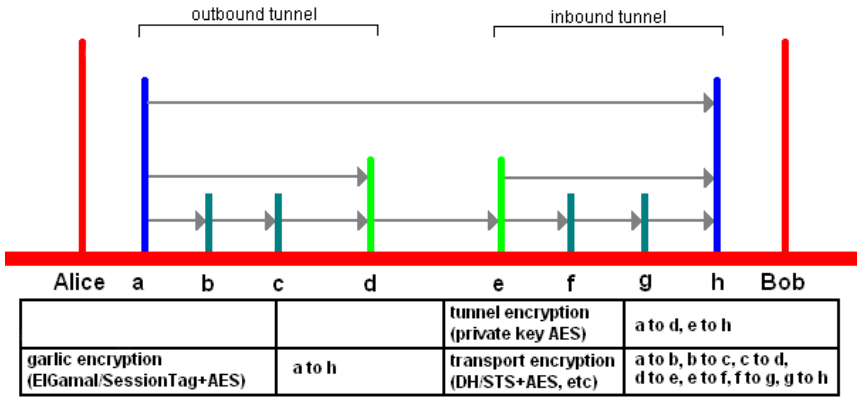

Fig. 3.7. Message encryption in I2P. Source: I2P documentation [73], licensed under CC BY-SA 4.0.

I2P protocol stack

The Invisible Internet Project proposes its protocol stack on top of which applications can be built. For analysis purposes, references to the OSI and TCP/IP stacks will be made. In theory, the TCP/IP transport layer and all lower layers are also a part of the stack; however, the overview will focus solely on the I2P-specific stack. Beginning from the bottom of I2P stack:

1. I2P Transport Layer
   (a) SSU2: Secure Semi-reliable UDP

   (b) NTCP2: NIO-based TCP
2. I2P Tunnel Layer
3. I2P Garlic Layer

The **I2P Transport Layer**, in the context of the TCP/IP and ISO/OSI stacks, as well as each layer after it, could be placed in the application layer. However, this specific layer can also be treated as an overlay on the traditional transport layer. This layer is responsible for providing encrypted connections between the I2P routers. These are not anonymous end-to-end connections but simple, not anonymous, hop-by-hop connections. In the I2P Transport Layer there are two protocols that are based on UDP and TCP: **SSU2** (Secure Semi-reliable UDP) and **NTCP2** (NIO-based TCP). In the future, this layer can also provide additional obfuscation that will make it harder for censors to block I2P traffic. The **I2P Tunnel Layer** is the layer that provides tunnel connections with full encryption. Protocol data units used within the I2P Tunnel Layer are tunnel messages and I2NP (I2P Network Protocol) messages. I2NP are messages used to pass information through multiple routers. Multiple I2NP messages, layered and encrypted along with instructions necessary for their delivery, are combined into a larger tunnel message which has a fixed size of 1024 bytes. Each hop in the path removes (decrypts) its layer, reads an instruction, and forwards the message to another router. The **I2P Garlic Layer** provides anonymous and encrypted end-to-end message delivery. Similarly to the tunnel layer, the I2P Garlic Layer uses I2NP messages. They are wrapped in layers to ensure anonymity for the sender as well as for the receiver. Garlic messages expand standard onion messages by allowing the message to contain multiple cloves. A clove is a fully formed message with delivery instructions.

The mentioned layers are considered as I2P core layers. There are also additional layers that are considered non-core layers:

1. I2P Client Layer
2. I2P End-to-End Transport Layer
   (a) Streaming library
   (b) Datagram library
3. I2P Application Interface Layer
4. I2P Application Proxy Layer
5. I2P Application Layer

The **I2P Client Layer** provides the ability to use I2P without direct router API usage. **I2CP**, or **I2P Client Protocol**, is a protocol of this layer. It provides asynchronous and secure communication by sending messages over the I2CP TCP socket between a client and a router. The **I2P End-to-End Transport Layer** provides functionalities similar to TCP and UDP from the TCP/IP transport layer, but over the I2P network. Two prominent solutions that exist within this layer are the **streaming library** (functionality similar to TCP streams over the I2P network) and the **datagram library** (messaging similar to UDP over the I2P network). The **I2P Application Interface Layer** provides libraries that make development on top of I2P easier. The protocols of this layer use the protocols of the **I2P End-**

to-End Transport Layer. Example protocols include **SAMv3**, which supports multiple communication types, including reliable in-order TCP-like streams, UDP-like datagrams, or unauthenticated raw datagrams with minimal overhead. It supports many programming languages as well and can be used to create native I2P applications. Another example is **I2PTunnel**, which is used to create general purpose TCP-based connections, allowing seamless proxying of existing non-I2P applications into the I2P network without the need to write I2P-specific code. The **I2P Application Proxy Layer** is used for proxy systems. Examples of protocols for this layer include SOCKS, HTTP, and IRC. Finally, the **I2P Application Layer** is a common name for the applications that can be built on top of the I2P stack.

I2P applications

There are many I2P applications, some of which are included in the standard I2P install bundle. One example of a bundled application is the **naming library and address book** - a web-of-trust-based, local naming system that allows users to add specific long I2P addresses as human-readable entries in their address book. Another included application is **I2PSnark**, a BitTorrent client that works entirely within the I2P network. While it is not the only BitTorrent client available for I2P, it is the default choice shipped with the install bundle. Typically, it is not possible to connect to non-I2P torrents from within I2P, or vice versa. Additionally, the bundle includes **I2Pmail**, also known as susimail, which is an internal and external email service operating on the I2P network.

There is also the possibility of creating custom applications and distributing them as I2P plug-ins which can then be easily integrated with I2P routers from other users. Users can access the I2P router API for example via a simple JSONRPC2 interface. The I2P network allows services to be hosted anonymously in the I2P network. These destinations, for example, websites, called Eepsites, are subject to the i2p pseudo-top level domain. The domain name is resolved into an I2P peer key by a proxy named EepProxy. Service providers can use solutions like I2PTunnel to share basic client-server applications in I2P, it also includes support for SOCKS proxies. On the other hand, if an application is peer-to-peer, requires a more sophisticated approach, enhanced anonymity, or performance optimisation, they can write I2P-specific code and achieve tailored solutions. There is also the possibility of embedding I2P into an application.

I2P services and applications generally work without issues, even when the user is behind a firewall or NAT. In such cases, the I2P router uses a mechanism called introducers to relay inbound connections, allowing communication without requiring open ports. However, forwarding the I2P port for both UDP and TCP can improve performance, speed, and overall integration with the network.

3.6.3. Lokinet

Lokinet is a relatively recent decentralised anonymous network that utilises onion routing architecture. The network was described in the 2018 paper [14]. It is one of the first networks to

introduce economic incentives for node runners. Lokinet introduces Low Latency Anonymous Routing Protocol (LLARP) for communication within the network. It is said to be a hybrid solution between Tor and I2P. In fact, the design is more similar to the Tor one, including client-server architecture; however, Lokinet, contrary to Tor, does not use semi-centralised directory authorities. Instead, it relies on DHT, similar to I2P, although in Lokinet DHT is built on blockchain. Moreover, LLARP is not restricted to TCP-only traffic like Tor is. Lokinet has formal support for exit nodes, similarly to Tor and contrary to I2P, which requires dedicated outproxies. Similarly to Tor and I2P, Lokinet supports services within the network - SNapps - that work analogously to Tor's onion services or the services within I2P. Lokinet also introduces a message storage concept, for users that are temporarily offline, that is not present in Tor nor in I2P. SNapps also use NAT/hole punching and opening firewall ports is not necessary.

However, the biggest difference is the placement in the ISO/OSI model; while Tor and I2P are built on top of the transport layer, Lokinet is built within the network layer and can support any IP-based protocol.

The infrastructure of Lokinet underlines a popular instant messaging app - Session, although Session is temporarily switched from utilising Lokinet directly for the sake of simplified Onion Requests due to compatibility issues on various platforms [74], however, since the infrastructure is the same, the Session user base directly increases the anonymity set of Lokinet. Moreover, the return to Lokinet is planned when compatibility issues are solved. The return is needed as low-latency properties of Lokinet will allow for anonymous voice and video calls, while the current simple Onion Requests mechanism is not able to provide such functionality.

3.6.4. Other designs

There are also newer proposals that aim to address issues with the original onion routing design or with the Tor network, as it is considered a direct successor to the onion routing design. Examples of such networks include HORNET (high-speed onion routing at the Network Layer) [75], TARANET (Traffic-Analysis Resistant Anonymity at the Network Layer) [76] and Dovetail [77]. They introduced potential improvements in terms of latency or traffic analysis; although they require changes in the network layer, they would be difficult to perform in existing networks built on top of the transport layer, such as Tor or I2P.

3.7. Classification

The classification of anonymous communication networks is shown in 3.8. The tree structure presents the main categories of ACNs that have been developed over the years, highlighting the technical diversity.

At the top of the diagram are all anonymous communication networks. The next level distinguishes five main families: anonymous publication systems, onion routing, Mix-nets, and single-hop proxies. Among these, single-hop proxies are shown with a dashed line. This is to underline

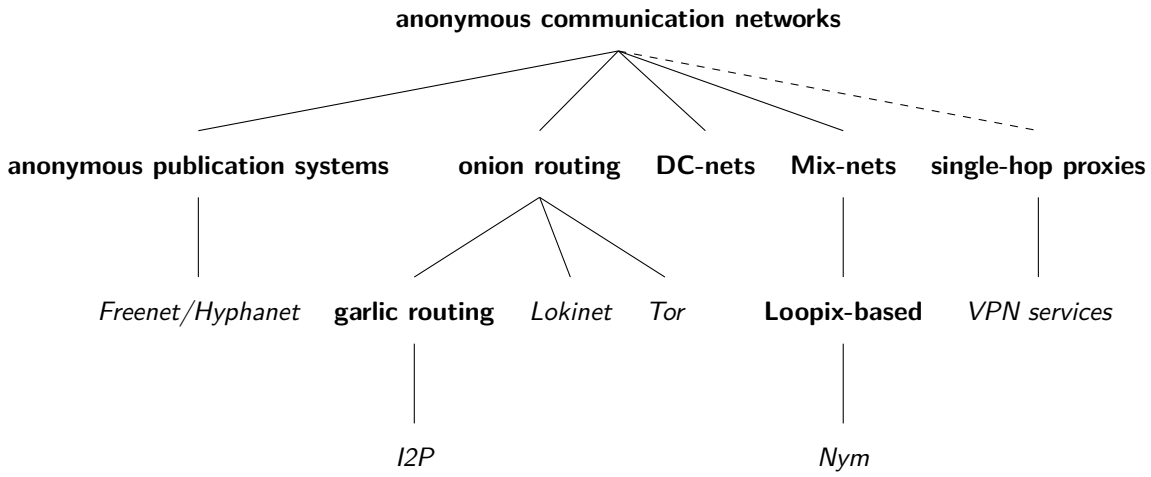

Fig. 3.8. Classification of anonymous communication networks

that, unlike other branches, single-hop proxies do not provide anonymity by design, but only by policy. Because of this, they cannot be considered proper ACNs in the strict sense.

Anonymous publication systems (such as Freenet/Hyphanet) are also present in the classification. Although they can provide strong anonymity, their main purpose is very narrow and is strictly based on anonymous publishing and distributed, censorship-resistant storage, and they cannot serve as general-purpose universal networks. For this reason, anonymous publication systems are not included in the comparative analysis in the following chapters.

Another important family included in the classification are DC-nets. Although DC-nets have strong theoretical anonymity properties, they are not widely used in practice today due to scalability, performance limitations, and significant bandwidth overhead. However, their unique approach has inspired many later designs and improvements in the field of anonymous communication.

The other two branches represent the main technical directions in the development of the ACN. Mix-nets (and their modern descendant, Loopix-based networks such as Nym) and onion routing (including classic Tor, Lokinet, and garlic routing solutions such as I2P) are the most influential approaches, each with their own properties and typical use cases. These designs are the most important for understanding practical anonymity online today.

Because the goal of this work is to compare fully functioning and practically relevant ACNs, the following chapters focus on four representative and currently active networks: Tor, I2P, Lokinet, and Nym. These systems illustrate the main architectures and techniques found in modern anonymous communication, and the analysis will show how their design choices affect their strengths, weaknesses, and best application areas.

3.8. Summary

This chapter has presented an overview of the main technical designs, historical milestones, and representative systems in the area of anonymous communication networks. By covering everything

from the earliest single-hop proxies and foundational concepts like Mix-nets, DC-nets, and onion routing, through to the systems used today such as Tor, I2P, Lokinet, and Nym, it has highlighted both the diversity and the progression of technical solutions in the field.

The technical background provided here is not just historical context; it is essential for understanding the strengths, weaknesses, and practical implications of different ACN architectures. In it is crucial for the comparative analysis carried out later in the thesis. The following chapters build on this material directly.

4. RELATED WORK

This chapter reviews the most relevant related work on anonymous communication networks in terms of comparative studies and practical analyses. The goal is to provide a clear overview of how previous research has approached the analysis and comparison of ACNs, which topics have been covered in depth, and where significant gaps remain, especially in terms of real-world usage scenarios and the inclusion of modern systems.

Including this section is essential to show the current state of research in this field. By understanding what has already been analysed and where the literature falls short, it becomes possible to justify the need for the new, scenario-driven comparative analysis presented in later chapters. The structure of this chapter is straightforward: it overviews existing literature and highlights why an updated and use-case-oriented comparison is still needed.

The existing literature on the analysis of anonymous communication networks often concentrates on traffic classification techniques [78, 79, 80, 81], rather than practical usage in various use cases or application areas. Although some studies provide comparative analyses of anonymous networks, they often lack a clearly defined goal or context for their comparison [82]. Other works address usage aspects [83, 84], but these papers provide only general guidance rather than conducting a comparative analysis based on specific use case scenarios.

Furthermore, most of these papers include analysis of legacy systems such as JAP and do not compare more recent network designs such as Lokinet or Nym. As ACNs evolve, older studies become less representative in terms of current performance or development status.

In summary, although the literature provides valuable technical analysis of anonymous communication networks, it primarily focuses on aspects like traffic classification and tends to overlook practical, scenario-driven comparisons that reflect how these systems are actually used. With the introduction of new networks such as Lokinet and Nym and the decline of legacy solutions, there is a clear need for up-to-date comparative analysis that considers concrete usage scenarios. Without this perspective, users might not realise that they are not using the best tool for their specific needs, which can lead to suboptimal choices and missed opportunities to improve privacy or efficiency. For these reasons, conducting a comparative analysis rooted in usage scenarios is crucial, not only to fill a gap in the research but also to help users make better informed decisions when choosing an anonymous communication network for a given use case.

5. USE CASES AND APPLICATION AREAS

This chapter presents various use cases and application areas for anonymous communication networks (ACNs). It groups these use cases based on similar requirements and establishes criteria for evaluating ACNs in each group. This classification is crucial, as it forms the foundation for later analysis and comparison of ACNs in terms of usability between different use case groups.

5.1. Identifying use cases and application areas

Identifying the practical applications of anonymous communication networks (ACNs) is essential to understanding their relevance and effectiveness in real-world scenarios. In this section, these use cases and application areas will be identified.

In restricted countries, ACNs can be used to **circumvent Internet censorship**, to be able to browse it freely. Among the most prominent examples of censoring countries are China and its Great Firewall [85], Iran and its extensive history of Internet censorship, along with other Middle East countries. Censorship in the Arab World especially intensified during the Arab Spring, where Tor played a crucial role as a tool that was used to bypass Internet restrictions [86]. There are also less known but equally severe examples of Internet censorships, for example, in Africa [87]. Freedom House is an organisation that performs periodic measurements of Internet censorship throughout the world [88]. One may think that in order for the user to reach the website, he would simply need to use an ACN and visit it, circumventing the censorship. While it may be true in some cases, in others it is a little more complicated. There may be some built-in security features that are non-compliant with ACNs nature. For example: If someone is accessing a website on the clearnet from the ACN, he would from time to time change his circuit, which was described before. This implies the fact that the exit node will change, and therefore the location from which the user visits the website. While in the ACNs such a location change is normal, it may seem suspicious for a web service as it may be associated with a hacker attack. The example of location issues described above was a real case of Facebook and an issue that legitimate users faced when reaching it via the Tor network. They could also count how many people use Facebook through Tor. In April 2016, less than two years after the first introduction of the onion service, which took place in October 2014, the number of users per month passed one million [89]. Not only did Facebook create an onion service, they also used single hops. As it is not a secret what is the real address of Facebook's servers, they reduced the number of nodes (from the server side) necessary for end-to-end user-server communication, reducing delays [90]. In addition, Facebook provided an SSL certificate for their onion address. The benefits of such action were provided before.

Censorship is not the only issue in web browsing. Another prominent one is surveillance, and ACN can provide **surveillance protection in web browsing**; therefore, protect its users against monitoring their web activities. An external observer, i.e. an ISP, will only see that a specific user is

using the ACN, unless the additional obfuscation is provided. In that case, the ISP can see various masking traffic, unrelated to the real actions of the user. While protection against censorship is challenging, protection against surveillance is even more difficult. In case of censorship the feedback loop is straightforward: certain websites are either reachable or blocked. Surveillance, on the other hand, is much harder to detect, and users are usually unaware that they are being monitored.

Getting unbiased results is crucial when it comes to research in order to ensure access to diverse, accurate, and not manipulated information, free from personalisation or censorship. ACNs help in **unbiased research** by masking user identity and location, preventing search engines from tailoring results based on user's past behaviour, geography, or profile data, therefore promoting objectivity and neutrality in search outcomes.

ACNs that support clearnet connections can serve to **circumvent geo-restrictions in the Web** tools in terms of circumventing geo-blocking or simply virtually changing location for various reasons. VPNs are tools that are often used for this use case. However, as was described before, there is a major difference between these two: VPN providers in most cases are single-hop proxies and provide the so-called privacy by policy, which means that a user has to trust some entity with his privacy and hope that they will stick to their policies, while the ACN privacy by design approach ensures that if any entity within the network or outside of it wants to reveal the user's identity, it is not able to do that. Another benefit is the fact that many ACNs are open-source and free. In the case of VPN providers they are usually paid services, and if a user from a restricted country is purchasing such a service from within that country it may draw the attention of the censoring government [91]. On the other hand, if the VPN service is free and at the same time it is not funded by volunteers and various organisations - it is highly likely that such a company sells the data of its users as running infrastructure costs money.

One of the lesser known advantages of hidden services is the fact that they can be used for **remote machine access**. Moreover, thanks to NAT punching the port forwarding can be avoided, which would not be possible in case of using clearnet alternatives that require opening an appropriate port. In other words, a hidden service can be set on any machine behind a firewall to reach it regardless of open ports, at the same time providing additional security benefits.

The Internet of Things (IoT) refers to the concept of connecting everyday physical objects to the Internet so that they can collect, share, and exchange data. These objects can be anything from household appliances and wearable devices to industrial machinery and cars. Each device can communicate with other devices or systems over the Internet or other networks. IoT security is crucial, especially when it comes to critical infrastructure. What is even more concerning is the fact that IoT manufacturers often underestimate the role of security in their devices, resulting in lack of data encryption or insufficient updates. Considering the fact that these devices are often exposed to the Internet without proper authorisation or confidentiality, a conclusion can be drawn that the problem is in fact exceptionally serious. **Utilising ACNs with IoT** can provide enhanced security as devices are not exposed to the internet, the fact of communication between devices or between devices and

computing centers can be hidden.

A use case that is increasingly popular is **using ACNs to run cryptocurrency nodes**. Nodes can communicate within the network, protecting their real IP address. Due to such an operation, it is impossible to censor these nodes nor to shut them down as their location is not known. Moreover, utilisation of hidden services can make the node setup easier and safer, as no ports on a firewall need to be forwarded. Examples of cryptocurrencies that support running nodes over ACNs are Bitcoin, the most popular cryptocurrency, or Monero, widely recognised as the most private of the cryptocurrencies. Their nodes can be run within clearnet, the Tor network, the I2P network, or any combination of them. In order for the nodes running within ACNs to connect to the clearnet nodes, they need to use bridge nodes that support both ways of communication. Moreover, users who use wallets over ACNs can anonymously connect to ACN-specific nodes, and the fact that the communication between these parties is hidden, as well as their identities behind the onion addresses. An additional benefit is related to the hole punching, as there is no need to open the ports necessary to communicate with the nodes.

Web-based cryptocurrency wallets can leverage hidden services to improve privacy and security for their users thanks to the security properties that come with them. An additional benefit from using hidden service for such use cases is that there is no risk from the exit node as the traffic never exits from the network. Malicious exit nodes could, for example, manipulate the cryptocurrency address in such a way that the cryptocurrency would be transferred to the attacker instead of the legit receiver. Of course, such malicious activity of the exit node would most likely be recognised by the ACN security mechanisms; however, it is better to prevent even the tiny risk of such an event, especially considering the fact of irreversibility of the cryptocurrency transactions and potential losses.

Email is a fundamental communication medium which can greatly benefit from the utilisation of ACNs, especially in the case of activists, people under surveillance or whistleblowers.

ACNs are a perfect tool for journalists to communicate with their sources safely and anonymously. An example of a solution that uses ACNs for **whistleblowing** is SecureDrop, which is a free and widely recognised platform for such communication. It utilises the Tor network in order to provide anonymity of the source, which is one of the most fundamental aspects of the free press, as was recognised by the European Convention of Human Rights (ECHR). Due to anonymity by design, it is impossible for journalists to reveal the whistleblowers identity, no matter how much pressure the government or other entities put. Another widely recognised platform for whistleblowing is WikiLeaks. It is a non-profit organisation and publisher of leaked documents founded in 2006. WikiLeaks releases a large number of documents exposing human rights violations and abuses by various entities, including governments. To protect the anonymity of whistleblowers, they use a Tor onion service as a secure submission platform. Although anonymity is also preserved when accessing regular websites with Tor, onion services provide an additional layer of security desired by WikiLeaks creators: these sites are inaccessible without Tor, reducing the risk of exposure if submissions occur outside of the Tor network. GlobaLeaks is a free, open-source, and internationally available software that enables the creation of

whistleblowing initiatives. The GlobaLeaks project started in 2010 and the first version was released in 2011. It uses Tor onion services for anonymous submissions. GlobaLeaks is a framework that was adopted by more than 10000 projects [92]. The use cases of the projects implementing GlobaLeaks include independent media, activists, governments, public agencies, and many more. One of the examples is WildLeaks [93] - the world's first whistleblowing initiative for environmental and animal abuse crimes.

There are several risks that can pose a threat to people while accessing regular websites. For example, malicious local DNS servers can redirect users to other IP addresses than expected. Another possibility is from the certificate authority site, where fraudulent digital certificates can be used to perform man-in-the-middle attacks. Such events occurred in the past, Turkish CA TURKTRUST being one such example [94]. BGP hijacking is another issue that can be used against users accessing websites, which is a takeover of IP addresses by corrupting the BGP protocol used in the Internet. In order to address such issues, there is a simple solution: create a hidden service to **securely access websites**. There would not be a risk from malicious DNS servers. Although certificate authorities are not present, an onion service still provides end-to-end encryption and server authentication.

Anonymous and **secure one-to-one file transfer** may be beneficial in numerous situations, for example, a doctor securely sending patient data to him or an organisation and its employees sending sensitive information between each other internally. The advantage is that there is no need to trust any third-party service provider like DropBox or WeeTransfer nor there is need to physically pass sensitive files stored on data storage devices. An example of such a solution in Tor is OnionShare. A user who wants to send a file creates an onion URL that allows the other user to fetch the file, with the possibility to enable URL expiration so that it would be impossible to download the file again after it is downloaded once. This guarantees that the file was not downloaded by any malicious intermediary. Moreover, the service can additionally require a private key to access the file. In I2P it can be achieved with numerous communicators with file sharing capabilities or eepsites secured with private key that disallow unauthorised access.

Users can distribute files over ACNs in the **one-to-many file publishing** manner to publish documents, media, or software without censorship. An example in history where such a use case was beneficial is the Zyprexa scandal [95]. Documents were published online alleging that Eli Lilly was aware of the severe side effects associated with its antipsychotic drug Zyprexa and continued to sell it regardless. Initially, a pharmacy giant was able to quickly remove the documents from the websites where they were hosted until they were moved to a hidden service. It was no longer possible for the company to remove uncomfortable documents and people had the opportunity to read them.

Container registries can have several benefits from utilising ACNs, hidden services in particular; they can prevent censorship of container images, protect against registry spoofing attacks, increasing security of pulling images, prevents tracking of contributors in case they are not comfortable with revealing their identities, and on top of that, hidden services can provide end-to-end encryption

and server authentication.

Users can **communicate asynchronously** over anonymous communication networks in a privacy-preserving manner. As the communication is asynchronous, it can accept longer delays, but it requires the possibility of replying. There is also a possibility of **instant messaging** over ACNs, however, such a use case would require low latency. Going a step further, ACNs could be used for anonymous **voice calls** or even **video calls**, hiding not only the identity of both parts of the communication, but more importantly the fact that two parties are communicating.

Using ACNs with **software updates** can prevent numerous attacks since it is not possible to tell who is performing an update. In addition, using hidden services can provide end-to-end encryption and server authentication for users that are performing the software updates.

Collecting usage data in a privacy-preserving manner is crucial. ACNs can be used to anonymously collect **statistics** to enirely unlink the user from the usage data he is sending, unless the data contain identifiable information that should also be removed. Using ACNs can allow for large-scale analysis while preserving users' privacy.

Using ACNs in **gaming** can help circumvent the censorship of the certain game, protect users from revealing their private data which leakage may lead to doxxing, or protect IP addresses of players to avoid DoS attacks. In addition, games that are not widely considered harmful, such as chess, can be forbidden in certain countries [96]; therefore, protecting the game server in this scenario can also be justified.

The hiding of the location and IPs of web services can help prevent DoS attacks, including DDoS. The attack can occur when the service content is not necessarily in accordance with the regime government and is therefore exposed to DDOS attacks from their site. ACNs can therefore be used to host **censorship and denial of service-resistant archives**. Instead of attacking a single server, an attacker would be forced to attack at least several introduction points. One of the first propositions of such systems that were censorship-proof by design were Eternity Service and Free Haven described before; however, the most popular archive today is The Web Archive which decided to create a hidden service for its archive as the website was targeted with numerous attacks.

Anonymous communication networks can be used for **source code hosting**, as code repositories hosted within ACNs can benefit from censorship resistance, while people fetching the repositories can benefit from end-to-end encryption, server authentication, and the fact of hiding communication parties, as well as protecting against targeted attacks for users accessing certain repositories.

Highly-available distributed data storage systems can further enhance their censorship-resistant features by using ACNs. It would also protect the publishers of these files as well as the people fetching them.

While it is not a secret that companies sell the usage data of their users, especially while browsing the web, which greatly violates their privacy. As user data are highly valuable for advertisers, the process of collecting the data is evolving. Utilising ACNs can help **protect the privacy while web**

browsing.

Anonymity of the voters is one of the most important characteristics of democratic elections. It is crucial to ensure unlinkability of a given vote to its voter, and using ACNs can help with it. ACNs could have been used for **e-voting systems** once traditional voting is replaced or complemented by online solutions.

5.2. Categorising use cases

The use cases identified in the previous section can be grouped into several groups based on the requirements they have in terms of the ACNs. Anonymity is a basic requirement for these use cases, but the strength of anonymity provided can be evaluated within criteria. Each group will be declared in terms of what kind of anonymity is needed for this group: sender, receiver (server), or both. Distinguished will be described in the following subsections.

5.2.1. Low-latency intra-network communication

Use cases and application areas for which the lowest possible latency is crucial. It includes instant messaging, gaming, voice, and video calls. As these use cases are focused on two-way communication along with the fact that giving up one-end anonymity may impose additional threats from the traffic analysis perspective with low-latency communication, the comparison will be based only on solutions providing two-way anonymity, meaning that the communication has to be performed within the network. While both sides of communication are often known to each other, the importance of such a property lies in the lack of exposing metadata - who is speaking to whom. The criteria defined for this group are visualised in Table 5.1.

Table 5.1: Criteria and weights for low-latency intra-network communication use cases

Criterion	**Weight**
Low latency	0.25
Low jitter	0.25
Throughput	0.20
Both-end anonymity strength	0.15
UDP support with low packet loss	0.15

If the one-way transmission delay is too high, the above-mentioned applications cannot be achieved. Therefore, a weight of 0.25 was assigned to the low-latency criterion. According to ITU-T recommendations [97] maximum one-way latency for international calls is 400 ms, ideally 150 ms or less.

The great variation between subsequent packet arrivals can significantly reduce perceived connection quality [98], and similarly to latency, making use cases of this group not feasible to deploy. This is the reason why the jitter was also assigned a weight of 0.25. The lower the jitter value, the

better the connection quality is; the value should be less than 30 ms.

Sufficient throughput is needed to transmit video and audio frames, especially those of high quality. One of the least demanding voice codecs is G.729 which requires only 8 kbit/s of throughput [99]. When it comes to video calls, Skype required [100] a minimum of 128 Kbit/s, and the recommended throughput for high quality video calls was 500 Kbit/s. The higher the throughput, the higher the quality of the video and audio frame can be. As it is also an important property for the use case scenarios, it was assigned a strong weight of 0.2.

Although stronger anonymity is better for ACN users, according to the literature, it is not possible to provide low latency, high bandwidth, and strong anonymity at the same time [101]. Choosing stronger anonymity over the latency or bandwidth would make use cases from this use case group irrelevant, therefore anonymity was assigned a value of 0.15, which is a smaller weight than the previous criteria, although anonymity is obviously still relevant and was not omitted.

In general, low-latency applications and protocols rely on the UDP protocol, as the overhead of TCP is considered too impractical. Examples of such protocols include QUIC and RTP. Even if TCP could have provided decent performance, running a service via ACN should not require major architectural changes for popular solutions used today. While TCP handles packet loss underneath, UDP does not. However, packet loss is more acceptable than high latency or jitter [98], it is still an important property that should be considered within this category. It was assigned a weight according to its importance in this category of use cases. Taking into account all of these aspects, a weight of 0.15 was assigned to UDP support with low packet loss.

5.2.2. Highest anonymity and latency-tolerant

Use cases and application areas for which latency is not crucial and can be traded for potential anonymity benefits. Those include emails, whistleblowing, asynchronous messaging, statistics, and e-voting systems. As the use cases are mainly focused on the sender anonymity, only this type of anonymity will be considered within this category. The criteria defined for this group are visualised in Table 5.2.

Table 5.2: Criteria and weights for highest anonymity and latency-tolerant use cases

Criterion	**Weight**
Sender anonymity strength	0.5
Reliable delivery	0.25
Message persistence	0.25

As low latency, high bandwidth, and strong anonymity cannot be combined at once [101], and considering the fact that currently there are no low-latency networks providing strong anonymity (they could have been potentially based on DC-nets) in this use case group, anonymity can be chosen with the cost of longer delays, providing the strongest metadata protection, and therefore can be considered

as the main focus within the category, receiving the largest criteria weight of 0.5. It also allows for the network to be more resilient against traffic analysis attacks, which will be described in the next chapter, that can be performed by a powerful global passive adversary - potentially the one that a whistleblower wants to reveal information about.

It is crucial for the user to determine whether his message has been delivered due to the importance of the information that is often associated with this use case group. The lack of a mechanism that addresses this issue, such as an acknowledgment, is a significant drawback that was present in the early Mix-net-based ACNs [102]; therefore, the reliable delivery criterion was assigned a weight of 0.25.

Asynchronous communication can often involve parties that are temporarily offline. It is therefore a decent feature for the network to have the built-in possibility of temporarily storing messages; otherwise, the message has to be sent to an always-online entity within the ACN, possibly an offline entity outside of the ACN, or a service provider must create such a mechanism on top of the network, if there is such a possibility. The lack of these possibilities also imposes security issues as described in the literature [103]. Due to these factors, a weight of 0.25 was assigned to the persistence of the message.

5.2.3. Web browsing-based

In this category, there are use cases that are strictly related to web browsing: Internet censorship circumvention, surveillance protection in web browsing, unbiased research, circumvent geo-restrictions in the Web, web-based cryptocurrency wallets, secure accessing websites, and protecting privacy in web browsing. Within this category, in the comparison the sender's anonymity will be considered mainly, as it is a category highly focused on users. The criteria defined for this group are visualised in Table 5.3.

Table 5.3: Criteria and weights for web browsing-based use cases

Criterion	**Weight**
Clearnet support	0.25
Censorship resistance	0.25
Latency	0.2
Sender anonymity strength	0.2
Web browsing optimisation	0.1

Currently, there are over 1.5 billion websites on the World Wide Web, and less than 200 million of them are active [104]. In Tor, the most popular anonymous communication network, there are around 800 thousand unique onion addresses [105, 106] and not every onion address is associated with a website, as there are many other use cases for them. Therefore, the vast majority of websites are not reachable as hidden services, requiring for this specific use case group to allow users for exit traffic. That is why the clearnet support criterion was assigned the most significant weight of 0.25.

As many countries impose censorship on ACNs, Tor in particular as the most popular network [85, 87], it is crucial that the network address censoring so that users in these countries can use ACNs

freely for web browsing-based use cases. The censorship resistance was for that reason considered as one of the second most important features in this use case scenarios, receiving a notable weight of 0.25.

Since web browsing is an interactive activity, it requires little latency. Although the lower the value, the better it is for the user, the anonymity must be preserved. It implies a trade-off between those properties as described in the literature [13]. Due to the significance of this property in terms of comfortable web browsing, the low latency criterion was assigned a weight of 0.2.

As this use case group does not aim for the strongest anonymity that cannot be achieved without additional latency, a trade-off must be accepted. For that reason, sender anonymity strength is considered as a criterion with a significant weight of 0.2, but not more significant than latency.

ACN optimisation in terms of web browsing such as ease of setup, browser fingerprinting resistance, HTTPS support, or others. While it may bring additional benefits for the utilisation of a given ACN in web browsing, it is not a critical functionality, and therefore was assigned the lowest weight of 0.1.

5.2.4. File sharing-based

This category involves use cases and application areas related to file sharing, including: one-to-many file publishing, one-to-one file transfer, anonymous container registries, anonymous software updates, source code hosting, and distributed data storage systems. Within this use case group, both-end anonymity, including mutual anonymity (parties of the communication are not known to each other), will be considered as in these use cases, anonymity is needed for both sides of communication. Although in theory this category could also compare anonymous publication systems, they can fulfil only the minority of the mentioned use cases; therefore, they would not be taken under consideration. The criteria defined for this group are visualised in Table 5.4.

Table 5.4: Criteria and weights for file sharing-based use cases

Criterion	**Weight**
Download speed	0.3
Both-end anonymity strength	0.3
Volume correlation resistance	0.3
File sharing protocols support	0.1

Download speed is critical for transferring large files. Although the higher the download speed, the better, the high speeds that are known from non-ACN-based file sharing are not achievable in ACNs. Due to its intuitively large significance in this group, it was assigned a weight of 0.3.

Due to the importance of anonymity not only for the receiver/downloader but also sender/publisher as described in the literature [52, 53], a both-end anonymity strength was assigned a weight of 0.3.

Although volume correlation resistance can be considered as a part of anonymity strength,

a separate criterion was distinguished due to its importance in this group of usage scenarios with an appropriate weight of 0.3. Large volume transfers create an identifiable pattern that can be easily distinguished by an attacker if there are no additional protections against it [13].

There are numerous file sharing protocols that optimise the download process, for example, BitTorrent. The ability to deploy them in an ACN without major modifications is considered a desirable property, although not a critical one, and therefore assigned a weight of 0.1.

5.2.5. Infrastructure security and resilience-based

In this category, the highest importance is associated with the security and resilience of the underlying infrastructure. It includes remote access to hosts, utilising ACNs with IoT, running cryptocurrency nodes, as well as censorship and denial-of-service resistant archives. As these use cases are primarily focused on server-side anonymity, only ACNs providing such functionality will be considered. The criteria defined for this group are visualised in Table 5.5.

Table 5.5: Criteria and weights for infrastructure security and resilience-based use cases

Criterion	**Weight**
Resilience to active attacks	0.3
Authorisation mechanisms	0.3
Server anonymity strength	0.2
Academic research and maturity	0.2

Long-running services within this category must have high resistance to active attacks provided by the ACN, especially disruptive DoS ones that may cause decreased availability of the service, as was the case with an Internet Archive [107]. The ACN must be highly resistant to these types of attacks; therefore, it was assigned a notable weight of 0.3 was assigned.

One of the benefits of utilising ACNs in this use case group is the possibility of omitting exposing an infrastructure to the clearnet. As hidden services across ACNs are stored in DHT structure and can possibly be enumerated [108], additional authorisation mechanisms to prevent direct access to hidden services are desired - otherwise, these mechanisms need to be implemented on the application side. Due to its importance, it was assigned a weight of 0.3.

In this use case group, providing server anonymity is crucial. The hiding location may positively correlate with the resilience to active attacks mentioned earlier. Anonymity can also serve to protect critical infrastructure from discovery [109]. Taking these aspects into account, a weight of 0.2 was assigned to the anonymity of the server.

Academic research directly influences the resilience of the network as it investigates various aspects, potential threats, and attacks on a given ACN and gives more confidence in its safety. Maturity is directly correlated with smaller chances of existing bugs, as many of them were discovered through the years of existence of a given ACN. The reasons for this are similar to the Kerckhoffs principle [110].

The more a system is studied and tested openly, the less it relies on secrecy to stay secure. Instead, security comes from technical aspects, examined by many experts and used over time, which helps to find and fix problems. So, academic research and maturity make an ACN safer because they lead to more thorough testing and fewer hidden bugs. The academic research and maturity criterion is therefore assigned a significant weight of 0.2.

5.3. Summary

In summary, this chapter presented a detailed list of use cases and application areas for anonymous communication networks, grouped them according to shared requirements, and defined criteria for their evaluation. The role of this chapter in the thesis is to connect technical aspects with practical reality, ensuring that the later analysis of ACNs is focused on real user needs. As such, this chapter provides the necessary basis for comparative analysis and clarifies what exactly needs to be compared and why.

6. THREATS, ATTACKS AND LIMITATIONS

This chapter examines the threats faced by anonymous communication networks, focusing on passive and active attacks that compromise user anonymity and system integrity. It categorises and describes common attack vectors, evaluates how different ACNs respond to them, and highlights specific vulnerabilities to each network. The chapter also addresses broader limitations that are crucial for the long-term viability and accessibility of ACNs.

This chapter is important because understanding the range of threats is essential for evaluating existing ACNs and proposing any improvements. Only with a solid overview of real attack scenarios can we properly evaluate which ACNs are suitable for particular use cases and which are not. In addition, many of the threats presented here have already influenced the design of the most popular ACNs, with several countermeasures already in place. Recognising these defences is key before moving on to any comparison.

The first section discusses both passive and active attacks, describing in detail how they are carried out and how each ACN attempts to address them. The next part looks at economic sustainability as an often overlooked, but significant, limitation for ACNs going forward. Finally, the chapter examines the ongoing "censorship arms race" between ACN developers and those trying to block them. The presentation of these threats and countermeasures provides the necessary background for the comparative analysis in the following chapters and highlights the areas that require further development, which will be addressed in a dedicated section later on.

6.1. Attacks

Attacks on Anonymous Communication Networks (ACNs) can be broadly classified into two main categories: passive and active. **Passive attacks** on ACNs are mostly related to traffic analysis. These attacks are often extremely difficult to detect since the attacker does not interfere directly with the network traffic. **Active attacks**, on the other hand, involve some form of manipulation or interference with the network, such as injecting, modifying, or blocking messages. Active adversaries may attempt to compromise keys, introduce malicious nodes, perform denial-of-service attacks, or exploit protocol vulnerabilities. The following list contains several examples of the most prominent attacks on ACNs for both categories:

- Passive
  - Timing correlation
  - Size/volume correlation
  - Intersection attack/statistical disclosure/traffic pattern observation
  - Browser fingerprinting
  - Unencrypted content observation

- Active
  - Key exposure
  - Tagging attack
  - Content modification
  - Sybil attack
  - N - 1 attack
  - Denial of Service (DoS)
  - Compulsion attack
  - Cryptographic attacks
  - Vulnerabilities exploit
  - ACN-specific attacks

Timing correlation is one of the most effective attacks against low latency ACNs. Given an adversary who can watch traffic entering and leaving the network, he can easily distinguish who sent the message due to the timing correlation. In order to mitigate this kind of attack, a non-deterministic strategy needs to be included. In Mix-nets, batching is implemented with various strategies. Addressing the timing correlation is always associated with additional latency and, therefore, smaller usability in lower-latency usage scenarios. While there are propositions of optional inclusion of non-trivial delays and batching strategies in I2P, they are not yet implemented. As in I2P, each user acts as a router, it will impose additional effort on the attacker to distinguish the traffic originating from the user from the traffic he is passing through; however, it is not a complete mitigation of this type of attack. Tor and Lokinet are the most vulnerable to the timing correlation attack; however, the more users the network has, and therefore the larger anonymity set is, the harder it is to perform this attack.

Size correlation is an attack in which an adversary correlates the message with its sender due to its size. Lokinet is the most vulnerable to this type of attack as it does not propose any mitigation. Tor uses fixed-size messages; however, it is only a partial mitigation of the issue, especially when a user is sending large messages, as the sequential chunks clearly form a larger message, and the pattern can be, therefore, observed. In I2P, the chunks can be initially combined into larger messages and then, after leaving the sender's outbound tunnel, split and delivered through the inbound tunnels of multiple receivers, which can make the size correlation harder to perform. Nym provides the best defence against this attack, providing both padded Sphinx messages along with cover traffic, which makes observing message size patterns impossible.

Identifying users by correlating on-line presence over time is known as **intersection attack/statistical disclosure/traffic pattern observation**. When an adversary can observe one endpoint of the communication as well as the fact of a user entering the network, the fact of intensified message number on the receiving side can be associated with a certain user that is most likely a sender. This attack is especially feasible when an adversary can observe the network for a long period of time. In theory, when a user is constantly connected to the ACN, it would not be distinguishable; however,

this is not an acceptable assumption to make. The more users and routers are on the network, the more complex the attack is to be performed. Masking traffic and non-public routers, for example pluggable transports and bridges from Tor network, can also mask a fact of connecting to the network. The best solution to this issue, deployed by Nym, is to include a cover traffic - as every endpoint is constantly receiving it, an adversary is not able to tell whether a certain entity is receiving larger traffic than usual. Batching and mixing are another potential solution to this problem.

Browser fingerprinting is a browser-related attack that involves the collection of user data that can later be used to identify him. Tor solves this issue by providing a custom, hardened browser. Although other ACNs are vulnerable to this issue, it is less performable in hidden service-focused ACNs like I2P.

Unencrypted content observation is an attack associated with malicious exit nodes that can observe unencrypted traffic from users that reach clearnet resources over unsafe protocols such as HTTP. The solution for this attack is either to only use end-to-end encrypted protocols, for example, HTTPS-based websites, or not to leave the ACN at all and focus on the hidden services.

Although brute-force-based attacks on cryptographic keys are not feasible in an acceptable time with current technology, the key can be acquired by other means. In case of keys that are not periodically rotated and therefore ACNs that do not provide perfect forward secrecy, it may be potentially used to decrypt an encryption layer from the past. This attack is known as **key exposure**. In most of the ACNs, the keys are ephemeral and therefore leaked keys can only be used to unwrap one layer of encryption for a short period of time.

Tagging attack is an attack based on modifying a message so that it would be distinguishable later in the message path. Message integrity checking in ACNs is crucial when it comes to defending against this type of attack, and most of them include the mechanisms for providing integrity.

Content modification is an attack that is one step further compared to the passive unencrypted content observation attack described above. In this case, the malicious exit node modifies the content before passing it through. The mitigation is the same as described with the unencrypted content observation attack.

Sybil attack is an attack in which an adversary creates a large number of routers and tries to, for example, deanonymise users or perform other attacks such as DoS. I2P is particularly vulnerable to this attack due to its peer-to-peer architecture and the lack of effective mitigation mechanisms. In Tor, the attack needs to be performed over a long period of time as a sudden spike in the number of nodes can be easily noticed and addressed. Lokinet and Nym consider themselves to be more resistant to this type of attack due to economic incentives for the node runners. Nym also includes penalisation of the operators that register multiple nodes.

N - 1 attack is an attack aimed at Mix-nets in which an adversary floods the mix with his n - 1 messages, leaving space for one non-adversary message which he aims to target. Nym addresses this issue by using cover traffic and stratified topology.

Partitioning attack and eclipse attack are attacks in which an adversary tries to isolate a single target (eclipse attack) or a larger network segment (partitioning attack), forcing victims to connect to the routers controlled by the adversary. The attack involves interference with the mechanisms for discovering routers within the network, for example, netDb from I2P or Tor directory servers. ACNs such as I2P with more decentralised nature during new user joining, can also suffer from a specific type of the eclipse attack called the bootstrap attack in which an attacker provides a list of his malicious routers instead of the legitimate ones.

Attack that aims to disrupt the system, for example an ACN, is known as **Denial of Service (DoS)**. The attack has its distributed version - DDoS, which is performed from many places. The larger the network, the more costly the attack becomes. In peer-to-peer ACNs like I2P naive DDoS may potentially have the opposite effect, as each new user will also act as a router and speed-up the network, unless the attacker modifies his nodes to only consume from the network and do not contribute. There are many forms of DoS, often addressing ACN-specific technical solutions, for example, the Sniper attack [111] on Tor.

Compulsion attack is a non-technical type of attack in which a router runner is forced to give up control over his router in order for the adversary to control the traffic passing through it. This attack is not useful against networks that have perfect forward secrecy, which means frequent key rotation, as gaining control over nodes is not useful for decrypting previous messages. Although this attack may have been more dangerous if the node runner ran all nodes in a given path, most of the ACNs provide diversification mechanisms for the nodes in order to avoid them from being under the same jurisdiction et cetera.

While today's ACNs use algorithms considered safe, meaning that they cannot be brute forced in an acceptable time, the highest danger is associated with quantum computers and their ability to break currently safe cryptographic algorithms; therefore performing **cryptographic attacks**. While it is not a threat that is present today, Tor and I2P are already in the process of replacing their currently used algorithms with post-quantum-based ones.

There is also a possibility of **code vulnerabilities exploits**, however such attacks are mostly related to less mature ACNs with less research backup.

There are many **ACN-specific attacks** that are usually an effect of omitting a certain aspect during the projection of the network. Contrary to bugs, in this example, the ACN works as design but the issue lies with the design itself. The solution is also more costly than a resolution of a bug as it involves changes in the architecture. An example can be the guard discovery attack [112] on Tor.

6.2. Economic sustainability

Economic sustainability is a critical challenge in ACNs. The approach to the economic issues related to ACNs varies from network to network. Peer-to-peer solutions, such as I2P, benefit from the decentralised nature. Each I2P user who uses the network contributes to it for the others, enhancing

scalability and anonymity. The I2P development team does not accept direct donations; they encourage supporting application developers. One of the examples is StormyCloud, which provides a free outproxy service for I2P.

Tor has an entirely altruistic volunteer-based model in terms of running nodes. This approach has turned out to be working well for more than two decades of Tor history. Tor underlines the importance of diversity among their nodes, and the approach they have chosen seems to align with this goal.

Lokinet focuses on economic incentives for node runners, rather than volunteer-based networks. Although this approach can attract a broad set of participants, as it currently does, it introduces several potential challenges. Firstly, node runners can choose the cheapest infrastructure providers in order to maximise their profit, which can lead to less diversity of nodes within the network. Secondly, the token-based reward that Lokinet proposes may be volatile. The value of the token they provide is strictly speculative and may vary significantly over time, potentially discouraging node runners when the price goes down. The development team does not accept direct donations; similarly, for node runners, they benefit from the tokens.

The Nym network has a similar approach to Lokinet in terms of incentives for node runners. Although this can encourage more node runners, it shares the same risks, including token volatility and centralisation issues. Unlike other ACNs, Nym requires a monthly fee for network usage that funds the development behind Nym.

6.3. Censorship arms race

Efforts to circumvent Internet censorship using ACNs are continually met with countermoves by restrictive governments, creating an ongoing technological arms race. Tor is in particular an object of such actions [113], as it is the most popular ACN today. Governments try to block ACNs by targeting their management mechanisms, public relay nodes for client-server ACNs, websites from which you can download a given ACN, or by using deep packet inspection (DPI) to detect ACN-specific traffic. In response, the Tor Project has developed solutions such as bridge relays and pluggable transports to help users access the network even under heavy censorship. Although censors have access to significant resources and increasingly sophisticated techniques, the Tor Project adapts by introducing new technical measures, making censorship circumvention a continuous and evolving challenge. Other ACNs have not yet deployed such advanced censorship circumvention techniques; although it is often a part of their roadmap.

6.4. Summary

This chapter has provided an overview of the key threats, attacks, and limitations associated with anonymous communication networks. Passive and active attacks were discussed, along with countermeasures that are implemented in current ACNs. Among these, the threat of traffic analysis

remains significant, as it cannot be fully mitigated by currently popular onion routing-based solutions, while Mix-net-based approaches offer better protection against and therefore increased anonymity, as traffic analysis and the attacks based on it can significantly hurt the provided level of anonymity. Other threats were also considered, not only in terms of passive and active attacks but also about economic incentives and censorship. The issues mentioned within this chapter set the practical boundaries of ACN security and will be crucial for the comparative analysis in the next chapter. Moreover, addressing the open problems and limitations identified in this chapter will be essential for the future development of ACNs, and these will be discussed in detail in a dedicated section later in the thesis.

7. MULTI-CRITERIA COMPARATIVE ANALYSIS

This chapter presents a detailed multi-criteria comparative analysis of selected anonymous communication networks, the most important chapter of the thesis that will have a clear goal of providing an answer to the research question; whether indeed there is no universal ACN ideal for every use case. It begins with a literature-based comparison that highlights key architectural and functional differences among networks. This is followed by an experiment-based evaluation focused on performance metrics such as latency, jitter, and throughput. Finally, each ACN is rated in the context of specific use case categories, using weighted scoring to determine their suitability across different application scenarios.

7.1. Literature-based comparison

The table 7.1 summarises the comparison of ACNs currently operating based on the literature.

Tor was initially deployed in 2002; hence, it is the oldest working ACN in this comparison. It is considered a direct successor to the original Onion Routing project, utilising the same anonymisation technique. It is fully free for end users, but depends on volunteers for donating and running dedicated relays, as well as on the financial support from various organisations. Tor supports only TCP-based applications, with a strong emphasis on web applications. Access to the clearnet is fully supported and is one of the main goals of the network. Sadly, certain websites block traffic originating from the Tor network, most likely due to abuse. However, other websites, such as Facebook or Reddit, have dedicated hidden services to distinguish traffic coming from within the Tor network. Tor is vulnerable to various traffic analysis attacks due to its low latency with a lack of timing analysis attack defences or traffic volume obfuscation techniques. However, it does protect against browser fingerprinting due to the dedicated Tor Browser. Although in theory a Sybil attack could be easily performed in the Tor network, such an anomaly in the number of relays would be quickly spotted and blocked by the centralised Directory Authorities. The Tor circuits follow a free route topology. There are either three hops for clearnet traffic, or six hops for hidden-services-related traffic. These lengths could in theory be changed in appropriate files, but it is not easily accessible or advised by the Tor developers. While the number of Tor users is hard to precisely measure, the estimations from the dedicated metrics subpage vary around 2 000 000 daily users, with a recent spike to 8 000 000 users, although the number came back to the usual 2 000 000 and the reason for such a spike was not publicly given. The number of routers is roughly constant, adding up to around 10 700 relays with 8 800 publicly listed ones, and around 1 900 bridges.

In theory, I2P can be considered an older network, dating back to 2001 with the Invisible IRC project. However, the major architectural change and move towards universal networking rather than an Internet Relay Chat (IRC) specific one was performed in 2003; therefore, it is considered as an inception

Table 7.1: Literature-based comparison of ACNs

	Tor	**I2P**	**Lokinet**	**Nym**
Inception year	2002	2003*	2018	2019
Access	Free	Free	Free*	Paid
Anonymisation technique	Onion routing	Garlic routing	Onion routing	Mix-net
Architecture	client-server	peer-to-peer	client-server	client-server
Supported network and transport protocols	TCP	TCP and UDP	All IP-based	All IP-based
Clearnet support	High	Low	Medium	High
Traffic analysis resistance	Low	Medium	Low	High
Sybil attack resistance	Medium	Low	High	High
Network management	Directory Authorities	DHT	DHT/blockchain	Gateways, blockchain
Topology	Free route	Free route	Free route	Stratified
Intermediate hops number (one way)	3 or 6	0–14	3–17	5
Hidden services support	Yes	Yes	Yes	No
Users number	2,000,000 daily	More than 55,000 daily	More than 1,000,000 monthly	No information
Routers number	10,700	55,000	1,500/2,100	700

year in this comparison. Utilising the I2P network is free, and each user acts as a router for other users, unless it is unsafe for him - meaning that he is connecting to the I2P network from a restricted country. The categorisation of restricted countries is performed according to the Freedom House research. I2P uses a slightly modified version of onion routing. Thanks to unidirectional tunnels instead of bidirectional ones, it is possible to bundle multiple messages within a single transport via a given tunnel. After leaving the sender's outbound tunnel, the messages can then be distributed to appropriate recipients' inbound tunnels. The modified version was named garlic routing. The vast majority of messages are sent separately and are not combined; however, in order to distinguish differences in this anonymisation technique, it is considered as a distinct one. Moreover, the concept of unidirectional tunnels makes I2P only a semi-circuit-switched network with elements of a packet-switched network, as an end-to-end message path is never established. The length of a tunnel is modifiable. By default, the unidirectional tunnels have 3 hops, and there are usually two or three tunnels for a given purpose. The length of any

unidirectional tunnel can be set to any value from 0 to 7, with a possibility of probabilistic variations of the length to further distinguish traffic patterns. Even with a tunnel length of 0, the user still has the property of plausible deniability, as it is difficult to distinguish whether a certain traffic is designated for this user or is he just routing it for another user. In total, there can be as little as 0 hops to as much as 14 hops in one-way communication. I2P follows free route topology with remarks mentioned earlier. Multiple unidirectional tunnels with variable lengths and plausible deniability make the traffic analysis more difficult for an external observer, increasing the traffic analysis resistance of the network. It is not, however, fully resistant due to the same reasons as other onion routing-based low-latency ACNs. I2P supports both TCP and UDP traffic; however, each type of traffic requires a dedicated tunnel. Tunnels in the java-based I2P do not allow for a simple UDP-based tunnel; I2PD, a C++ implementation of the I2P router, provides such functionality. Java-based I2P allows only for a UDP-based Streamr tunnel. The I2P is mainly focused on the traffic within the network. In order for traffic to leave a network, an outproxy service needs to be established for a given application. For example, there is a built-in outproxy service for HTTP-based traffic provided by StormyCloud, allowing anonymous web browsing for I2P users. As each type of traffic needs a dedicated outproxy to leave a network and the outproxies are not always fully reliable, the clearnet support is considered as low. Due to its decentralisation, I2P is highly vulnerable to Sybil attacks, as there is almost no mechanism for controlling an unnaturally large number of nodes - a powerful attacker that would be able to launch tens of thousands of nodes would be able to take over the network. There is, however, a very basic anti-Sybil mechanism with checking IP closeness. Currently, there are around 55 000 routers within the I2P network and the number of users might be slightly larger, as users from restricted countries do not participate in routing messages.

Lokinet is a newer approach, existing since 2018 both when it comes to the code release and the whitepaper. In theory, the usage of Lokinet is free; however, the free exit nodes are usually non-functioning, which makes Lokinet unusable for clearnet usage. However, there is a possibility of purchasing access to private exit nodes or hosting custom ones, although it might be problematic from the anonymity point of view as it may link the user who wishes to remain anonymous directly to the exit node that he is using. Lokinet also utilises onion routing as an anonymisation technique, however on a lower layer than Tor and I2P do. While Tor supports TCP-based applications and I2P supports both TCP and UDP, Lokinet supports all IP-based protocols, including TCP and UDP but also, for example, ICMP. Hidden services hosted within Lokinet - SNApps, may therefore utilise a wide variety of traffic protocols. Lokinet utilises client-server architecture, free route topology and suffers from the exact same vulnerabilities in terms of traffic analysis as Tor. When it comes to the resistance of a Sybil attack it is more resistant than both Tor and I2P, despite the fact that it also uses decentralised DHT structure, in the form of blockchain, for the network management that in theory prones for the attack. The resistance is associated with a fact of cryptoeconomics - each time a new node is introduced to the network it needs to lock a fixed amount of tokens. With each new node the token pool shrinks, effectively increasing the price of the token and making the attack increasingly expensive with every

new node. While the exact number of Lokinet users is difficult to estimate, the Session application, which is the most popular application utilising Lokinet infrastructure, has more than one million users a month - it can be therefore considered as the lower boundary for the number of users. The number of nodes initially varied around 2,100; however, during the development of this thesis, it decreased to approximately 1,500. This reduction was most likely related to the migration associated with changes in economic incentives for node operators [114]. This demonstrates in practice how economic incentives for node operators can be unreliable and may negatively impact the network.

The youngest ACN in the comparison is the Nym network, which had its first presentation in late 2019, the paper describing it was written two years later, in 2021. The NymVPN, a service that enabled the utilisation of the Nym Mix-net, was officially launched in March 2025 and it is available as a paid service. Contrary to previous ACNs Nym is not onion routing-based - it uses Mix-nets with client-server architecture. It is heavily based on another Mix-net design, Loopix, and many technical solutions are derived from there, including stratified topology and traffic analysis resistance mechanisms like cover traffic and Poisson mixing. Nym has fixed-size 5-hop routing. Similarly to Lokinet, Nym provides economic incentives for node runners which not only encourages more people to run router nodes but also enhances Sybil attack resistance. Another similarity is the use of blockchain for network management, which in Nym is also joined by gateways to charge users for network usage. Nym provides decent built-in clearnet support and does not support hidden services. There are no data on the number of Nym users. The number of nodes is around 700.

7.2. Experiment-based comparison

The experiments were carried out using two computing environments: a private workstation and a cloud-based virtual machine. The private machine operated on macOS 13.6.3, equipped with a 2.4 GHz 8-core Intel Core i9 processor and 32 GB 2667 MHz DDR4 RAM. The cloud environment utilised an Amazon AWS t2.micro instance.

For each anonymous communication network that supports hidden services, an appropriate hidden service was configured and deployed. The scripts used for the setup process are provided in Appendix A: Bash scripts for hidden services setup. It should be noted that I2P requires additional configuration, includes bandwidth and tunnel setup, which is the most easily achieved through its graphical user interface. To enable remote configuration, SSH tunnelling was used to forward the I2P router console (port 7657) from the remote machine to the local environment. The firewall was not additionally configured for hidden services due to the NAT/hole punching capabilities.

Table 7.2 shows the results of the ACNs measurements for both clearnet and hidden services traffic. Latency and jitter were measured using a simple Python client-server application, which was accessed through a specific ACN that was measured. The code of the application is provided in Appendix B: Python application for measuring RTT and jitter.

In each trial, the measurement was repeated 50 times and the result was the average across

Table 7.2: Experiment-based comparison of ACNs

	Tor	**I2P**	**Lokinet**	**Nym**
RTT - clearnet [ms]	240	Not measured	Not measured	3152.92
Jitter - clearnet [ms]	20.05	Not measured	Not measured	1221.12
RTT - hidden services [ms]	393.94	813.27	352.13	N/A
Jitter - hidden services [ms]	43.97	54.25	26.6	N/A
Throughput - hidden services [Kbit/s]	2240	775	3504	N/A
Download speed - hidden services [Kbyte/s]	280	591	438	N/A

these repeats. The trials were also repeated multiple times, as a specific measurement may have been taken during the periodic routing path change or suffered from temporary bottlenecks. The median of these measurements was chosen as a final value to achieve the most objective result.

The throughput was initially measured by iperf3, but it turned out that it did not give correct results for ACNs, and therefore it cannot be considered a suitable tool for measuring throughput in them; it showed a far too low value compared to a simple file download; therefore, the throughput was measured by file download, which is more accurate, although not ideal. There is, however, a dedicated metric of file download speed which can be different from throughput in case ACN allows an alternative way of downloading files than simple client-server communication. There were also attempts to perform dedicated measurements for UDP; however, as Tor does not support UDP, measuring Nym alone does not make sense in terms of comparing ACNs. There was a similar issue with UDP for hidden services - Tor does not support UDP, and while I2P in theory does, there is no basic UDP tunnel in Java-based I2P; there is one in C++-based I2Pd, although there were difficulties in terms of compatibility with applications and packet loss, and the measurements were unstable - therefore, the measurements were discouraged, leaving Lokinet the only ACN with full UDP support for hidden services.

Measurements were performed for both clearnet and hidden services where there was such a possibility. As the Nym network does not support hidden services, it was only included in the clearnet measurements. Tor was included in both clearnet and hidden services. I2P requires a dedicated outproxy for each type of traffic. There were attempts to measure latency and jitter by a similar client-server web-based application using web sockets, but the I2P HTTP outproxy did not handle the web sockets properly, while Tor and Nym did not have such issues. There were also attempts to measure it with external tools, but the results were unstable and varied significantly between each trial. Moreover, the outproxy suffered from significant downtimes. Taking into account all of these aspects, the clearnet measurements for I2P were abandoned, as it is clearly not the right network for such traffic. Although

Lokinet in theory also supports browsing the Clearnet, the freely available exit nodes were not functioning in any of the attempts to perform measurements. There is a possibility of creating a private exit node or purchasing access to one; however, from the anonymity perspective, it is definitely not desired as the exit node can be directly correlated to an owner and/or vendor and potentially deanonymise the user. Certain Lokinet exit node vendors allow for cryptocurrency payments which may protect users as well as short-living exit nodes, but it was not purchased for the sake of measurements due to large Bitcoin fees at the time of performing them and lack of possibility of paying not in crypto, which is understandable considering the risk. Despite the choice of cryptocurrency, the vendor required information about a purchaser which potentially may lead to deanonymisation. Such inconveniences and potential threats are considered not desired and therefore measurements for the clearnet in Lokinet were not performed.

Although Nym is also a paid service, one does not pay for a certain entry/exit node, only for network access, with many layers of privacy protection as described in Nym section, therefore the risk is considered significantly lower than the one associated with Lokinet.

Certain measurements are not present in the results table. For example, packet loss was checked with a simple ping command for Lokinet and Nym. The median value across the measurements was 0%. The result is not present as the comparison is not fair - Lokinet was tested in terms of hidden services, while Nym was tested in terms of clearnet.

Another measurement that did not appear was Nym's throughput measurement for two reasons; firstly, there were issues with measuring it via the mechanism that was used for other ACNs and it was evaluated with online tests that were not performed for other ACNs. Secondly, placing this measurement in the hidden services table would not be accurate as it was performed for the clearnet connection; although, considering that connection to the inter-network service provider would involve the same number of hops, the measurements should vary significantly. The results in the tests showed very low throughput, around tens of Kbytes/s, which was expected due to the network architecture.

The results of the measurements were mostly expected, based on the literature, although there were certain unexpected aspects. In terms of latency and jitter on clearnet Tor performed surprisingly well, achieving 240 ms of Round-Trip Time (RTT), or 120 ms of latency and 20 ms of jitter. Nym was measured with 3152 ms of RTT, or 1576 ms of latency, and 1221 ms of jitter - significant values but they were expected due to the Mix-net architecture.

With hidden services, the lowest RTT and jitter were measured for Lokinet, 352 ms of RTT (176 ms of one-way latency) with 26.5 ms of jitter. Tor was not much slower; it had 394 ms of RTT (197 ms of one-way latency) and 44 ms of jitter. I2P turned out to be the slowest for hidden services in terms of RTT and jitter with 813 ms of RTT (406.5 ms of one-way latency) and 54 ms of jitter. In terms of throughput for hidden services, Lokinet also was measured with the highest value of 3504 Kbit/s, Tor with a slightly lower value of 2240 Kbit/s, and I2P with the lowest value of 775 Kbit/s. In the matter of download speed, Tor and Lokinet were assigned with the same value as in throughput and converted to KByte/s, 280 KByte/s and 438 KByte/s respectively, as throughput was measured by a client-server

file download and these networks do not have any techniques of optimising file downloading. However, there is a significant difference in terms of file download for I2P due to the built-in I2P snark that supports the BitTorrent protocol. The download speed is faster than in other ACNs, despite the fact of lower single-link throughput. The median value was 591 KByte/s with 21 peers. It shows the strength in terms of download speed of peer-to-peer file sharing protocols.

7.3. Comparative analysis in terms of use cases and application areas

In this section, ACNs will be evaluated in terms of using them in various use case groups. The scores will then be adequately justified. In general, ACNs will be evaluated with values from 1 to 10, with the possibility to rate 0 in case a certain trait is not present in the network at all. There is also a possibility of scores with halves. The scale is not linear.

7.3.1. Low-latency inter-network communication

Table 7.3: Comparison of ACNs in terms of low-latency inter-network communication use cases

	Tor	**I2P**	**Lokinet**	**Nym**
Low latency (0.25)	7.5	4	8	1.5
Low jitter (0.25)	6	5	8	1.5
High throughput (0.2)	7	4	8	1.5
Both-end anonymity strength (0.15)	5	5	5	8
UDP support with low packet loss (0.15)	0	5	10	10
Weighted sum	5.525	4.55	7.85	3.75

In Table 7.3 Tor, I2P, Lokinet and Nym were evaluated in terms of their use in use cases that require the lowest latency in inter-network communication. The latency should be less than 400 ms, preferably less than 150 ms, and the value of 10 would be assigned to the ACN with a value lower than 50 ms to be suitable for the most demanding use cases [97] and the most comfortable user experience; the ACN with a latency less than 50 ms would achieve a score of 10. As there is currently no ACN with so low latency, smaller scores were assigned. Moreover, these values refer to one-way latency and are assumed to be half the RTT value. Unfortunately, none of the ACNs achieved less than 150 ms of one-way latency; Lokinet scored 8, as it had the lowest RTT value. Tor's RTT was not much lower, and therefore he scored 7.5. The RTT of I2P was significant; therefore, a score of 4 was assigned. However, the highest latency was with the Nym network, which was assigned 1.5. Half a point indicates that the latency could have been higher, as Nym has relatively low latency compared to other Mix-nets [16].

In terms of jitter, the value should be less than 30 [98], and a score of 10 would be assigned

to the network with a jitter less than half of that value. As there are no such ACNs, the assigned values were once again smaller. Lokinet achieved the highest score of 8. In fact, it was the only score that fits in the advised maximum of 30 ms of jitter. Tor had a bit higher jitter value, although higher than the desired 30 ms; therefore, it scored 6 out of 10. Although latency in I2P was high, its jitter is relatively low in comparison; however, it was still higher than Tor's, so it was assigned a score of 4. The highest jitter was, of course, associated with the Nym network, which is a desirable property from the anonymity point of view but not necessarily in this use case scenario group.

The throughput of Tor, I2P, and Lokinet was sufficient in terms of basic requirements within this group. Although a throughput of 128 Kbit/s would be sufficient for basic video calls [100], a higher quality of video and a higher number of frames per second would require a higher bandwidth. The higher bandwidth, the better, and the value of 10 would indicate that there is no need to further increase the bandwidth. There are use cases that could have potentially been deployed in ACNs once the bandwidth is high, for example anonymous live streams, but live streams require significant throughput, at least 6 Mbit/s for 60 FPS in Full HD for the H.264 codec [115], but these values are not yet achievable with ACNs, therefore no 10 was assigned. Lokinet was measured with the highest throughput and scored a value of 8, Tor had a slightly lower value and achieved 7 out of 10, I2P had drastically lower throughput than Tor and Lokinet, although still sufficient for basic requirements; therefore, a score of 4 was assigned. Nym had the lowest throughput, which is directly associated with its architecture, and was assigned a score of 1.5.

In terms of anonymity strength, the value of 10 would be assigned to an ACN with undisputed protection against all forms deanonymisation, especially in terms of traffic analysis. Tor, I2P, and Lokinet scored an equal value of 5. Although there are some differences between them, they all share onion routing architecture or its variations. On the one hand, Tor can be considered as more anonymous as it has a larger anonymity set. On the other hand, I2P has better plausible deniability as an observer is not able to distinguish whether a user received traffic for himself or just passed it through, and I2P has better size/volume obfuscation. Lokinet can achieve anonymity on lower layers than Tor and I2P which can also be beneficial in terms of anonymity. As this paper does not aim to provide an anonymity measure for onion routing-based ACNs, they will be assigned the same weight as the biggest threat to all of these networks is traffic analysis to which they are all vulnerable. Nym is more resistant to traffic analysis, therefore, it was rated with a higher score of 8 out of 10, with one point subtracted due to the potentially low anonymity set, or at least not publicly known, and the second subtracted due to the current lack of the possibility of modifying parameters in a way that would allow more anonymity, as described in Nym's white paper [16].

In terms of UDP support, a fully operational support would be assigned with 10, while the lack of such a support would be assigned with 0. When support is only partial or not fully working, its value would be between those values. From existing ACNs, only Lokinet and Nym provide full support, and for that reason, they scored 10 out of 10. I2P provides limited support, with the issues described

earlier; as a consequence, it scored 5. Tor does not support UDP at all and therefore was assigned a 0.

The most suitable ACN for the low-latency use case group turned out to be Lokinet due to its consistently low latency and jitter, high throughput, and full support for UDP with low packet loss.

7.3.2. Highest anonymity latency-tolerant

Table 7.4: Comparison of ACNs in terms of highest anonymity latency-tolerant use cases

	Tor	**I2P**	**Lokinet**	**Nym**
Sender anonymity strength (0.5)	5	5	5	8
Reliable delivery (0.25)	10	10	10	10
Message persistence (0.25)	0	0	10	10
Weighted sum	5	5	7.5	9

Table 7.4 compares Tor, I2P, Lokinet, and Nym for scenarios where anonymity for the sender is the most important and latency is acceptable.

The anonymity of the sender is the main factor here. The strength and possible scores are defined as in the previous use case group. Nym got the highest score (8) because its Mix-net design is better at protecting users from traffic analysis. Tor, I2P, and Lokinet have similar weaknesses due to their onion routing-based methods, so they all scored 5. More justification was provided within the previous category.

Reliable delivery matters a lot when messages are critical. When the mechanisms of message reliability are undisputable, they would be assigned with 10. When an ACN lacks this functionality, as was the case with older ACN designs [102], it would be assigned 0. In terms of today's ACNs, all networks scored perfectly (10) because they deliver messages reliably. While it is expected from TCP-based messages, even UDP-based ones were not associated with significant packet loss, as was described before.

Message persistence, or keeping messages available until received, is crucial for asynchronous communication [103] and can further obfuscate traffic analysis. Lokinet and Nym already include this feature, so they got 10. Tor and I2P do not have this built-in, needing extra setup; therefore, they got 0.

The most suitable ACN for the highest anonymity latency-tolerant use case group turned out to be Nym due to its strong anonymity provided by its Mix-net architecture, reliable message delivery, and built-in message persistence. The design of Nym effectively mitigates traffic analysis threats, which are critical in sensitive communications.

Table 7.5: Comparison of ACNs in terms of web browsing-based use cases

	Tor	I2P	Lokinet	Nym
Clearnet support (0.25)	9	4	4	10
Censorship resistance (0.25)	9	5	1	1
Low latency (0.2)	8.5	4	8	1.5
Sender anonymity strength (0.2)	5	5	5	8
Web browsing optimisation (0.1)	10	3	6	6
Weighted sum	8.625	4.55	4.85	5.325

7.3.3. Web browsing-based

Table 7.5 compares Tor, I2P, Lokinet, and Nym for web browsing scenarios.

Clearnet support is the most important aspect here. An ACN with flawless support would score a value of 10, while a complete lack of such support would result in a score of 0. Nym scored highest (10) because it perfectly supports browsing the regular Internet. Tor is also good (9), with one point subtracted due to blocking exit nodes as described in the chapter about threats. I2P and Lokinet are limited in accessing regular sites, so both scored lower (4); their issues with clearnet support were described before.

Censorship resistance is important for unrestricted browsing. The existence of mechanisms that allow for seamless censorship circumvention regardless of a country would achieve a score of 10, while a lack of such mechanisms would result in a score of 0. Although there is no ACN that provides a perfect solution to censorship, as it is an ongoing arms race as described in Chapter 6, today's ACN Tor excels (9) because it has many methods to circumvent censorship, including bridges and pluggable transports. I2P provides moderate resistance (5) as it is more difficult to ban constantly changing peers in peer-to-peer design compared to long-running servers in the client-server design. Lokinet and Nym don't have strong anti-censorship capabilities, although both acknowledged the need for such a mechanism; therefore, both scored 1.

Low latency is crucial for comfortable browsing. Tor did very well (8.5), the score was based both on the latency in clearnet and for hidden services. The second was Lokinet (8). The latency of I2P was noticeably higher, scoring 4, and Nym's latency was very high, earning just 1.5. The latency criterion is similar to the one described in the first use case group, although it also includes measurements for clearnet, if applicable.

Sender anonymity is important for privacy. The strength and possible scores are defined as with the previous use case groups. Nym has stronger anonymity (8) due to its Mix-net architecture. Tor, I2P, and Lokinet have moderate anonymity (5) due to their onion routing methods as described

above.

The optimisation of web browsing, such as easy setup and fingerprint protection would be rated in terms of the ACN with the strongest support - it will receive a value of 10, while others would receive smaller values, depending on how large the difference is. Given this criterion, it strongly favours Tor (10), as Tor Browser is easy-to-install software that includes a dedicated hardened browser that protects against browser fingerprints. Lokinet and Nym have some optimisations, mostly in terms of ease of setup (6), while I2P lacks significant optimisations and has difficulties in setting up clearnet support, scoring 3.

The most suitable ACN for web browsing-based use cases turned out to be Tor, due to its excellent balance of clearnet support, censorship resistance, low latency, great latency/anonymity trade-off, and robust optimisations tailored specifically for web browsing.

7.3.4. File sharing-based

Table 7.6: Comparison of ACNs in terms of file sharing-based use cases

	Tor	**I2P**	**Lokinet**
Download speed (0.3)	6	8	7
Both-end anonymity strength (0.3)	5	5	5
Volume correlation resistance (0.3)	3	6	1
File sharing protocols support (0.1)	0	10	0
Weighted sum	4.2	6.7	3.9

Table 7.6 evaluates Tor, I2P, and Lokinet for file sharing purposes. Nym is not evaluated as it does not provide mutual anonymity for the server and receiver; in other words, hidden services are required for this use case group.

The download speed here matters greatly. A value of 10 would indicate that a given ACN achieved a perfect download speed and there is no need to further increase it - according to research [116], people's valuation of a bandwidth beyond 100 Mbit/s is relatively low. It can be assumed that it is approximately equal to a download speed of 12.5 MByte/s. However, these download speeds are not realistic with today's ACNs, so values lower than 10 were assigned. I2P got the best score (8), Lokinet scored decently (7), while Tor, less suited for large file transfers, got 6. It is important to note that I2P had lower throughput than Tor and Lokinet, but the download speed is faster - it is achieved thanks to the fact of using P2P file sharing; relatively slow upload speeds from various peers can sum up into more significant download speed, utilising I2P's multiple tunnels and making the bottlenecks less likely as BitTorrent favours faster peers.

Both-end anonymity, or anonymity for senders and receivers, is similar across Tor, I2P, and Lokinet. All scored equally (5) due to vulnerabilities in the traffic analysis. The definition of strength and scores are defined with previous section; however the anonymity has to be applied to both sides with location hiding on both sides; therefore, only ACNs supporting hidden services are considered.

The resistance to volume correlation is significant in large transfers. The value of 10 would be assigned to an ACN with flawless volume correlation resistance mechanisms; for example cover traffic joined by constant message rate and size. From the considered ACNs, the best mechanisms are in I2P (6) due to garlic routing capabilities, many tunnels, plausible deniability, and significant possibilities in terms of changing the properties of tunnels, obfuscating the volume pattern. Moreover, utilising peer-to-peer file-sharing protocols can further help to obscure traffic patterns as smaller chunks are sent through the network from various sources. However, there is still room for an adversary with enough resources to still perform the traffic analysis. Tor has moderate protection (3) such as fixed-size cells, and Lokinet lacks effective protection, scoring lowest (1).

The support for file-sharing protocols is mostly a binary score; a network either provides support, scoring 10, or does not provide, scoring a value of 0. Given this criterion, it clearly favours I2P (10) because of its support, for example, for BitTorrent with various clients (not only built-in I2PSnark), but also supports less-used protocols like Kad network clients or Gnutella clients. Tor, Lokinet, and Tor do not have built-in support nor are they easily deployable, all scoring 0.

The most suitable ACN for file-sharing-based use cases turned out to be I2P, due to its high download speeds, mostly enabled by peer-to-peer protocols, strong volume correlation resistance through garlic routing and other architectural characteristics, and built-in support for common file-sharing protocols.

7.3.5. Infrastructure security and resilience-based

Table 7.7: Comparison of ACNs in terms of infrastructure security and resilience-based use cases

	Tor	**I2P**	**Lokinet**
Resilience to active attacks (0.3)	8	6	7
Authorisation mechanisms (0.3)	10	7	0
Server anonymity strength (0.2)	5	5	5
Academic research and maturity (0.2)	9	6	2
Weighted sum	8.2	6.1	3.5

Table 7.7 compares Tor, I2P and Lokinet for infrastructure protection and resilience. Nym is not evaluated as it does not provide a possibility of server anonymity, which is needed for this use case

group.

Resilience to active attacks, such as denial of service, is strongest in Tor (8), benefiting from its central management and history. Lokinet, due to blockchain-based incentives, has decent resilience especially for Sybil attacks; therefore, it scored 7, and I2P, with its decentralised design, scored slightly lower - 6. Although I2P is slightly more resistant to DDoS attacks, it is vastly more prone to Sybil attacks, as described in previous chapters. The value of 10 would be assigned to a network that provides reasonably complete resilience to active attacks. As there is no such network today, the scores were appropriately adjusted.

Authorisation mechanisms would be assigned with a score of 10 for the existing strong mechanisms within ACN, 0 for the lack of them. These are present in Tor, which scored the highest (10) due to its established security options. I2P has reasonable mechanisms for encrypted lease sets (7). Lokinet currently does not have integrated security features, scoring 0.

The strength of server anonymity is similar across all networks, scoring moderately (5) due to shared vulnerabilities from onion routing. This criterion is defined as in the previous sections, considering only ACNs that provide server-side anonymity.

Academic research and maturity strongly benefit Tor (9) as it is widely studied and tested. I2P has a moderate research background and maturity, scoring 6. Lokinet, being new with less research, got the lowest score (2). The value of 10 would be assigned to a network that demonstrated stable long-term performance, supported by a significant number of research papers and the absence of significant vulnerabilities over time. Although Tor's research is the most extensive and the network can be considered as mature, the value of 9 indicates that this topic should still benefit from more science coverage.

The most suitable ACN for infrastructure security and resilience-based use cases turned out to be Tor, due to its resilience to active attacks, robust access control mechanisms, and extensive academic research and maturity.

7.4. Summary

A detailed side-by-side comparison made it clear that there is no single universal ACN suitable for all possible use cases. Instead, each network finds its strengths in different areas, shaped by both their technical architectures and their real-world performance.

In low-latency use cases, Lokinet turned out to be the most suitable ACN. This is mainly due to its low latency and jitter, high throughput, and native support for UDP, which gives it a clear advantage over other networks in this group.

For the category of use cases where the highest possible anonymity is required and high latency is acceptable, Nym proved to be the strongest option. Its Mix-net design, with built-in defences against traffic analysis and support for message persistence, offers the best protection for senders, making it the most robust choice when anonymity is the top priority.

When it comes to web browsing-based scenarios, where clearnet support, censorship resistance, and practical usability matter the most, Tor remains the network of choice. Its decent clearnet support, strong censorship circumvention mechanisms, solid balance between latency and anonymity, and optimisation for web browsing made it the most suitable ACN for this category.

In the area of file sharing-based use cases, I2P turned out to be the most suitable. Its peer-to-peer nature, quick download speed, garlic routing, and built-in support for common file-sharing protocols, combined with good resistance to volume correlation attacks, make it the most effective for this category.

Finally, for infrastructure security and resilience, Tor was shown to be the most reliable choice. Its robust authorisation mechanisms, high resilience to active attacks, maturity, and research background made it an ACN of choice for this category.

In summary, the comparative analysis shows that no single ACN can be called the best in every context. Each specialises in a particular field, and the most appropriate choice always depends on the requirements and threat model of the specific use case. These results directly answer the research question stated at the beginning of the thesis and provide a practical guide for selecting the right ACN for each scenario.

8. FUTURE DIRECTIONS

Previous chapters, especially comparative analysis, point to application areas where ACNs are especially useful. In order to maintain this usability and develop it further, appropriate steps should be taken. The aim here is to show not only what needs fixing, but also where the most significant improvements can realistically be made. The structure is simple: first, general recommendations that every ACN can benefit from, then dedicated sections for each network, focusing on the specific issues and future priorities that emerged from earlier analysis. It gives a clear direction for the desired development directions.

8.1. Common

This section addresses future directions that are relevant to all today's ACNs, regardless of architectural or protocol differences. The previous chapters highlighted several challenges, such as the need of a high user adoption, the potential threat of quantum computing, and the ongoing arms race with censors, that is affecting or can potentially affect every network.

The popularity of a given ACN directly influences the size of the anonymity set and therefore the provided level of anonymity. The more users the network has, the more diverse they are, the harder it is to target one specific user as he blends into the crowd. In order for the popularity to increase, awareness needs to be spread. One way to do this is by giving talks and interviews. Although people care about their privacy, awareness about privacy by design solutions leaves room for improvement.

Another crucial consideration in the future of ACNs is the support for post-quantum cryptography. Quantum computers pose a threat to ACNs that do not use quantum computers-proof cryptography algorithms. The process of introducing them in ACNs is already ongoing, for example, I2P and Nym publicly stated switching to PQC as a part of their roadmap.

One of the most important things ACNs need to address is progressive blocking by restrictive countries. In order to address this issue in Tor, bridges were created, but with time they are also getting blocked in one way or another, which was described before in this section. This leads to an unfair arm race with several countries that have a huge amount of money they can spend on ACN censorship. Current bridge distribution mechanisms seem to be insufficient, and pluggable transports were introduced that will further enhance the censorship circumvention capabilities of Tor. Although other ACNs such as Lokinet and Nym face similar blocking, they have not yet implemented mitigation techniques. They can utilise similar mechanisms of bridges and pluggable transports as Tor does or introduce custom censorship circumvention solutions. Although I2P seems to be the least censored ACN, most likely due to the peer-to-peer architecture and large number of routers in general, it can still be censored via DPI. Maybe censoring regimes consider I2P as a niche solution and therefore unworthy of censoring, although once the network gets more recognition, the more aggressive blocking will likely

be imposed. I2P should therefore also propose censorship circumvention techniques.

8.2. Tor

Following from the previous analysis, this section focuses on future work for Tor, especially in those use cases where it performed best and in areas where limitations were most apparent. Specific attention is paid to improving usability, broadening platform support, addressing discrimination by online services, and exploring deeper technical enhancements that could sustain or even expand Tor's strengths.

In the comparative analysis, Tor was shown to be the most suitable ACN for web browsing-based use cases, due to its strong clearnet support, robust censorship resistance, low latency, and user-friendly tools such as the Tor Browser. To further enhance Tor's advantage in this area, future improvements should focus on optimising web browsing experience even further by speeding up page loads, improving exit node reliability, and making setup seamless on all major platforms, including mobile. Addressing these areas will help Tor remain the de facto choice for privacy-conscious web users while keeping pace with the evolving expectations of everyday Internet browsing.

One of the main features of anonymous communication networks is the number of users, as it directly influences anonymity. In addition to the number of them, the users should be as diverse as possible. In order to encourage more people to use Tor, the software needs to be as user-friendly as possible. A great deal of progress has been made in this field over the years, resulting in an easy-to-use Tor Browser. Sadly, it is not yet available on iOS devices. Alternatives like Orbot need to be utilised; however, it lacks the security properties of the Tor Browser. The development of Tor Browser is significant, as Tor is the most suitable network for web browsing-based activities, as proved in the comparative analysis chapter.

Certain websites block traffic that originates from Tor or discriminate against Tor users by other means. An example of a partial discrimination in this field is present in Wikipedia where Tor users can view articles but cannot edit them, despite the fact that they are logged into their accounts or not. There is a need to make the website runners aware of the legitimate intentions of most of the ACN users.

Although the performance of the relays in terms of committed bandwidth is dependent on the runners of the relays, the delays can be improved on the Tor project side. Potentially ongoing rewrite to Rust may help with it as Rust can better utilise concurrency. In theory, cryptography could also be improved - it should be examined whether pre-computing approaches as described with cMix would improve Tor's performance. Tor could also include UDP support or move down one layer and work within the Network layer. As UDP is becoming increasingly popular in the Web thanks to QUIC, enabling faster web browsing, supporting it can be beneficial in terms of end-user delay. However, supporting UDP would be a major change.

Tor also turned out to be the most suitable choice in the infrastructure security and resilience-

based use case group, as confirmed in the comparative analysis chapter. This is largely due to its established authorisation mechanisms, resilience to active attacks, reasonable server anonymity, and the maturity and depth of academic research behind the project. In terms of future improvements, it will be crucial for Tor to further strengthen relay security and to encourage node runners to maintain a diverse and globally distributed network of relays, something that directly impacts its resistance to both technical and organisational attacks and lowers the chance of a powerful adversary to control enough nodes to successfully perform complex passive attacks. At the same time, Tor Project should continue investing in research to proactively identify and address emerging threats.

8.3. I2P

This section looks at the specific recommendations for I2P, in light of its performance in different categories during the comparative analysis and risks defined in previous chapters. Issues like Sybil resistance, protocol latency, and incomplete UDP support proved to be real-world bottlenecks. The focus should be put on making I2P more robust and competitive, building on its existing strengths while addressing its weakest points.

The comparative analysis clearly demonstrated that I2P is best suited for file sharing-based use cases, primarily due to its peer-to-peer architecture, strong support for file sharing protocols like BitTorrent, and robust volume correlation resistance. To further reinforce I2P's position as the top ACN for file sharing, improvements should be directed toward optimising peer discovery, increasing the speed and reliability of large downloads, and providing additional size obfuscation methods. Optional cover traffic is a proposition that should be examined. Very trivial potential improvement to file speed is to change the default bandwidth settings - current ones are significantly low and there is a high possibility that many users may not adjust their bandwidth to their capabilities.

For a long time, I2P did not have any solutions to address Sybil attacks, and it is one of the biggest threats to the I2P network right now, as was described in the threats chapter. The work on it is ongoing, and there is a mechanism for detecting such an attack via IP closeness. Sadly, it is not yet sufficient for a full mitigation of this attack. One of the proposals was to include proof-of-work mechanisms for creating a new identity, making the attack more expensive to perform. Nevertheless, enhancing Sybil attacks, described more extensively in the dedicated threats and attacks chapter, resistance should be a crucial consideration.

As was shown with the latency measurements, I2P suffers from a significant latency. I2P included speeding up cryptography as a part of their roadmap, which will have a positive impact on latency. As part of speeding up, they can consider precomputation proposals as in Chaum's cMix. Another aspect that may influence latency is the complexity of the I2P protocol stack - a simplification of it may be considered if decreasing latency by other means does not help.

Mainline I2P does not support basic UDP tunnels yet - it is only possible with I2PD - a C++ implementation of an I2P router. Even then the integration is not perfect; it was noticed during the

measurements that while iperf3 worked fine with TCP tunnels, it did not work with UDP ones. There is significant room for improvement for UDP support. Once a decent UDP support is provided, the potential I2P's use cases might be wider than they are now.

8.4. Lokinet

This section outlines key areas for Lokinet's future growth. In the comparative analysis, Lokinet stood out as the most effective solution for low-latency inter-network communication use cases, thanks to its consistently low latency, low jitter, and high throughput along with support for UDP traffic. The focus is on improving transparency and usability through better documentation, re-evaluating current architectural choices, and increasing the number and diversity of nodes.

Lokinet is heavily focused on latency. It should be examined whether the number of hops is not too large - according to the documentation, it includes 7 hops, while Tor utilises 6 for hidden services, I2P as well. Although Lokinet has indeed lower latency than Tor, the difference is not that significant, as was shown in the measurements results. Potentially, there is also room for improvement within the cryptography used. Ideally, while aiming for the latency decrease, the jitter decrease should be addressed as well - potentially it will decrease alongside the latency.

High throughput is yet another advantage of Lokinet. Throughput can be increased by providing even greater rewards for node runners that contribute with significant bandwidth.

The Lokinet documentation is relatively modest compared to Tor or I2P. The Lokinet whitepaper puts a significant focus on the token itself, leaving many ACN-related topics undescribed. Even such a basic aspect as the number of hops is not properly explained in the documentation. The documentation could definitely be improved.

Despite the fact that the number of nodes in Lokinet is not the lowest, it still should be increased. Currently, running nodes is associated with a significant cost. While it may be beneficial in terms of Sybil attack resistance, it may discourage some node runners who would like to voluntarily run nodes rather than utilise a for-profit token scheme.

8.5. Nym

Nym, as the youngest ACN in this comparison, stands out in the high anonymity, latency-tolerant use case group. Its Mix-net design and cover traffic make it much harder to attack with traffic analysis, giving it a clear advantage here. Still, there is room for improvement: the number of nodes could be higher, parameter adjustments promised in the literature are not yet available, and the documentation around intra-network services could be clearer. Addressing these would further strengthen Nym's position as the go-to choice for users prioritising maximum anonymity.

As Nym is the youngest network and was just recently officially launched, it has the lowest number of nodes within the network. Although the placement of nodes seems to be diversified, the number could have been increased to make the network safer, in general.

While both Nym and Loopix paper describe modifying network parameters in a way that would allow for an adjustment of provided anonymity level and latency, it is not modifiable yet. With this mechanism implemented, Nym can provide even higher anonymity than it does now, further enhancing its advantage in the category of high anonymity and latency-tolerant use cases.

Despite the fact that Nym supports a communication fully inside the network, it is not yet clear fully how to set up such a communication, and there were documentation changes recently. It would be beneficial if, for example, a whistleblowing platform could have easily set up a service that allows users to access it without leaving the network, providing built-in storage and, at the same time, increases user anonymity as the service receives cover traffic. Nym could also have provided a simple list of available services within the network if the service provider wishes for his service to be publicly available.

8.6. Summary

This chapter reviewed future directions for ACNs, based on previous chapters, especially the comparative analysis. The key common points, such as growing the user base, getting ready for quantum-safe cryptography, and pushing back against increasing censorship, apply to every ACN, no matter the architecture. Then, each network got its own list of next steps, tailored to the area where it currently leads: web browsing along with infrastructure security and resilience for Tor, file sharing for I2P, low latency for Lokinet, and strong anonymity for Nym. Each of these sections ties directly back to these ACNs' limitations and strengths. In the broader context of the thesis, this chapter points out the practical steps that need to be taken next if the aim is to build anonymous communication networks that are safer and more usable.

9. EDUCATIONAL DEMONSTRATOR

This chapter presents an educational demonstrator designed to help students understand how anonymous communication networks work in practice and to illustrate the effects of the analysis performed in this work. The demonstrator combines theoretical background with practical exercises to highlight key differences between various anonymity solutions.

A key goal of the demonstrator is to show that there is no single universal ACN suitable for all use cases. Instead, the choice of the network should be guided by the specific requirements of a given scenario. Through hands-on tasks such as clearnet browsing, hidden service setup, and anonymous file sharing, students will gain insight into which networks are better suited for particular applications and why.

This demonstrator was created to show students the practical effects of the comparative analysis from Chapter 7, where it was shown that there is no single universal ACN for all use cases. The aim is to let students see in practice how the networks differ and why these trade-offs matter. The choice of networks and exercises reflects the main categories discussed in the analysis. Students are expected to know basic Linux and networking; technical setup is described later. Reading a theoretical introduction to the laboratory is strongly advised. The demonstrator is included here to turn theory into practical skills and highlight the most important findings of the thesis. This chapter first describes the scope of the laboratory, then the course of the laboratory, design considerations followed by the laboratory setup, and finally points to the theoretical introduction and exercises in Appendix C: Educational demonstrator: Theory and practical exercises.

9.1. Scope of the laboratory

This laboratory is designed to help students understand how anonymous communication networks (ACNs) work in practice, what their main features are, and where they can be used. The lab covers both the theory and practical exercises to show differences between various anonymity solutions, starting from basic proxies and moving to more advanced systems. Students will learn about the difference between anonymity by policy and anonymity by design and why this matters, the basics of Mix-nets and onion routing and how they affect anonymity, the most popular ACNs today: Tor, I2P, Lokinet, and Nym: How do they work, what they are good at, and what problems they face, as well as the real use cases for these systems, including web browsing, hidden services, and file sharing.

9.2. Course of the laboratory

During the lab, students will go through five main practical exercises:

1. Anonymity by policy vs. anonymity by design
2. Mix-nets and onion routing

3. Browsing clearnet with ACNs
4. Hidden Services
5. File sharing with ACNs

In the first exercise, students will see how single-hop proxies work, check the logs, and notice the weaknesses. Then, they will switch to an ACN setup and compare the difference. After finishing this exercise, students should be able to tell the difference between these approaches, along with the drawbacks and potential threats to anonymity by policy design.

In the second exercise, students will send messages using onion routing and Mix-nets and look at the results. They will see how Mix-nets provide better protection against traffic analysis, but with the cost of a larger delay. After finishing this exercise, students should be able to tell the difference between ACNs based on Mix-nets and onion routing, along with the advantages and disadvantages of each design. Students should also be able to name several use cases for each design.

In the third exercise, students will try to reach regular websites using ACNs, comparing the support of the clearnet for each ACN. After finishing this exercise, students should be able to configure ACNs to browse clearnet. Students should be able to choose a preferred ACN for this application area.

In the fourth exercise, students will set up a Tor hidden service, first without and then with authorisation keys. They'll see how this allows access to web pages and SSH services securely. After finishing this exercise, students should be able to configure hidden services within the Tor network. A student should be able to name their advantages and potential use cases.

In the last exercise, students will measure the download speed of a large file shared over Tor via OnionShare and I2P via I2PSnark. They will compare which is faster and which hides traffic patterns better. After finishing this exercise, students should be able to point out the differences between Tor and I2P in terms of file sharing. The student should point out a preferable ACN for this use case.

9.3. Design considerations

To maximise educational value within the time constraints of a typical laboratory session (1.5 to 2 hours), the first two exercises use simplified simulators rather than real-world anonymous communication networks. This approach was chosen to more clearly visualize the concepts of anonymity by policy versus anonymity by design, as well as to illustrate the core differences between Mix-nets and onion routing architectures without the overhead of complex setups. Exercises 3 to 5 focus on practical tasks with actual ACNs (Tor, I2P, and Lokinet), chosen for their relevance to real-world applications and their significance in a given category, as proven in the analysis. Not all networks or categories could be included; for example, Nym was omitted due to its commercial distribution model (NymVPN as a paid service), and latency was only compared at the architectural level (Mix-nets vs. onion routing), as a detailed evaluation of minor architectural differences between onion routing-based solutions would be less instructive for the demonstrator's goals. Similarly, the hidden services exercise is focused exclusively on Tor, as it turned out to be the most favourable solution in the comparative analysis

for the infrastructure security and resilience-based use cases. These project decisions ensure that the demonstrator provides focused, high-impact learning experiences that align closely with the findings of the analysis, even if some technical details and alternative solutions were necessarily excluded due to practical considerations.

9.4. Laboratory setup

Laboratory should be performed on a Debian-based Linux distribution, ideally on Debian 12/Bookworm. Each workstation should have Python, Docker, Firefox, any non-Firefox browser (for example, DuckDuckGo browser, Opera or Brave), OnionShare, I2P router, and Lokinet with GUI installed. I2P router and Lokinet should be turned on in order to establish necessary tunnels and paths in advance. I2P's bandwidth parameters should be changed in both router client and I2PSnark client in order to avoid throttling.

9.5. Exercises

Practical exercises, along with the theoretical introduction, are included in Appendix C: Educational demonstrator: Theory and practical exercises. Files

9.6. Summary

This chapter presented an educational demonstrator that combines theoretical background and practical exercises to help students understand how anonymous communication networks (ACNs) work in practice. The chapter described the scope of the laboratory, the main exercises and the technical setup, with more details and step-by-step instructions provided in Appendix C: Educational demonstrator: Theory and practical exercises.

The main goal of the demonstrator is to show that there is no single universal ACN for all use cases, as established in the comparative analysis in Chapter 7. The exercises and their design reflect the most important categories identified in the analysis, allowing students to directly observe the trade-offs and limitations discussed earlier in the thesis.

The demonstrator can be used as a teaching tool in courses on network security, privacy, or related topics, or as a self-study guide for anyone interested in anonymous communication technologies. By working through the exercises, students gain practical skills, see how theoretical concepts apply in real-world scenarios, and learn to evaluate which ACN is best suited for specific applications. In the context of the thesis, this chapter translates the findings of the multi-criteria analysis into an educational resource, highlighting the practical impact of the research.

10. SUMMARY

The main goal of this thesis was to determine whether there is a universal anonymous communication network exists; one that is optimal for all use case scenarios. The analysis conducted here, based on both the technical literature and the experimental results, leaves no room for doubt: there is no such universal solution. The choice of an anonymous communication network (ACN) always comes down to the specific requirements and threat model of the intended application.

The thesis began by outlining the background and motivation for studying ACN in the Introduction chapter, defining the scope of the work, and explaining why this topic is relevant.

The Theoretical introduction chapter established the necessary knowledge foundations, necessary to understand the insights, strengths, and trade-offs of anonymous communication systems.

Next, the Overview of the anonymous communication networks chapter presented a detailed overview of the major types of ACNs in use today, introducing the main architectural approaches, such as single-hop proxies, Mix-nets, DC-nets, anonymous publication systems, and onion routing, offering a concise description of the networks that were analysed in detail later in the thesis. This chapter set the scene for a deeper analysis, ensuring that subsequent comparisons were provided with an appropriate context.

The Related work chapter provided a critical overview of existing research, highlighting the lack of practical, use case-driven comparisons between major networks.

The chapter Use cases and application areas mapped out the real-world use cases and application areas of ACNs, categorised them according to their technical requirements, and established clear criteria for evaluation. This classification formed the basis for the subsequent analysis and ensured that the comparative analysis was grounded in practical reality.

The Threats, attacks and limitations chapter detailed the range of adversarial strategies that ACNs must defend against, from passive and active attacks to broader limitations such as economic sustainability and the ongoing arms race with censorship. Understanding these threats is essential for any serious evaluation of network suitability.

The core of the thesis was the Multi-criteria comparative analysis chapter. Here, the major ACNs today: Tor, I2P, Lokinet, and Nym, were evaluated side-by-side, both through literature-based comparison and original experimental measurements. Each network was assessed in the context of the defined use case categories. The results show that each ACN has its own strengths: Tor is most effective for web browsing and infrastructure security, I2P is best for file sharing, Lokinet performs the best in low-latency use cases, and Nym offers the highest levels of anonymity. There is no configuration or network that achieves all goals simultaneously.

The chapter Future directions built on the findings of the comparative analysis to suggest concrete next steps both for the field as a whole and for each network individually. Key recommendations

include general guidance for all major ACNs, as well as targeted improvements for each ACN separately, based on their strengths and weaknesses that were defined in the comparative analysis.

A practical outcome of this work is the educational demonstrator described in the Educational demonstrator chapter. The demonstrator puts the conclusions of the thesis into practice, enabling students to observe the strengths and weaknesses of each network in various scenarios through carefully designed laboratory exercises. This bridges the gap between theory and hands-on understanding.

To ensure continued progress in the field of anonymous communication networks and their usability, it is necessary to maintain an ongoing and critical evaluation of new ACN designs as they emerge, extending the comparative analysis to next-generation systems, and updating the set of practical use cases as technology and threats evolve. While this thesis provides a decent overview as of today, the networks will most definitely evolve and most likely new ones will appear, along with newer and improved designs, which will also need to be evaluated in terms of their usability in various use case scenarios.

In terms of the development of ACNs, research should mainly focus on following the directions defined in the Future directions chapter, as well as addressing the issues defined in the Threats, attacks and limitations chapter.

To conclude, this thesis demonstrates, both in theory and in practice, that there is no universal anonymous communication network suitable for every use case. The optimal choice always depends on a clear understanding of the technical requirements, threat models, and inherent limitations of each design. The results and tools presented here provide a foundation for both further research and practical deployment of ACNs, making it possible to match the right network to the right scenario, rather than chasing a non-existent "one-size-fits-all" solution.

BIBLIOGRAPHY

[1] A. Pfitzmann and M. Köhntopp, "Anonymity, unobservability, and pseudonymity - A proposal for terminology," in *Designing Privacy Enhancing Technologies, International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, July 25-26, 2000, Proceedings* (H. Federrath, ed.), vol. 2009 of *Lecture Notes in Computer Science*, pp. 1–9, Springer, 2000.

[2] D. Cole, "We kill people based on metadata," *The New York Review of Books*, May 2014.

[3] European Data Protection Supervisor, "Data protection." `https://www.edps.europa.eu/data-protection/data-protection_en`. Accessed: 2025-04-20.

[4] M. Eddy, "7 vpn services found recording user logs despite 'no-log' pledge," *PCMag*, July 2020. Accessed: 2025-04-20.

[5] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, p. 149–160, August 2001.

[6] P. Maymounkov and D. Eres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Lecture Notes in Computer Science 2429*, vol. 2429, 04 2002.

[7] S. Crocker, "Host software." RFC 1, Apr. 1969.

[8] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, p. 84–90, February 1981.

[9] M. M. Helmers, "E-mail discussion groups and academic culture," *CMC Magazine*, September 1997. Accessed: 2025-04-20.

[10] D. Chaum, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *Journal of Cryptology*, vol. 1, p. 65–75, January 1988.

[11] M. Reed, P. Syverson, and D. Goldschlag, "Anonymous connections and onion routing," *IEEE Journal on Selected Areas in Communications*, vol. 16, p. 482–494, May 1998.

[12] D. Goldschlag, M. Reed, and P. Syverson, "Onion routing," *Communications of the ACM*, vol. 42, p. 39–41, February 1999.

[13] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[14] K. Jefferys, S. Harman, J. Ross, and P. McLean, "Loki: Private transactions, decentralised communication," Tech. Rep. Version 3, Loki Project, July 2018. Accessed: 2025-04-20.

[15] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis, "The loopix anonymity system," in *26th USENIX Security Symposium (USENIX Security 17)*, (Vancouver, BC), pp. 1199–1216, USENIX Association, August 2017.

[16] C. Diaz, H. Halpin, and A. Kiayias, "The nym network: The next generation of privacy infrastructure," Tech. Rep. 1.0, Nym Technologies SA, February 2021. Accessed: 2025-04-20.

[17] J. Boyan, "The anonymizer: Protecting user privacy on the web," *CMC Magazine*, September 1997. Accessed: 2025-04-20.

[18] Primepq, "Red de mezcla." Wikimedia Commons, 2008. Licensed under Creative Commons Attribution–ShareAlike 3.0.

[19] P. Baran, "On distributed communications: Ix. security, secrecy, and tamper-free considerations," Research Memorandum RM-3765-PR, RAND Corporation, Santa Monica, CA, 1964. Accessed: 2025-04-20.

[20] C. Gulcu and G. Tsudik, "Mixing e-mail with babel," in *Proceedings of Internet Society Symposium on Network and Distributed Systems Security*, NDSS-96, p. 2–16, IEEE Comput. Soc. Press, 1996.

[21] D. Kesdogan, J. Egner, and R. Büschkes, "Stop-and-go MIXes: Providing probabilistic anonymity in an open system," in *Proceedings of Information Hiding Workshop (IH 1998)*, Springer-Verlag, LNCS 1525, 1998.

[22] A. Pfitzmann, B. Pfitzmann, and M. Waidner, *ISDN-Mixes: Untraceable Communication with Very Small Bandwidth Overhead*, p. 451–463. Springer Berlin Heidelberg, 1991.

[23] O. Berthold, H. Federrath, and S. Köpsell, *Web MIXes: A System for Anonymous and Unobservable Internet Access*, p. 115–129. Springer Berlin Heidelberg, 2001.

[24] M. Rennhard and B. Plattner, "Introducing morphmix: peer-to-peer based anonymous internet usage with collusion detection," in *Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, CCS02, p. 91–102, ACM, November 2002.

[25] M. J. Freedman and R. Morris, "Tarzan: a peer-to-peer anonymizing network layer," in *Proceedings of the 9th ACM conference on Computer and communications security*, CCS02, ACM, November 2002.

[26] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: design of a type iii anonymous remailer protocol," in *Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405)*, SECPRI-03, p. 2–15, IEEE Comput. Soc, 2003.

[27] D. Chaum, *Blind Signature System*, pp. 153–153. Boston, MA: Springer US, 1984.

[28] C. Diaz and A. Serjantov, "Generalising mixes," in *Proceedings of Privacy Enhancing Technologies workshop (PET 2003)* (R. Dingledine, ed.), pp. 18–31, Springer-Verlag, LNCS 2760, March 2003.

[29] A. Serjantov, R. Dingledine, and P. Syverson, *From a Trickle to a Flood: Active Attacks on Several Mix Types*, p. 36–52. Springer Berlin Heidelberg, December 2002.

[30] G. Danezis and B. Laurie, "Minx: a simple and efficient anonymous packet format," in *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, CCS04, p. 59–65, ACM, October 2004.

[31] E. Shimshock, M. Staats, and N. Hopper, *Breaking and Provably Fixing Minx*, p. 99–114. Springer Berlin Heidelberg, July 2008.

[32] G. Danezis and I. Goldberg, "Sphinx: A compact and provably secure mix format," in *2009 30th IEEE Symposium on Security and Privacy*, p. 269–282, IEEE, May 2009.

[33] C. Diaz, S. J. Murdoch, and C. Troncoso, *Impact of Network Topology on Anonymity and Overhead in Low-Latency Anonymity Networks*, p. 184–201. Springer Berlin Heidelberg, 2010.

[34] G. Danezis and L. Sassaman, "Heartbeat traffic to counter (n-1) attacks," in *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2003)*, October 2003.

[35] Katzenpost Project, "Katzenpost: Anonymous communication for everyone," 2025. Accessed: 2025-04-20.

[36] E. J. Infeld, D. Stainton, L. Ryge, and T. Hacker, "Echomix: A strong anonymity system with messaging," *arXiv preprint arXiv:2501.02933*, 2025. Accessed: 2025-06-01.

[37] J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich, "Vuvuzela: scalable private messaging resistant to traffic analysis," in *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, p. 137–152, ACM, October 2015.

[38] A. Kwon, D. Lazar, S. Devadas, and B. Ford, "Riffle: An efficient communication system with strong anonymity," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, p. 115–134, December 2015.

[39] D. Chaum, D. Das, F. Javani, A. Kate, A. Krasnova, J. De Ruiter, and A. T. Sherman, "cmix: Mixing with minimal real-time asymmetric cryptographic operations," in *Applied Cryptography and Network Security* (D. Gollmann, A. Miyaji, and H. Kikuchi, eds.), pp. 557–578, Springer International Publishing, 2017.

[40] PauAmma, "Dining cryptographers." Wikimedia Commons, 2013. Licensed under Creative Commons Attribution–ShareAlike 3.0 Unported (CC BY-SA 3.0).

[41] M. Waidner and B. Pfitzmann, *The Dining Cryptographers in the Disco: Unconditional Sender and Recipient Untraceability with Computationally Secure Serviceability*, p. 690–690. Springer Berlin Heidelberg, November 1990.

[42] E. G. Sirer, M. Polte, and M. Robson, "Cliquenet: A self-organizing, scalable, peer-to-peer anonymous communication substrate," Technical Report TR2001, Cornell University, Ithaca, NY, 2001. Accessed: 2025-04-20.

[43] Electronic Privacy Information Center, "Carnivore surveillance system." `https://epic.org/privacy/carnivore/`. Accessed: 2025-04-20.

[44] S. Goel, M. Robson, M. Polte, and E. G. Sirer, "Herbivore: A Scalable and Efficient Protocol for Anonymous Communication," Tech. Rep. 2003-1890, Cornell University, Ithaca, NY, February 2003.

[45] S. Dolev and R. Ostrobsky, "Xor-trees for efficient anonymous multicast and reception," *ACM Transactions on Information and System Security*, vol. 3, p. 63–84, May 2000.

[46] P. Golle and A. Juels, *Dining Cryptographers Revisited*, p. 456–473. Springer Berlin Heidelberg, 2004.

[47] H. Corrigan-Gibbs and B. Ford, "Dissent: accountable anonymous group messaging," in *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, p. 340–350, ACM, October 2010.

[48] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson, "Dissent in numbers: Making strong anonymity scale," in *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 179–192, October 2012. Accessed: 2025-04-20.

[49] H. Corrigan-Gibbs, D. Boneh, and D. Mazières, "Riposte: An anonymous messaging system handling millions of users," in *2015 IEEE Symposium on Security and Privacy*, pp. 321–338, 2015.

[50] R. Anderson, "The eternity service," in *Proceedings of Pragocrypt '96*, 1996.

[51] A. Back, "The eternity service," *Phrack Magazine*, vol. 7, September 1997. Accessed: 2025-04-20.

[52] I. Goldberg and D. Wagner, "TAZ servers and the rewebber network: Enabling anonymous publishing on the world wide web," *First Monday*, vol. 3, August 1998.

[53] R. Dingledine, M. J. Freedman, and D. Molnar, "The free haven project: Distributed anonymous storage service," in *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability* (H. Federrath, ed.), Springer-Verlag, LNCS 2009, July 2000.

[54] M. Waldman, A. Rubin, and L. Cranor, "Publius: A robust, tamper-evident, censorship-resistant and source-anonymous web publishing system," in *Proceedings of the 9th USENIX Security Symposium*, pp. 59–72, August 2000.

[55] M. Waldman and D. Mazières, "Tangler: a censorship-resistant publishing system based on document entanglements," in *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS 2001)*, pp. 126–135, November 2001.

[56] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability* (H. Federrath, ed.), vol. 2009 of *Lecture Notes in Computer Science*, pp. 46–66, Springer-Verlag, LNCS 2009, July 2000.

[57] XcepticZP, "Freenet request sequence zp." Wikimedia Commons, 2009. Licensed under Creative Commons Attribution–ShareAlike 3.0 Unported (CC BY-SA 3.0).

[58] Freenet Project Inc., "Freenet manual." `https://freenet.org/resources/manual/`, 2024. Accessed: 2025-04-20.

[59] K. Bennett, T. Stef, C. Grothoff, T. Horozov, and I. Patrascu, "The gnet whitepaper," *unknown*, June 2002.

[60] W. Dai, "Pipenet 1.0." Post to Cypherpunks mailing list, January 1998.

[61] P. Boucher, A. Shostack, and I. Goldberg, "Freedom systems 2.0 architecture," white paper, Zero Knowledge Systems, Inc., December 2000.

[62] J. Tracey, "Onion routing: An exploration of the tor anonymity network." Online thesis, 2024. Accessed: 2025-06-14.

[63] J. T. (Tga.D), "Tor circuit diagram." Wikimedia Commons, 2024. Originally published in the author's thesis. Licensed under Creative Commons Attribution–ShareAlike 4.0 International (CC BY-SA 4.0).

[64] The Tor Project, Inc., "Tor specifications." Online documentation. Licensed under Creative Commons Attribution 4.0 International (CC BY 4.0).

[65] J. Appelbaum and A. Muffett, "The ".onion" special-use domain name." RFC 7686, October 2015. Accessed: 2025-04-20.

[66] DigiCert, "Ordering a .onion certificate from digicert," December 2015. Accessed: 2025-04-20.

[67] Hellenic Academic Research Institutions Certification Authority (HARICA), "Dv certificates for onion websites," April 2021. Accessed: 2025-04-20.

[68] T. T. Project, "Get bridges." `https://bridges.torproject.org/`. Accessed: 2025-06-14.

[69] S. Frolov and E. Wustrow, "HTTPT: A Probe-Resistant proxy," in *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*, USENIX Association, August 2020.

[70] N. P. Hoang, P. Kintis, M. Antonakakis, and M. Polychronakis, "I2p metrics portal." `https://i2p-metrics.np-tokumei.net/overview`. Accessed: 2025-04-20.

[71] N. P. Hoang, P. Kintis, M. Antonakakis, and M. Polychronakis, "An empirical study of the i2p anonymity network and its censorship resistance," in *Proceedings of the Internet Measurement Conference 2018*, IMC '18, (New York, NY, USA), pp. 379–392, ACM, 2018.

[72] I2P Team, "I2p network topology." Licensed under Creative Commons Attribution–ShareAlike 4.0 International.

[73] I2P Team, "End-to-end encryption diagram." I2P documentation, 2025. Licensed under Creative Commons Attribution–ShareAlike 4.0 International.

[74] A. Linton, "Onion requests: Session's new message routing solution." `https://getsession.org/onion-requests-session-new-message-routing-solution`, January 2020. Accessed: 2025-04-20.

[75] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig, "Hornet: High-speed onion routing at the network layer," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS'15, p. 1441–1454, ACM, October 2015.

[76] C. Chen, D. E. Asoni, A. Perrig, D. Barrera, G. Danezis, and C. Troncoso, "Taranet: Traffic-analysis resistant anonymity at the network layer," in *2018 IEEE European Symposium on Security and Privacy (EuroSamp;P)*, p. 137–152, IEEE, April 2018.

[77] J. Sankey and M. Wright, *Dovetail: Stronger Anonymity in Next-Generation Internet Routing*, p. 283–303. Springer International Publishing, 2014.

[78] A. Montieri, D. Ciuonzo, G. Aceto, and A. Pescape, "Anonymity services tor, i2p, jondonym: Classifying in the dark (web)," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, p. 662–675, May 2020.

[79] A. Montieri, D. Ciuonzo, G. Bovenzi, V. Persico, and A. Pescape, "A dive into the dark web: Hierarchical traffic classification of anonymity tools," *IEEE Transactions on Network Science and Engineering*, vol. 7, p. 1043–1054, July 2020.

[80] K. Shahbar and A. N. Zincir-Heywood, "Packet momentum for identificationof anonymity networks," *Journal of Cyber Security and Mobility*, vol. 6, no. 1, p. 27–56, 2017.

[81] K. Shahbar, *Analysis of Multilayer-Encryption Anonymity Networks*. PhD thesis, Dalhousie University, October 2017.

[82] A. Ali, M. Khan, M. Saddique, U. Pirzada, M. Zohaib, I. Ahmad, and N. Debnath, "Tor vs i2p: A comparative study," in *Proceedings of the 2016 IEEE International Conference on Industrial Technology (ICIT)*, March 2016.

[83] S. Winkler and S. Zeadally, "An analysis of tools for online anonymity," *International Journal of Pervasive Computing and Communications*, vol. 11, p. 436–453, November 2015.

[84] T. Ries, A. Panchenko, R. State, and T. Engel, "Comparison of Low-Latency Anonymous Communication Systems: Practical Usage and Performance," in *Proceedings of the Ninth Australasian Information Security Conference (AISC 2011)*, CRPIT, Vol. 116, (Perth, Australia), pp. 77–86, Australian Computer Society, 2011. Available via ORBilu repository.

[85] N. P. Hoang, J. Dalek, M. Crete-Nishihata, N. Christin, V. Yegneswaran, M. Polychronakis, and N. Feamster, "GFWeb: Measuring the great firewall's web censorship at scale," in *Proceedings of the 33rd USENIX Security Symposium (Sec '24)*, USENIX Association, August 2024.

[86] I. Zahorsky, "Tor, anonymity, and the arab spring: An interview with jacob appelbaum," *Peace and Conflict Monitor*, August 2011. Accessed: 2025-04-20.

[87] B. Okunoye, "Censored continent: Understanding the use of tools during internet censorship in africa: Cameroon, nigeria, uganda and zimbabwe as case studies," Tech. Rep. 2020-07-001, OTF, The Tor Project, July 2020. Accessed: 2025-04-20.

[88] Freedom House, "Freedom on the net: Country scores," 2025. Accessed: 2025-04-20.

[89] Facebook, "Title of the note," Year of Publication. Accessed: 2025-04-20.

[90] M. Braga, "Why facebook is making it easier to log on with tor–and other companies should, too," *Fast Company*, November 2014. Accessed: 2025-04-20.

[91] I. Borogan and A. Soldatov, "Russia's bankers become secret policemen," *Center for European Policy Analysis (CEPA)*, November 2023. Accessed: 2025-04-20.

[92] GlobaLeaks Project, "Globaleaks: Free and open-source whistleblowing software," 2025. Accessed: 2025-04-20.

[93] Earth League International, "Wildleaks: The world's first whistleblowing initiative dedicated to environmental crime," 2025. Accessed: 2025-04-20.

[94] Microsoft Corporation, "Microsoft security advisory 2798897: Fraudulent digital certificates could allow spoofing," January 2013. Accessed: 2025-04-20.

[95] U.S. Department of Justice, "Zyprexa scandal." `https://www.justice.gov/archive/opa/pr/2009/January/09-civ-038.html`, January 2009. Accessed: 2025-06-14.

[96] BBC News, "Taliban suspends chess over gambling concerns," *BBC News*, May 2025.

[97] International Telecommunication Union, "Itu-t recommendation g.114: One-way transmission time," May 2003. Accessed: 2025-04-20.

[98] P. K. Sharma, S. Chaudhary, N. Hassija, M. Maity, and S. Chakravarty, "The road not taken: Re-thinking the feasibility of voice calling over tor," 2020.

[99] International Telecommunication Union, "Itu-t recommendation g.729: Coding of speech at 8 kbit/s using conjugate-structure algebraic-code-excited linear prediction (cs-acelp)," March 1996. Accessed: 2025-04-20.

[100] Microsoft Corporation, "How much bandwidth does skype need?," 2025. Accessed: 2025-04-20.

[101] D. Das, S. Meiser, E. Mohammadi, and A. Kate, "Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency - choose two," in *2018 IEEE Symposium on Security and Privacy (SP)*, p. 108–126, IEEE, May 2018.

[102] R. Dingledine, M. J. Freedman, D. Hopwood, and D. Molnar, "A Reputation System to Increase MIX-net Reliability," in *Proceedings of Information Hiding Workshop (IH 2001)* (I. S. Moskowitz, ed.), pp. 126–141, Springer-Verlag, LNCS 2137, April 2001.

[103] N. Gelernter, A. Herzberg, and H. Leibowitz, *Two Cents for Strong Anonymity: The Anonymous Post-office Protocol*, p. 390–412. Springer International Publishing, 2018.

[104] Internet Live Stats, "Total number of websites," 2025. Accessed: 2025-04-20.

[105] The Tor Project, "Tor metrics," 2025. Accessed: 2025-04-20.

[106] K. Loesing, S. J. Murdoch, and R. Dingledine, "A case study on measuring statistical data in the Tor anonymity network," in *Proceedings of the Workshop on Ethics in Computer Security Research (WECSR 2010)*, LNCS, Springer, January 2010.

[107] B. Kahle, "Learning from cyberattacks." `https://blog.archive.org/2024/11/14/learning-from-cyberattacks/`, November 2024. Accessed: 2025-04-20.

[108] G. Owen and N. Savage, "Empirical analysis of tor hidden services," *IET Information Security*, vol. 10, p. 113–118, May 2016.

[109] The Tor Project, "Who uses tor?," 2019. Accessed: 2025-04-20.

[110] F. A. P. Petitcolas, *Kerckhoffs' Principle*, p. 675–675. Springer US, 2011.

[111] R. Jansen, F. Tschorsch, A. Johnson, and B. Scheuermann, "The sniper attack: Anonymously deanonymizing and disabling the Tor network," in *Proceedings of the Network and Distributed Security Symposium - NDSS '14*, IEEE, February 2014.

[112] L. Oldenburg, G. Acar, and C. Diaz, "From "onion not found" to guard discovery," *Proceedings on Privacy Enhancing Technologies*, vol. 2022, p. 522–543, November 2021.

[113] R. Dingledine, "The tor censorship arms race: The next chapter." Presentation at DEF CON 27, August 2019. Accessed: 2025-04-20.

[114] Session, "Migrating from the oxen network to session network," May 2025. Accessed: 2025-06-01.

[115] YouTube, "Live encoder settings, bitrates, and resolutions," 2024. Accessed: 2025-06-01.

[116] Y. Liu, J. Prince, and S. Wallsten, "Distinguishing bandwidth and latency in households' willingness-to-pay for broadband internet speed," *Information Economics and Policy*, vol. 45, pp. 1–15, 2018.

LIST OF FIGURES

LIST OF TABLES

APPENDIX A: BASH SCRIPTS FOR HIDDEN SERVICES SETUP

Tor

This script automates the installation and configuration of the Tor service on a Debian-based system. It sets up two separate onion services, forwarding traffic from the Tor network to local port 2135 for latency measurements. The script provides the proper permissions for the hidden service directories, updates the Tor configuration, and restarts the Tor service to apply the changes. After execution, the generated .onion addresses for the onion service is retrieved. Troubleshooting can be done via system logs if needed.

```
#!/bin/bash

CPU_ARCHITECTURE=$(dpkg --print-architecture)
DISTRIBUTION=$(lsb_release -c | awk '{print $2}')

apt update
apt install apt-transport-https

tee /etc/apt/sources.list.d/tor.list > /dev/null <<EOF
   deb      [signed-by=/usr/share/keyrings/deb.torproject.org-keyring.gpg]
        https://deb.torproject.org/torproject.org $DISTRIBUTION main
   deb-src [signed-by=/usr/share/keyrings/deb.torproject.org-keyring.gpg]
        https://deb.torproject.org/torproject.org $DISTRIBUTION main
EOF

apt update
apt install gnupg
wget -qO-
    https://deb.torproject.org/torproject.org/A3C4F0F979CAA22CDBA8F512EE8CBC9E886DDD89.asc
    | gpg --dearmor | tee /usr/share/keyrings/deb.torproject.org-keyring.gpg >/dev/null

apt update
apt install tor deb.torproject.org-keyring

mkdir /var/lib/tor/tcp_latency_onion_service
mkdir /var/lib/tor/tcp_throughput_onion_service

chown -R debian-tor /var/lib/tor/
chmod 700 -R /var/lib/tor/

cat <<EOF >> /etc/tor/torrc
HiddenServiceDir /var/lib/tor/tcp_latency_onion_service/
HiddenServicePort 2135 127.0.0.1:2135
EOF
```

```
systemctl restart tor

cat /var/lib/tor/tcp_latency_onion_service/hostname # get the onion address for the
    latency app
# journalctl -e -u tor@default # optionally to troubleshoot
```

I2P

This script handles the installation of the I2P software on a Debian-based system. It adds the official I2P repository, imports the trusted GPG key, and installs both the core I2P package and its keyring.

Note: Unlike Tor, I2P does not automatically create tunnels for the application through script execution. Tunnels must be manually configured after script execution, which can be most easily accomplished using the web-based I2P Router Console, which is typically accessible at http://localhost:7657.

```
apt-get update
apt-get install apt-transport-https lsb-release curl

echo "deb [signed-by=/usr/share/keyrings/i2p-archive-keyring.gpg] https://deb.i2p.net/
    $(lsb_release -sc) main" > /etc/apt/sources.list.d/i2p.list

curl -o i2p-archive-keyring.gpg https://geti2p.net/_static/i2p-archive-keyring.gpg

gpg --keyid-format long --import --import-options show-only --with-fingerprint
    i2p-archive-keyring.gpg

cp i2p-archive-keyring.gpg /usr/share/keyrings

apt-get update
apt-get install i2p i2p-keyring
```

Lokinet

This script installs and configures Lokinet on a Debian-based system. After installation, it updates the Lokinet configuration to set a private key and assigns a virtual interface IP. Lokinet differs from Tor and I2P in that it makes all listening ports accessible over the network, meaning that any service that is run locally (on any port) can be reached by the Lokinet address. After restarting the service, the Lokinet address is obtained.

```
apt update
apt install gnupg
curl -so /etc/apt/trusted.gpg.d/oxen.gpg https://deb.oxen.io/pub.gpg
```

```
echo "deb https://deb.oxen.io $(lsb_release -sc) main" | sudo tee
    /etc/apt/sources.list.d/oxen.list
apt update
apt install lokinet

sed -i 's|#keyfile=.*|keyfile=/var/lib/lokinet/snappkey.private|' /etc/loki/lokinet.ini
sed -i 's|#ifaddr=.*|ifaddr=172.16.0.1/16|' /etc/loki/lokinet.ini
systemctl restart lokinet

host -t cname localhost.loki 127.3.2.1 # get the loki address
```

APPENDIX B: PYTHON APPLICATION FOR MEASURING RTT AND JITTER

Server

The following script is the server component of the Python application to measure RTT and jitter. It listens on port 2135 for incoming TCP connections and handles client requests by echoing received data back to the sender. The server operates in a continuous loop, handling one client connection at a time.

```
import socket

HOST = '0.0.0.0'
PORT = 2135
BUFFER_SIZE = 1024

def run_server() -> None:
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((HOST, PORT))
    server_socket.listen()
    print(f"TCP latency server is running on port {PORT}...")
    while True:
        client_socket, client_addr = server_socket.accept()
        print(f"Connection from {client_addr}")
        handle_client(client_socket, client_addr)

def handle_client(client_socket, client_addr) -> None:
    try:
        while True:
            data = client_socket.recv(BUFFER_SIZE)
            if not data:
                break
            client_socket.sendall(data)
    except Exception as e:
        print(f"Error: {e}")
    finally:
        client_socket.close()
        print(f"Connection with {client_addr} closed")

if __name__ == "__main__":
    run_server()
```

Client

This script is the client component of the Python application. It connects to the server (localhost for I2P client tunnel, it should be replaced with appropriate onion address for Tor or

loki address for Lokinet) on port 2135 and performs a series of latency measurements by sending timestamped messages and calculating the round-trip time (RTT) for each. The script computes and displays the average RTT, jitter, and packet loss statistics. Although this specific version of the application is designed for reliable in-order delivery via TCP, the client is implemented to handle the potential out-of-order messages. This design choice makes it easy to adapt the script for UDP-based communication, requiring only minor adjustments to the socket type and the socket-related communication logic.

```
import socket
import time
import statistics

HOST = "127.0.0.1"
PORT = 2135
TIMEOUT = 10
TRIALS = 50

def run_client() -> None:
    time_pairs = {}  # key: send_time (float), value: recv_time or None
    try:
        client_socket = establish_tcp_socket()
    except Exception as e:
        print(f"Failed to connect: {e}")
        return
    perform_measurements(time_pairs, client_socket)
    client_socket.close()
    avg_rtt, jitter, packet_loss = calculate_results(time_pairs)
    print_results(avg_rtt, jitter, packet_loss)

def establish_tcp_socket() -> socket.socket:
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.settimeout(TIMEOUT)
    client_socket.connect((HOST, PORT))
    return client_socket

def perform_measurements(time_pairs, client_socket) -> None:
    for _ in range(TRIALS):
        send_time = time.time()
        send_bytes = str(send_time).encode()
        time_pairs[send_time] = None
        try:
            client_socket.sendall(send_bytes)
            data = client_socket.recv(1024)
            recv_time = time.time()
            try:
                echoed_send_time = float(data.decode())
```

```
            except ValueError:
                print("Received malformed response.")
                continue
            if echoed_send_time in time_pairs and time_pairs[echoed_send_time] is None:
                time_pairs[echoed_send_time] = recv_time
                rtt = recv_time - echoed_send_time
                print(f"RTT = {rtt:.6f} sec")
            else:
                print("Mismatched or duplicate response.")
        except socket.timeout:
            print("Timeout occurred.")
        except Exception as e:
            print(f"Communication error: {e}")
        time.sleep(1)

def calculate_results(time_pairs) -> tuple:
    rtts = [
        recv - sent
        for sent, recv in time_pairs.items()
        if recv is not None
    ]
    avg_rtt = sum(rtts) / len(rtts) if rtts else 0
    jitter = statistics.stdev(rtts) if len(rtts) > 1 else 0
    packet_loss = ((TRIALS - len(rtts)) / TRIALS) * 100
    return avg_rtt,jitter,packet_loss

def print_results(avg_rtt, jitter, packet_loss) -> None:
    print("\n--- Results ---")
    print(f"Average Latency: {avg_rtt * 1000:.2f} ms")
    print(f"Jitter: {jitter * 1000:.2f} ms")
    print(f"Packet Loss: {packet_loss:.2f}%")

if __name__ == "__main__":
    run_client()
```

APPENDIX C: EDUCATIONAL DEMONSTRATOR: THEORY AND PRACTICAL EXERCISES

This appendix contains the theory and a full set of practical laboratory exercises designed for the educational demonstrator. It begins with a short theoretical introduction to anonymous communication networks and then provides step-by-step instructions for each exercise. The tasks cover both the basics and more advanced use cases, effectively addressing the most important aspects of the thesis. Each exercise includes learning outcomes and questions to help students connect theory with hands-on experience.

Theoretical introduction

Anonymous communication networks are systems built to hide who is talking to whom. Firstly, deployed were simple solutions like proxies or remailers, where you had to trust the operator not to reveal your identity. This is called **anonymity by policy** - you are only anonymous if the operator adheres to the rules. History shows that this is not enough, as providers sometimes keep logs or are forced to give up user data. Therefore, these solutions cannot be treated as truly anonymous communication networks.

Anonymity by design is a different approach: the network is built so that no one can even find who the users are. This is achieved due to cryptography, usually joined by layered encryption, and careful design. This is the basis for modern ACNs.

The first idea for a strong anonymous network came from David Chaum, who described Mix-nets in the 1980s. A Mix-net network consists of mixes - dedicated servers that receive messages, batch, shuffle (mix), and encrypt them, so that it is hard to link the sender to the receiver. Their main drawback is that they are slow and not good for latency-vulnerable use cases like web browsing. Later, onion routing was invented for lower latency, thanks to its circuit-switched nature - sending data through an established bidirectional circuit, making it possible to browse the web comfortably while staying anonymous; however, this introduced a significant drawback in terms of traffic analysis vulnerability. Nonetheless, onion routing turned out to be a dominant strategy and is widely used today. Mix-nets seem to have been forgotten for some time; however, recently they are appearing more often in the literature, mostly thanks to the Loopix design that allows using Mix-net architecture with relatively low latency, although still higher than the one introduced by the onion routing-based solutions. Loopix has several interesting properties, including built-in message storage, reliable delivery, cover traffic and random, independent delays.

The four main ACNs used today are the following:

1. Tor
2. I2P

3. Lokinet
4. Nym

Tor is the most popular ACN today. It uses onion routing and thousands of volunteer relays. Tor is good for anonymous web browsing and running hidden websites that can optionally be protected with authorisation mechanisms. Tor is a perfect ACN for accessing normal websites (clearnet), but can be slow or blocked in some countries - that is why they introduced censorship circumvention mechanisms in terms of non-public relays or pluggable transports that can optionally obfuscate the traffic, making it more difficult to classify and block.

I2P is a fully peer-to-peer network using garlic routing, which is a variation of onion routing. I2P utilises unidirectional tunnels instead of the bidirectional circuits, and users have a lot of freedom in terms of customising tunnels' parameters: they can change the number of inbound/outbound tunnels for specific purposes, make them longer or shorter, or even choose variation of tunnel length. All of these aspects make traffic analysis more difficult as the users have the plausible deniability - it is hard to tell whether a certain user was receiving message for himself or was he just passing it through. I2P is good for internal hidden services and especially good for anonymous file sharing due to peer-to-peer file sharing protocols support and volume obfuscation techniques described earlier, but it is not the best tool for accessing the clearnet; although there are outproxy services for HTTP and HTTPS, however, they are not always reachable.

Lokinet is anonymous communication networks that similarly to Tor and I2P uses onion routing architecture, but contrary to it works on the network layer and therefore supports all IP-based protocols. It can in theory work with clearnet, the free exit nodes are often non-functioning but there is a possibility of renting dedicated exit nodes from private sellers or setting up custom exit nodes. The main advantage of the lower layer design is the smaller latency and jitter. Lokinet includes economic incentives for node runners.

Nym is a newest ACN, based on the previously mentioned Loopix. It is made to be resistant to powerful attackers and is best for people who need strong anonymity even if the network is slower. It is accessible as a mode in the NymVPN service, making this network the only officially paid network in this list. Nym includes economic incentives for node runners.

Exercise 1: Anonymity by policy and anonymity by design

Learning outcomes: After finishing this exercise, a student should be able to tell the difference between these approaches, along with the drawbacks and potential threats of anonymity by policy design.

1. Clone the repository from https://github.com/Karakean/Master-Thesis The files related to the educational demonstrator will be within the laboratory/ directory. It will be the working directory for Exercises 1 and 2.
2. Run routers as Docker containers:

```
docker compose up --build
```

3. Wait for the routers to start and then run client:

```
python3 client.py
```

4. Select "single hop proxy" mode and send the "Hello" message.
5. Enter the single-hop proxy:

```
docker exec -it single-hop-proxy /bin/bash
```

6. Read the output.log file.
7. **Question**: What do you see? Do you see any issues with this design? Do you know of any services that use this design?
8. Exit the client, run it again in the "anonymous communication network" mode, and send the same message as before.
9. Enter the entry node and read the output.log file similarly to how you did it with a single hop proxy.
10. Repeat for the middle node and the exit node.
11. **Question**: What is the difference compared to the single-hop proxy? What has improved and what can be worse? Why is the single hop proxy approach known as anonymity by policy and the ACN-based approach known as anonymity by design?

Exercise 2: Mix-nets and onion routing

Learning outcomes: After finishing this exercise, a student should be able to tell the difference between ACNs based on Mix-nets and onion routing, along with the advantages and disadvantages of each design. The student should also be able to name a few use cases for each design.

1. Run the routers analogously to the previous exercise.
2. Select "anonymous communication network" mode and send 10 messages with numbers from 1 to 10.
3. Enter each node and read the output files.
4. **Question**: Consider an attacker who is watching a destination that receives messages from an exit node along with all users that are sending messages to the anonymous communication network. Do you see any issues with the current design with regard to such an attacker?
5. Change ROUTER_MODE from "onion-routing" to "mix-net" in the .env file.
6. Re-run the docker containers.
7. Once again select the "anonymous communication network" mode and send 5 messages with numbers from 1 to 5.
8. Enter each node and read the output files.
9. **Question**: What do you see? Justify such a state.
10. Send 5 more messages from 5 to 10.
11. Enter each node and read the output files.

12. **Question**: What do you see? Justify such a state.
13. **Question**: What are the main differences between the Mix-nets and onion routing? Which design will be more suitable in case of asynchronous communication or whistleblowing and which will be more suitable in latency-vulnerable use cases such as web browsing or instant messaging?

Exercise 3: Browsing clearnet with ACNs

Learning outcomes: After finishing this exercise, a student should be able to configure ACNs for browsing clearnet. A student should be able to point out a preferable ACN for this application area.

1. Download and install Tor Browser https://www.torproject.org/download/
2. Turn on Tor Browser and enter https://example.com/
3. **Question**: Relays from which countries are used within your circuit that is used for this website? Were the actions necessary to visit the website complex and/or difficult?
4. Open the Firefox browser.
5. Open the *application menu* and go to *settings*.
6. Scroll down to the *Network Settings* and open them.
7. In the *Configure Proxy Access to the Internet* section choose radio button *Manual proxy configuration*
8. In the *HTTP Proxy* textbox write 127.0.0.1 with port 4444.
9. Click the checkbox *Also use this proxy for HTTPS*.
10. In the section *No proxy for* write localhost, 127.0.0.1
11. Save the changes.
12. Go to *about:config*.
13. Ensure that *media.peerConnection.ice.proxy_only* property is set to *true*.
14. Change property *keyword.enabled* to *false*.
15. Restart the Firefox browser.
16. Try to reach https://example.com/ from the Firefox browser. Try to refresh it a few times.
17. **Question**: Is it reachable? If not, what may be the reasons behind it? Were the actions necessary to visit the website complex and/or difficult?
18. Open the Lokinet GUI and make sure the state is *Connected to Lokinet*. If not, click the power button.
19. In order to make sure that the connection to Lokinet is established run:

    ```
    ping directory.loki
    ```

    If it works, then it means that the Lokinet works properly.
20. **Question**: Can ping work over Tor or I2P?
21. From the GUI turn on the VPN mode.
22. **Question**: Were the actions necessary to turn on the VPN mode complex and/or difficult? Does it work? If not, what may be the reasons behind it? What might help?

23. **Question**: Based on this exercise, which ACN do you find the most suitable for clearnet web browsing?

Exercise 4: Hidden services

Learning outcomes: After finishing this exercise, a student should be able to configure Hidden Services within the Tor network. A student should be able to name their advantages and potential use cases.

Exercise 4.1: Initial Tor setup

In order to create hidden services, you first need to install the necessary packages.

1. First update packages. Run:

```
apt update
```

2. Install necessary packages:

```
apt install -y apt-transport-https gnupg
```

3. Get the current Debian release. Run:

```
lsb_release -c | awk '{print $2}'
```

4. Create apt sources list for Tor:

```
tee /etc/apt/sources.list.d/tor.list > /dev/null <<EOF
  deb     [signed-by=/usr/share/keyrings/deb.torproject.org-keyring.gpg]
      https://deb.torproject.org/torproject.org <DISTRIBUTION> main
  deb-src [signed-by=/usr/share/keyrings/deb.torproject.org-keyring.gpg]
      https://deb.torproject.org/torproject.org <DISTRIBUTION> main
EOF
```

   Where <DISTRIBUTION> should be replaced with the output of the previous step.

5. Verify Tor packages:

```
wget -qO- https://deb.torproject.org/torproject.org/
    A3C4F0F979CAA22CDBA8F512EE8CBC9E886DDD89.asc | gpg --dearmor | tee /usr/share/keyrings
    /deb.torproject.org-keyring.gpg >/dev/null
```

6. Now install Tor packages:

```
apt update && apt install tor deb.torproject.org-keyring
```

Exercise 4.2: HTTP

1. Create a directory with a student ID number as a name. Create index.html with visible ID inside the web page body.

2. Run a python-based web server:

```
python3 -m http.server 80 &
```

3. Create the necessary directory for the Hidden Service:

```
mkdir /var/lib/tor/laboratory_onion_service_website
```

4. Set proper permissions and ownership of the tor directory:

```
chown -R debian-tor /var/lib/tor/
chmod 700 -R /var/lib/tor
```

5. Modify the Tor configuration under /etc/tor/torrc. Add these two lines:

```
HiddenServiceDir /var/lib/tor/laboratory_onion_service_website/
HiddenServicePort 80 127.0.0.1:80
```

   The first line specifies the directory where the Hidden Service configuration will be stored. The second line maps a hidden service port to a local port on the localhost interface.

6. In order for the changes to apply, Tor needs to be restarted:

```
systemctl restart tor
```

7. If everything went successfully, then onion address should be visible after running this command:

```
cat /var/lib/tor/laboratory_onion_service_website/hostname
```

8. If something went wrong and the hostname file does not exist, you can find logs with the following command:

```
journalctl -e -u tor@default
```

9. Reach onion address via Tor Browser. Due to NAT/hole punching, it is not necessary to open ports on a firewall.
10. **Question**: What properties do Hidden Services have? What use cases for them can you name?

Exercise 4.3: HTTP with authorisation key

In this part of the exercise, you will add additional authorisation for the web service.

1. Install necessary packages:

```
apt update && apt install -y openssl basez
```

2. Generate private key:

```
openssl genpkey -algorithm x25519 -out
    /tmp/website_key.prv.pem
cat /tmp/website_key.prv.pem | grep -v " PRIVATE KEY" |
    base64pem -d | tail --bytes=32 | base32 | sed 's/=//g' >
    /tmp/website_key.prv.key
```

3. Extract public key from the private key:

```
openssl pkey -in /tmp/website_key.prv.pem -pubout | grep -v "
    PUBLIC KEY" | base64pem -d | tail --bytes=32 | base32 |
    sed 's/=//g' > /tmp/website_key.pub.key
```

4. Create a directory for authorised clients:

```
mkdir /var/lib/tor/laboratory_onion_service_website/authorized_clients/
```

5. Within the newly created directory create <ID>.auth file, where <ID> should be replaced with your student ID number. Edit this file adding this one line:

```
descriptor:x25519:<base32-encoded-public-key>
```

   Where <base32-encoded-public-key> is the public key that was created earlier.

6. Once again, set proper permissions and ownership for newly created files:

```
chown -R debian-tor /var/lib/tor/
chmod 700 -R /var/lib/tor/
```

7. Restart Tor service:

```
systemctl restart tor
```

8. Now try to refresh the Tor Browser. You should be prompted for a key - enter the private key that you created before.
9. There is an alternative way of reaching a Tor Hidden Service. It allows you to omit entering a private key via checkbox. First you need to create an appropriate directory for storing authorisation credentials:

```
mkdir -p /var/lib/tor/onion_auth
```

10. Then you need to edit configuration file at /etc/tor/torrc and add following line:

```
ClientOnionAuthDir /var/lib/tor/onion_auth
```

    Keep in mind that configurations should be separated with an empty line - therefore make an empty line under the hidden service lines you have created before and then paste the ClientOnionAuthDir line.

11. Within the newly created authorisation directory create a file named *<ID>_website.auth_private*, where <ID> is your student ID number.
12. Edit this file and add a following line, replacing marked sections with proper values:

```
<56-char-onion-addr-without-.onion-part>:descriptor:x25519:<x25519
    private key in base32>
```

13. Set proper ownership and permissions:

```
chown -R debian-tor /var/lib/tor/
chmod 700 -R /var/lib/tor/
```

14. Restart Tor in order to apply changes:

```
sudo systemctl restart tor
```

15. Reach the website via torsocks and curl:

```
torsocks curl http://<your_onion_address.onion>
```

16. **Question**: What use cases do you see for Hidden Services protected with authorisation keys?

Exercise 4.4: SSH

The last stage of this exercise should be done in pairs. One student in the pair will host a hidden SSH service, and the second will be a client connecting to it. Steps 1 to 4 should be performed by the student who will host the Hidden Service.

1. Install SSH server:

```
apt update && apt install -y openssh-server
```

2. Start SSH server:

```
systemctl start ssh
```

3. Now edit the configuration file under /etc/tor/torrc, adding following lines:

```
HiddenServiceDir /var/lib/tor/laboratory_onion_service_ssh/
HiddenServicePort 22 127.0.0.1:22
```

   Recall that leaving an empty line between existing configurations.

4. Perform analogous steps to protect a hidden service with authorisation, as you did in the previous stage (steps 2-7).
5. The student who will connect to the SSH service needs to perform steps 9-14 analogously as in the previous stage.
6. The client should connect to the SSH server by running the command below:

```
torsocks ssh <USER>@<ONION_ADDRESS.onion>
```

   Where <USER> can be replaced with either root or another existing user. If you choose a root user, remember to check on the server side if the /etc/ssh/sshd_config file contains the line PermitRootLogin with the value set to *yes*.

7. **Question**: What are the benefits of using SSH over Tor?

Exercise 5: File sharing with ACNs

Learning outcomes: After finishing this exercise, the student should be able to point out the differences between Tor and I2P in terms of file sharing. The student should point out a preferable ACN for this use case.

1. The lab instructor will provide a link to OnionShare with a file of 500 MB in size. Measure the elapsed time to download this file.
2. Knowing the file size and elapsed time, calculate the download speed for the Tor network.
3. The lab instructor will provide a magnet link for I2P snark for the same file. Measure the elapsed time to download this file. Do not remove the file or turn off the I2P once downloaded - make sure that the torrent remains in the *seeding* state.
4. Calculate the download speed for the I2P network.
5. **Question**: Which network was faster? Why? Which network provides better obfuscation when downloading large files and why?

APPENDIX D: EDUCATIONAL DEMONSTRATOR: ACN SIMULATOR

This appendix provides the source code and configuration files for the ACN simulator used in the educational demonstrator described in Chapter 9. The simulator enables students to experiment with simplified single-hop proxy and ACN, making key theoretical concepts from the thesis directly observable through hands-on tasks. The included files allow for easy deployment of the laboratory environment. Each section below describes the purpose of the corresponding file and its role in the overall setup.

Client

This section contains the Python code for the client application used in the simulator. The client enables students to interact with both a single-hop proxy and a multi-hop ACN. By sending messages through different network modes, users can observe how their data is transmitted, how encryption is applied, and how the system responds under various configurations. The client code is designed to be run from the command line and supports easy switching between the simulator modes.

```
import os
import socket
from utils import send, aes_key_from_key_material, encrypt_aes_cbc

SINGLE_HOP_PROXY_PORT = int(os.environ['SINGLE_HOP_PROXY_PORT'])
ENTRY_NODE_PORT = int(os.environ['ENTRY_NODE_PORT'])
SINGLE_HOP_PROXY_KEY_MATERIAL = os.environ['SINGLE_HOP_PROXY_KEY_MATERIAL']
ENTRY_NODE_KEY_MATERIAL = os.environ['ENTRY_NODE_KEY_MATERIAL']
MIDDLE_NODE_KEY_MATERIAL = os.environ['MIDDLE_NODE_KEY_MATERIAL']
EXIT_NODE_KEY_MATERIAL = os.environ['EXIT_NODE_KEY_MATERIAL']

def send_to_single_hop_proxy(socket: socket.socket, message: bytes) -> None:
    single_hop_proxy_key = aes_key_from_key_material(SINGLE_HOP_PROXY_KEY_MATERIAL)
    encrypted_message = encrypt_aes_cbc(message, single_hop_proxy_key)
    send(socket, encrypted_message)
    print("Data successfully sent to the single hop proxy.")

def send_layered(socket: socket.socket, message: str) -> None:
    exit_key = aes_key_from_key_material(EXIT_NODE_KEY_MATERIAL)
    encrypted_message_exit = encrypt_aes_cbc(message, exit_key)
    middle_key = aes_key_from_key_material(MIDDLE_NODE_KEY_MATERIAL)
    data_for_middle_node = {"forward_to": "exit-node", "data": encrypted_message_exit}
    encrypted_message_middle = encrypt_aes_cbc(data_for_middle_node, middle_key)
    entry_key = aes_key_from_key_material(ENTRY_NODE_KEY_MATERIAL)
    data_for_entry_node = {"forward_to": "middle-node", "data":
        encrypted_message_middle}
    encrypted_message_entry = encrypt_aes_cbc(data_for_entry_node, entry_key)
```

```
    send(socket, encrypted_message_entry)
    print("Data successfully sent to the first node of the anonymous communication
        network.")

def create_socket(port: int) -> socket.socket:
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('localhost', port))
    return client_socket

def handle_message(mode_name: str, client_socket: socket.socket, input_message: str) ->
    None:
    request = {
        "website": "http://very-sensitive-website.com",
        "data": input_message  # This data may be encrypted with TLS if using HTTPS or
            unencrypted if using HTTP
    }
    if mode_name == 'anonymous communication network':
        send_layered(client_socket, request)
    else:
        send_to_single_hop_proxy(client_socket, request)

def handle_mode(mode_name: str, port: int) -> None:
    client_socket = create_socket(port)
    while True:
        input_data = input(f"Enter message to send through the {mode_name} or type
            'exit' to quit: ")
        if not input_data:
            print("No message entered. Please enter message to send or type 'exit' to
                quit.")
            continue
        elif input_data.lower() == 'exit':
            break
        handle_message(mode_name, client_socket, input_data)
    client_socket.close()

def run_client() -> None:
    print("Select mode:\n1. Single hop proxy\n2. Anonymous communication network")
    mode = input("Enter 1 or 2: ").strip()
    if mode == '1':
        handle_mode('single hop proxy', SINGLE_HOP_PROXY_PORT)
    elif mode == '2':
        handle_mode('anonymous communication network', ENTRY_NODE_PORT)
    else:
        raise ValueError("Invalid mode. Please enter 1 to use single hop proxy or 2 for
            anonymous communication network.")

def main():
```

```
    run_client()

if __name__ == "__main__":
    main()
```

Router

This section presents the Python code for the router component. The router instance simulates either a single-hop proxy proxy or a single node within the anonymous communication network that operates in onion routing or Mix-net mode. The router is responsible for receiving, decrypting, buffering and mixing (in Mix-net mode), as well as forwarding messages to the next node (if present). The code is structured for containerised deployment, allowing multiple routers to be chained together to form a multi-hop network for the practical exercises.

```
import json
import logging
import os
import random
import socket
import time
from utils import send, aes_key_from_key_material, decrypt_aes_cbc

BUFFER_SIZE = 1024
HOST = "0.0.0.0"
PORT = os.environ['PORT']
KEY_MATERIAL = os.environ['KEY_MATERIAL'] # usually this is negotiated, for example,
    via a Diffie-Hellman exchange
ROUTER_MODE = os.environ['ROUTER_MODE']
MIX_NODE_THRESHOLD = os.environ['MIX_NODE_THRESHOLD']
NEXT_NODE_HOSTNAME = os.environ.get('NEXT_NODE_HOSTNAME', "")
NEXT_NODE_PORT = int(os.environ.get('NEXT_NODE_PORT', "-1"))

def release_mix_buffer(socket: socket.socket, messages_buffer: list) -> None:
    logging.info("Mix node threshold reached, sending buffered messages.")
    for msg in messages_buffer:
        time.sleep(random.random() + 0.1) # Additional delay
        send(socket, msg)
    messages_buffer.clear()

def handle_mix_message(socket: socket.socket, messages_buffer: list, data: str) -> None:
    messages_buffer.append(data)
    logging.info(f"Message buffered. Current count: {len(messages_buffer)}")
    random.shuffle(messages_buffer) # Mixing
    if len(messages_buffer) == int(MIX_NODE_THRESHOLD):
        release_mix_buffer(socket, messages_buffer)
```

```
def handle_message(socket: socket.socket, content: str, messages_buffer: list) -> None:
    data_json = json.loads(content)
    if data_json.get("forward_to") is None:
        return
    data = data_json["data"]
    if ROUTER_MODE == "onion-routing":
        send(socket, data)
    else:
        handle_mix_message(socket, messages_buffer, data)

def create_listening_socket() -> socket.socket:
    listening_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    listening_socket.bind((HOST, int(PORT)))
    listening_socket.listen(5)
    logging.info(f"Listening on port {PORT}...")
    return listening_socket

def create_sending_socket():
    sending_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    if NEXT_NODE_HOSTNAME and NEXT_NODE_PORT != -1:
        sending_socket.connect((NEXT_NODE_HOSTNAME, NEXT_NODE_PORT))
        logging.info(f"Connected to next node at {NEXT_NODE_HOSTNAME}:{NEXT_NODE_PORT}")
    return sending_socket

def handle_incoming_communication(key: bytes, messages_buffer: list, sending_socket:
    socket.socket, client_socket: socket.socket, client_ip: str) -> None:
    while True:
        received_data = client_socket.recv(BUFFER_SIZE)
        if not received_data:
            break
        logging.getLogger().info("\n")
        logging.info(f"Received message from {client_ip} on port: {PORT}.\n" +
                     f"Content before decryption: {received_data}")
        decrypted_message = decrypt_aes_cbc(received_data, key)
        logging.info(f"Conent after decryption: {decrypted_message}")
        handle_message(sending_socket, decrypted_message, messages_buffer)

def disconnect(sending_socket, client_socket):
    client_socket.close()
    sending_socket.close()
    logging.info("Connection closed.")

def run_server() -> None:
    key = aes_key_from_key_material(KEY_MATERIAL)
    messages_buffer = []
    listening_socket = create_listening_socket()
```

```
    sending_socket = create_sending_socket()
    while True:
        client_socket, client_addr = listening_socket.accept()
        try:
            handle_incoming_communication(key, messages_buffer, sending_socket,
                client_socket, client_addr[0])
        except Exception as e:
            logging.error(f"Error: {e}")
        finally:
            disconnect(sending_socket, client_socket)

def main():
    run_server()

if __name__ == "__main__":
    main()
```

Utils

This section provides utility functions shared by both the client and router components. These utilities handle cryptographic operations such as key derivation, AES encryption/decryption, and message formatting, as well as logging and network transmission.

```
import base64
import json
import os
import logging
import socket
from cryptography.hazmat.primitives import hashes, padding
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from dotenv import load_dotenv

load_dotenv()
logging.basicConfig(filename="output.log", level=logging.INFO)

def send(socket: socket.socket, data: str) -> None:
    socket.sendall(data.encode('utf-8'))
    dest_ip, dest_port = socket.getpeername()
    logging.info(f"Sending data to {dest_ip} to port {dest_port}: {data}")

def aes_key_from_key_material(key_material) -> bytes:
    hkdf = HKDF(
        algorithm=hashes.SHA256(),
        length=32,
        salt=None,
```

```
        info=b"AES-key-derivation",
        backend=default_backend(),
    )
    return hkdf.derive(key_material.encode())

def encrypt_aes_cbc(plaintext: bytes, key: bytes) -> str:
    padder = padding.PKCS7(algorithms.AES.block_size).padder()
    padded_data = padder.update(json.dumps(plaintext).encode()) + padder.finalize()
    iv = os.urandom(16)
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    encryptor = cipher.encryptor()
    encrypted_bytes = encryptor.update(padded_data) + encryptor.finalize()
    return base64.b64encode(iv + encrypted_bytes).decode('utf-8')

def decrypt_aes_cbc(ciphertext_with_iv_base64: bytes, key: bytes) -> str:
    ciphertext_with_iv = base64.b64decode(ciphertext_with_iv_base64)
    iv, ciphertext = ciphertext_with_iv[:16], ciphertext_with_iv[16:]
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    decryptor = cipher.decryptor()
    padded_plaintext = decryptor.update(ciphertext) + decryptor.finalize()
    unpadder = padding.PKCS7(algorithms.AES.block_size).unpadder()
    plaintext_bytes = unpadder.update(padded_plaintext) + unpadder.finalize()
    return plaintext_bytes.decode('utf-8')
```

Requirements

This section lists the Python packages required to run the simulator contained within the requirements.txt file.

```
cryptography==45.0.2
python-dotenv==1.1.0
```

Dockerfile

This section provides the Dockerfile used to containerise the router component of the simulator. The Dockerfile provides reproducibility and simplifies the technical setup described in the laboratory instructions.

```
FROM python:3.13-slim

WORKDIR /app

ADD router.py ./router.py
ADD utils.py ./utils.py
ADD requirements.txt ./requirements.txt

RUN pip install -r requirements.txt
```

```
CMD ["python", "router.py"]
```

Docker compose

This section includes the Docker Compose configuration file, which orchestrates the deployment of the entire simulated network environment. It defines the services for each node and manages the environment variables, network connections, and dependencies. With this file, students can launch the entire ACN simulator stack with a single command, making the setup fast and reliable for educational use.

```
version: '3.9'
services:
  single-hop-proxy:
    build:
      context: .
    ports:
      - "${SINGLE_HOP_PROXY_PORT}:${SINGLE_HOP_PROXY_PORT}"
    env_file:
      .env
    environment:
      KEY_MATERIAL: ${SINGLE_HOP_PROXY_KEY_MATERIAL}
      PORT: ${SINGLE_HOP_PROXY_PORT}
  entry-node:
    build:
      context: .
    ports:
      - "${ENTRY_NODE_PORT}:${ENTRY_NODE_PORT}"
    env_file:
      .env
    environment:
      KEY_MATERIAL: ${ENTRY_NODE_KEY_MATERIAL}
      PORT: ${ENTRY_NODE_PORT}
      NEXT_NODE_HOSTNAME: ${MIDDLE_NODE_HOSTNAME}
      NEXT_NODE_PORT: ${MIDDLE_NODE_PORT}
    depends_on:
      middle-node:
        condition: service_started
  middle-node:
    build:
      context: .
    env_file:
      .env
    environment:
      KEY_MATERIAL: ${MIDDLE_NODE_KEY_MATERIAL}
      PORT: ${MIDDLE_NODE_PORT}
      NEXT_NODE_HOSTNAME: ${EXIT_NODE_HOSTNAME}
```

```
      NEXT_NODE_PORT: ${EXIT_NODE_PORT}
    depends_on:
      exit-node:
        condition: service_started
  exit-node:
    build:
      context: .
    env_file:
      .env
    environment:
      KEY_MATERIAL: ${EXIT_NODE_KEY_MATERIAL}
      PORT: ${EXIT_NODE_PORT}
```

Environment variables

This section provides the content of the .env file; in other words, the configuration of the sample environment variables required for the simulator to run. These variables define parameters such as hostnames, port numbers, key materials, router modes, and Mix-net thresholds. The .env file ensures that each container and service is configured properly, supporting setups for different laboratory scenarios.

```
SINGLE_HOP_PROXY_HOSTNAME="single-hop-proxy"
SINGLE_HOP_PROXY_KEY_MATERIAL="single-hop-proxy-key-material"
SINGLE_HOP_PROXY_PORT=1818
ENTRY_NODE_HOSTNAME="entry-node"
ENTRY_NODE_KEY_MATERIAL="entry-node-key-material"
ENTRY_NODE_PORT=2828
MIDDLE_NODE_HOSTNAME="middle-node"
MIDDLE_NODE_KEY_MATERIAL="middle-node-key-material"
MIDDLE_NODE_PORT=3838
EXIT_NODE_HOSTNAME="exit-node"
EXIT_NODE_KEY_MATERIAL="exit-node-key-material"
EXIT_NODE_PORT=4848
ROUTER_MODE="onion-routing" # or "mix-net"
MIX_NODE_THRESHOLD=10
```