



UNIVERSIDADE FEDERAL DE SANTA CATARINA  
CAMPUS FLORIANÓPOLIS  
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Anderson Simioni Assunção

**UMA ARQUITETURA MODULAR PARA REDES OVERLAY PEER-TO-PEER  
DESCENTRALIZADAS COM SUPORTE A COMUNICAÇÃO SEGURA,  
PERSISTÊNCIA DE DADOS E APLICAÇÕES DESCENTRALIZADAS**

Florianópolis  
2026

Anderson Simioni Assunção

**UMA ARQUITETURA MODULAR PARA REDES OVERLAY PEER-TO-PEER  
DESCENTRALIZADAS COM SUPORTE A COMUNICAÇÃO SEGURA,  
PERSISTÊNCIA DE DADOS E APLICAÇÕES DESCENTRALIZADAS**

Trabalho de Conclusão de Curso submetido ao Curso de Graduação em Ciência da Computação do Centro Tecnológico da Universidade Federal de Santa Catarina para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Carlos Becker Westphall,  
Dr.

Florianópolis  
2026



Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.  
Dados inseridos pelo próprio autor.

Simioni Assunção, Anderson  
UMA ARQUITETURA MODULAR PARA REDES OVERLAY PEER-TO-PEER  
DESCENTRALIZADAS COM SUPORTE A COMUNICAÇÃO SEGURA,  
PERSISTÊNCIA DE DADOS E APLICAÇÕES DESCENTRALIZADAS /  
Anderson Simioni Assunção ; orientador, Carlos Becker  
Westphall, 2026.  
109 p.

Trabalho de Conclusão de Curso (graduação) -  
Universidade Federal de Santa Catarina, Centro Tecnológico,  
Graduação em Ciências da Computação, Florianópolis, 2026.

Inclui referências.

1. Ciências da Computação. 2. Redes peer-to-peer. 3.  
Sistemas descentralizados. 4. Redes overlay. 5.  
Criptografia pós-quântica. I. Becker Westphall, Carlos. II.  
Universidade Federal de Santa Catarina. Graduação em  
Ciências da Computação. III. Título.



Anderson Simioni Assunção

**UMA ARQUITETURA MODULAR PARA REDES OVERLAY PEER-TO-PEER  
DESCENTRALIZADAS COM SUPORTE A COMUNICAÇÃO SEGURA,  
PERSISTÊNCIA DE DADOS E APLICAÇÕES DESCENTRALIZADAS**

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo Curso de Graduação em Ciência da Computação.

Florianópolis, 10 de julho de 2026.

---

Profa. Lúcia Helena Martins Pacheco, Dra.  
Coordenadora do Curso

**Banca Examinadora:**

---

Prof. Carlos Becker Westphall, Dr.  
Orientador  
Universidade Federal de Santa Catarina

---

Profa. Carla Merkle Westphall, Dra.  
Avaliadora  
Universidade Federal de Santa Catarina

---

Prof. Wesley dos Reis Bezerra, Dr.  
Avaliador  
Instituto Federal Catarinense

## RESUMO

Este trabalho apresenta o projeto, a implementação e a validação experimental de uma arquitetura modular para redes overlay peer-to-peer descentralizadas. A proposta separa nós físicos, responsáveis pela conectividade e pelo transporte, de nós virtuais, responsáveis pelas identidades e interações lógicas das aplicações. O protótipo AnonNetCore implementa descoberta de pares, tabelas distribuídas, construção e execução de rotas, sessões físicas e virtuais, encaminhamento por relay, transporte TCP e UDP experimental, entrega confiável e transferência de conteúdo por faixas de bytes. A segurança combina identificadores SHA-512, assinaturas ML-DSA, estabelecimento de chaves ML-KEM e criptografia autenticada AES-256-GCM-SIV. A validação utiliza clusters Docker, testes integrados, uma API HTTP/WebSocket, ferramentas de observabilidade e uma prova de conceito de rede social. Os resultados demonstram a integração funcional entre descoberta, roteamento, persistência distribuída, comunicação virtual e aplicações externas. O trabalho permanece um MVP experimental: travessia completa de NAT, endurecimento contra ataques, escalabilidade em redes públicas e auditoria formal de segurança são trabalhos futuros.

**Palavras-chave:** redes peer-to-peer; redes overlay; sistemas descentralizados; privacidade; tabelas hash distribuídas; criptografia pós-quântica.

## ABSTRACT

This work presents the design, implementation, and experimental validation of a modular architecture for decentralized peer-to-peer overlay networks. The proposal separates physical nodes, responsible for connectivity and transport, from virtual nodes, responsible for application identities and logical interactions. The AnonNetCore prototype implements peer discovery, distributed tables, route construction and execution, physical and virtual sessions, relay forwarding, TCP and experimental UDP transport, reliable delivery, and byte-range content transfer. Security combines SHA-512 identifiers, ML-DSA signatures, ML-KEM key establishment, and AES-256-GCM-SIV authenticated encryption. Validation uses Docker clusters, integrated smoke tests, an HTTP/WebSocket API, observability tools, and a social-network proof of concept. Results demonstrate functional integration among discovery, routing, distributed persistence, virtual communication, and external applications. The work remains an experimental MVP: complete NAT traversal, attack resistance, public-network scalability, and formal security auditing remain future work.

**Keywords:** peer-to-peer networks; overlay networks; decentralized systems; privacy; distributed hash tables; post-quantum cryptography.

## LISTA DE ILUSTRAÇÕES

|                                                                                                                                                           |    |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| Figure 1 – PRISMA flow diagram. . . . .                                                                                                                   | 23 |
| Figure 2 – Layered architecture. . . . .                                                                                                                  | 35 |
| Figure 3 – Connection Management diagram. . . . .                                                                                                         | 39 |
| Figure 4 – Distributed Routing Table. . . . .                                                                                                             | 41 |
| Figure 5 – Generic route construction through a route strategy. . . . .                                                                                   | 42 |
| Figure 6 – Example of hop-by-hop route construction with a TTL-based strategy. . . . .                                                                    | 42 |
| Figure 7 – Peer discovery process. . . . .                                                                                                                | 45 |
| Figure 8 – Security mechanisms and the stages at which they are applied. . . . .                                                                          | 46 |
| Figure 9 – Distributed Data Table. . . . .                                                                                                                | 47 |
| Figure 10 – Ephemeral Virtual Nodes. . . . .                                                                                                              | 48 |
| Figure 11 – Content discovery and resolution model. . . . .                                                                                               | 56 |
| Figure 12 – Message routing process. . . . .                                                                                                              | 60 |
| Figure 13 – From distributed discovery to virtual application communication. . . . .                                                                      | 63 |
| Figure 14 – Implemented layered architecture of the AnonNetCore Python MVP. . . . .                                                                       | 64 |
| Figure 15 – Main implemented flows: bootstrap, route creation, virtual session establishment, and DHT publish/query. . . . .                              | 70 |
| Figure 16 – Separation between the route protocol lifecycle and the route strategy used to select physical hops. . . . .                                  | 71 |
| Figure 17 – Cryptographic identity, independent physical and virtual key establishment, and two-layer message protection in the implemented Core. . . . . | 74 |
| Figure 18 – Integration flow between the social PoC, the local Core, the physical overlay route, and the destination application. . . . .                 | 81 |
| Figure 19 – Cluster execution console showing multiple physical nodes running in the experimental Docker environment. . . . .                             | 85 |
| Figure 20 – Social proof of concept showing profile state, friends, and feed synchronization through the local API. . . . .                               | 86 |
| Figure 21 – Debug Console overview used to observe the experimental execution. . . . .                                                                    | 87 |
| Figure 22 – Final smoke-test summary showing the successful integrated Core and social PoC validation flows. . . . .                                      | 87 |

## LISTA DE TABELAS

|                                                                                               |    |
|-----------------------------------------------------------------------------------------------|----|
| Table 1 – Comparative Analysis of Selected Works . . . . .                                    | 26 |
| Table 2 – Functional requirements of the proposed architecture. . . . .                       | 29 |
| Table 3 – Non-functional requirements of the proposed architecture. . . . .                   | 30 |
| Table 4 – Core entities of the conceptual model. . . . .                                      | 31 |
| Table 5 – Examples of route strategies supported by the architectural model. .                | 44 |
| Table 6 – Layered abuse and Sybil mitigations. . . . .                                        | 52 |
| Table 7 – Storage replication policies. . . . .                                               | 55 |
| Table 8 – Resilience mechanisms by network state. . . . .                                     | 57 |
| Table 9 – Main technologies used in the experimental MVP. . . . .                             | 63 |
| Table 10 – Implemented DHT namespaces. . . . .                                                | 69 |
| Table 11 – Implemented protocol families and message types. . . . .                           | 76 |
| Table 12 – Summary of local persistence entities. . . . .                                     | 79 |
| Table 13 – Summary of implemented local API groups. . . . .                                   | 80 |
| Table 14 – Implemented smoke tests and validation goals. . . . .                              | 83 |
| Table 15 – Summary of experimental validation results. . . . .                                | 84 |
| Table 16 – Comparison between the implemented MVP and related systems or<br>concepts. . . . . | 88 |
| Table 17 – Current limitations and future work. . . . .                                       | 90 |

## LISTA DE ABREVIATURAS E SIGLAS

|      |                                        |
|------|----------------------------------------|
| AAD  | Associated Authenticated Data          |
| AES  | Advanced Encryption Standard           |
| API  | Application Programming Interface      |
| DDT  | Distributed Data Table                 |
| DHT  | Distributed Hash Table                 |
| DPNT | Distributed Physical Nodes Table       |
| DPT  | Distributed Pointer Table              |
| DRT  | Distributed Route Table                |
| DTT  | Distributed Tag Table                  |
| KEM  | Key Encapsulation Mechanism            |
| MVP  | Minimum Viable Product                 |
| NAT  | Network Address Translation            |
| P2P  | Peer-to-Peer                           |
| PN   | Physical Node                          |
| PoC  | Proof of Concept                       |
| PoW  | Proof of Work                          |
| PQC  | Post-Quantum Cryptography              |
| RTT  | Round-Trip Time                        |
| SHA  | Secure Hash Algorithm                  |
| TCP  | Transmission Control Protocol          |
| TTL  | Time to Live                           |
| UDP  | User Datagram Protocol                 |
| UFSC | Universidade Federal de Santa Catarina |
| VN   | Virtual Node                           |

## SUMÁRIO

|            |                                                           |           |
|------------|-----------------------------------------------------------|-----------|
| <b>1</b>   | <b>INTRODUCTION . . . . .</b>                             | <b>12</b> |
| <b>2</b>   | <b>THEORETICAL BACKGROUND . . . . .</b>                   | <b>14</b> |
| 2.1        | Peer-to-Peer Networks . . . . .                           | 14        |
| 2.2        | NAT Traversal . . . . .                                   | 15        |
| 2.3        | Post-Quantum Cryptography . . . . .                       | 16        |
| 2.4        | Cryptographic Hash Functions . . . . .                    | 17        |
| 2.5        | Distributed Hash Tables (DHTs) . . . . .                  | 18        |
| 2.6        | Kademlia and XOR-Based Routing . . . . .                  | 20        |
| <b>3</b>   | <b>SYSTEMATIC LITERATURE REVIEW METHODOLOGY . . . . .</b> | <b>21</b> |
| 3.1        | Research Question . . . . .                               | 21        |
| 3.2        | Databases . . . . .                                       | 21        |
| 3.3        | Search String . . . . .                                   | 21        |
| 3.4        | Types of Publications . . . . .                           | 22        |
| 3.5        | Inclusion Criteria . . . . .                              | 22        |
| 3.6        | Exclusion Criteria . . . . .                              | 22        |
| <b>4</b>   | <b>SELECTION PROCESS (PRISMA) . . . . .</b>               | <b>23</b> |
| <b>5</b>   | <b>DATA EXTRACTION AND ANALYSIS . . . . .</b>             | <b>25</b> |
| 5.1        | Data Extraction Process . . . . .                         | 25        |
| 5.2        | Comparative Analysis . . . . .                            | 25        |
| 5.3        | Discussion of Findings . . . . .                          | 26        |
| <b>6</b>   | <b>PROPOSED ARCHITECTURE . . . . .</b>                    | <b>28</b> |
| 6.1        | Design Rationale . . . . .                                | 28        |
| 6.2        | System Requirements . . . . .                             | 29        |
| 6.2.1      | Functional Requirements . . . . .                         | 29        |
| 6.2.2      | Non-Functional Requirements . . . . .                     | 29        |
| <b>6.3</b> | <b>Conceptual Model . . . . .</b>                         | <b>30</b> |
| 6.3.1      | Network Entities . . . . .                                | 30        |
| 6.3.2      | Resource Representation . . . . .                         | 31        |
| 6.3.3      | Identity Model . . . . .                                  | 32        |
| 6.3.4      | Overlay Abstraction . . . . .                             | 32        |
| 6.3.5      | Kademlia-Inspired DHT Responsibility Model . . . . .      | 32        |
| 6.3.6      | Core Distributed Structures . . . . .                     | 33        |

|            |                                                   |           |
|------------|---------------------------------------------------|-----------|
| 6.3.6.1    | Distributed Pointer Table (DPT) . . . . .         | 33        |
| 6.3.6.2    | Distributed Tag Table (DTT) . . . . .             | 33        |
| 6.3.6.3    | Distributed Data Table (DDT) . . . . .            | 34        |
| 6.3.6.4    | Distributed Route Table (DRT) . . . . .           | 34        |
| 6.3.6.5    | Distributed Physical Nodes Table (DPNT) . . . . . | 34        |
| <b>6.4</b> | <b>Layered Architecture . . . . .</b>             | <b>35</b> |
| 6.4.1      | Physical and Classical Network Layer . . . . .    | 35        |
| 6.4.2      | Overlay Network Layer . . . . .                   | 36        |
| 6.4.3      | Service Layer . . . . .                           | 36        |
| 6.4.4      | Application Layer . . . . .                       | 37        |
| <b>6.5</b> | <b>System Components . . . . .</b>                | <b>37</b> |
| 6.5.1      | Peer Management . . . . .                         | 37        |
| 6.5.2      | Connection Management . . . . .                   | 38        |
| 6.5.3      | Routing Mechanism . . . . .                       | 40        |
| 6.5.4      | Route Strategy Abstraction . . . . .              | 43        |
| 6.5.5      | Discovery Module . . . . .                        | 45        |
| 6.5.6      | Cryptographic Module . . . . .                    | 46        |
| 6.5.7      | Storage Management . . . . .                      | 47        |
| <b>6.6</b> | <b>Communication Protocols . . . . .</b>          | <b>48</b> |
| 6.6.1      | Peer Discovery Protocol . . . . .                 | 49        |
| 6.6.2      | Connection Establishment . . . . .                | 49        |
| 6.6.3      | Routing and Lookup . . . . .                      | 49        |
| 6.6.4      | Data Exchange Protocol . . . . .                  | 50        |
| <b>6.7</b> | <b>Security Model . . . . .</b>                   | <b>50</b> |
| 6.7.1      | Identity and Authentication . . . . .             | 50        |
| 6.7.2      | Encryption Mechanisms . . . . .                   | 51        |
| 6.7.3      | Post-Quantum Cryptography . . . . .               | 51        |
| 6.7.4      | Key Management . . . . .                          | 51        |
| 6.7.5      | Abuse Control and Sybil Resistance . . . . .      | 52        |
| 6.7.6      | Publication Proof-of-Work . . . . .               | 53        |
| <b>6.8</b> | <b>Distributed Storage . . . . .</b>              | <b>54</b> |
| 6.8.1      | Data Distribution Strategy . . . . .              | 54        |
| 6.8.2      | Replication Mechanisms . . . . .                  | 55        |
| 6.8.3      | Data Consistency . . . . .                        | 55        |
| 6.8.4      | Data Lifecycle and Expiration . . . . .           | 55        |
| 6.8.5      | Content Discovery and Resolution Model . . . . .  | 55        |
| <b>6.9</b> | <b>Resilience and Fault Tolerance . . . . .</b>   | <b>56</b> |
| 6.9.1      | Node Churn Handling . . . . .                     | 57        |
| 6.9.2      | Redundancy Strategies . . . . .                   | 57        |

|             |                                                               |           |
|-------------|---------------------------------------------------------------|-----------|
| 6.9.3       | Failure Recovery . . . . .                                    | 57        |
| 6.9.4       | Network Rebalancing . . . . .                                 | 58        |
| <b>6.10</b> | <b>Operational Workflows . . . . .</b>                        | <b>58</b> |
| 6.10.1      | Node Join Procedure . . . . .                                 | 58        |
| 6.10.2      | Node Departure Procedure . . . . .                            | 58        |
| 6.10.3      | Data Lookup Process . . . . .                                 | 59        |
| 6.10.4      | Message Routing Process . . . . .                             | 59        |
| <b>7</b>    | <b>IMPLEMENTATION . . . . .</b>                               | <b>61</b> |
| <b>7.1</b>  | <b>Implementation Methodology . . . . .</b>                   | <b>61</b> |
| <b>7.2</b>  | <b>Implementation Scope . . . . .</b>                         | <b>62</b> |
| <b>7.3</b>  | <b>Technology Stack . . . . .</b>                             | <b>62</b> |
| <b>7.4</b>  | <b>Implemented Architecture . . . . .</b>                     | <b>63</b> |
| 7.4.1       | Physical Layer . . . . .                                      | 64        |
| 7.4.2       | Relay-Assisted Physical Transport . . . . .                   | 65        |
| 7.4.3       | Virtual Layer . . . . .                                       | 66        |
| 7.4.4       | DHT and Distributed Tables . . . . .                          | 67        |
| <b>7.5</b>  | <b>Bootstrap and Peer Discovery . . . . .</b>                 | <b>69</b> |
| <b>7.6</b>  | <b>Route Construction and Route Execution . . . . .</b>       | <b>70</b> |
| 7.6.1       | Route Strategy Abstraction . . . . .                          | 70        |
| 7.6.2       | Hop-by-Hop Route Construction and Execution . . . . .         | 72        |
| <b>7.7</b>  | <b>Virtual Sessions and Virtual Messages . . . . .</b>        | <b>73</b> |
| <b>7.8</b>  | <b>Reliable Session Payloads . . . . .</b>                    | <b>75</b> |
| <b>7.9</b>  | <b>Protocol Message Types . . . . .</b>                       | <b>76</b> |
| <b>7.10</b> | <b>Local Persistence Model . . . . .</b>                      | <b>78</b> |
| <b>7.11</b> | <b>Local API . . . . .</b>                                    | <b>79</b> |
| <b>7.12</b> | <b>Social Proof of Concept . . . . .</b>                      | <b>81</b> |
| <b>7.13</b> | <b>Experimental Execution . . . . .</b>                       | <b>82</b> |
| <b>7.14</b> | <b>Validation and Test Results . . . . .</b>                  | <b>83</b> |
| <b>7.15</b> | <b>Comparison with Related Systems and Concepts . . . . .</b> | <b>88</b> |
| <b>7.16</b> | <b>Known Limitations and Future Work . . . . .</b>            | <b>89</b> |
| <b>7.17</b> | <b>Chapter Summary . . . . .</b>                              | <b>91</b> |
| <b>8</b>    | <b>CONCLUSION . . . . .</b>                                   | <b>92</b> |
|             | <b>Declaration of Artificial Intelligence Use . . . . .</b>   | <b>95</b> |
|             | <b>REFERENCES . . . . .</b>                                   | <b>96</b> |
| <b>A</b>    | <b>SCIENTIFIC ARTICLE DERIVED FROM THIS WORK . . . . .</b>    | <b>98</b> |

## 1 INTRODUCTION

The increasing centralization of modern internet services has raised significant concerns regarding privacy, surveillance, and control over digital communication infrastructures. Most contemporary online platforms rely on centralized architectures, where user data, communication flows, and identity management are controlled by a limited number of service providers. This model creates single points of failure and exposes users to risks such as large-scale data breaches, censorship, and pervasive monitoring.

At the same time, the rapid progress in quantum computing introduces new challenges for digital security. Several widely used cryptographic algorithms, particularly those based on integer factorization or discrete logarithms, may become vulnerable once sufficiently powerful quantum computers become available. As a result, the design of communication systems that incorporate post-quantum cryptographic primitives has become an important research topic in modern network security.

Decentralized communication architectures offer a promising alternative to traditional centralized systems. Peer-to-peer (P2P) networks distribute responsibility across participating nodes, eliminating the need for permanent central authorities and improving system resilience. However, designing large-scale decentralized systems introduces several technical challenges, including efficient peer discovery, routing in dynamic environments, traversal of Network Address Translators (NATs), and the preservation of user anonymity.

Several existing systems, such as Tor, I2P, and IPFS, attempt to address some of these challenges by providing privacy-preserving communication or decentralized storage. Nevertheless, these systems often focus on specific aspects of the problem and may present limitations in terms of scalability, routing efficiency, or resistance to traffic analysis.

This work proposes the design and implementation of a decentralized network layer focused on anonymity-oriented communication and distributed data persistence. The proposed architecture is based on a peer-to-peer overlay network where nodes communicate without relying on centralized infrastructure. The system incorporates multiple distributed discovery structures, probabilistic routing mechanisms, and cryptographic identity models in order to preserve user privacy while enabling efficient communication.

In the proposed model, nodes are identified through cryptographic keys rather than network addresses, and routing information is stored using distributed hash table-based structures that allow scalable discovery of peers, routes, and published data. The full architecture is designed to support NAT traversal mechanisms such as UDP

hole punching and relay-assisted communication, while the experimental MVP validates the main overlay behavior over TCP, experimental UDP transport, and relay-assisted physical forwarding. Cryptographic protocols, including post-quantum primitives, are used to protect communication confidentiality, authentication, and integrity.

To evaluate the feasibility of the proposed architecture, this work also presents a prototype application in the form of a decentralized social network, implemented as a proof of concept over the proposed network layer. The prototype allows experimental evaluation of communication flows, routing behavior, and security properties of the system.

The main contributions of this work include the design of a modular decentralized networking architecture, the integration of privacy-preserving routing and distributed discovery mechanisms, and the practical validation of the approach through an experimental implementation.

## 2 THEORETICAL BACKGROUND

### 2.1 PEER-TO-PEER NETWORKS

Peer-to-peer (P2P) networks are distributed systems in which nodes, referred to as peers, communicate directly with each other without relying on centralized coordination (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022). Unlike traditional client-server architectures, P2P systems decentralize control, enabling each node to act both as a client and as a server (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022). This paradigm enhances system robustness, scalability, and fault tolerance (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).

A fundamental characteristic of P2P networks is their ability to self-organize. Nodes dynamically join and leave the network, a phenomenon known as *churn*, requiring the system to continuously adapt its topology and routing structures (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022). Efficient handling of churn is essential to maintain connectivity and ensure reliable data dissemination in large-scale environments (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).

P2P networks can be broadly classified into structured and unstructured overlays (Mehr-Un-Nisa et al., 2019), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022). Unstructured P2P networks, such as early file-sharing systems, rely on flooding or random walks for resource discovery, which may lead to high communication overhead and limited scalability (Mehr-Un-Nisa et al., 2019). In contrast, structured P2P networks employ well-defined topologies and deterministic routing mechanisms, typically based on Distributed Hash Tables (DHTs) (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).

Distributed Hash Tables provide a scalable and efficient method for resource lookup by mapping keys to nodes in a deterministic manner (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022). Classical DHT-based systems, such as Chord (STOICA et al., 2001), organize nodes in a logical ring structure and guarantee lookup operations in logarithmic time relative to the number of nodes. Similarly, Kademlia (MAYMOUNKOV; MAZIERES, 2002) introduces a distance metric based on the XOR operation, enabling efficient routing and parallel query mechanisms. Other relevant systems include Pastry (ROWSTRON; DRUSCHEL, 2001) and Tapestry (ZHAO; KUBIATOWICZ; JOSEPH, 2001), which explore alternative routing strategies and proximity-aware optimizations.

These structured overlays significantly improve scalability and lookup efficiency; however, they also introduce challenges related to network dynamics, heterogeneity,

and real-world deployment constraints (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022). In particular, issues such as node churn, network latency, and the presence of nodes behind Network Address Translation (NAT) devices complicate direct peer-to-peer communication (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).

Despite these limitations, P2P networks remain a fundamental building block for decentralized systems, enabling applications such as distributed storage, content distribution, and resilient communication infrastructures (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022). Their inherent decentralization and scalability make them particularly suitable for scenarios requiring fault tolerance and autonomy, motivating their continued evolution and integration with modern networking and cryptographic techniques (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).

## 2.2 NAT TRAVERSAL

Network Address Translation (NAT) is widely used in modern networks to allow multiple devices to share a single public IP address. While NAT improves address utilization and provides a basic level of security by isolating internal networks, it introduces significant challenges for peer-to-peer (P2P) communication (LI; WANG; WANG, 2022). In particular, nodes behind NATs are typically not directly reachable from external networks, which hinders the establishment of direct end-to-end connections (LI; WANG; WANG, 2022).

Several techniques have been proposed to overcome NAT-related limitations. One of the most widely adopted approaches is the use of Session Traversal Utilities for NAT (STUN), which allows a node to discover its public-facing IP address and port as seen by external servers (PETIT-HUGUENIN et al., 2020). However, STUN alone is insufficient in more restrictive NAT scenarios, since address discovery does not guarantee that an inbound path can be created for all NAT behaviors (PETIT-HUGUENIN et al., 2020).

Traversal Using Relays around NAT (TURN) addresses this limitation by relaying traffic through an intermediate server when direct communication cannot be established (REDDY et al., 2020). Although effective, TURN introduces additional latency and creates a degree of centralization, which may be undesirable in fully decentralized systems (REDDY et al., 2020).

Interactive Connectivity Establishment (ICE) combines multiple techniques, including STUN and TURN, to systematically discover the best possible communication path between peers (KERANEN; HOLMBERG; ROSENBERG, 2018). ICE attempts direct connectivity first and falls back to relay-based communication when necessary,

improving reliability in heterogeneous network environments (KERANEN; HOLMBERG; ROSENBERG, 2018).

Another widely used technique in P2P systems is UDP hole punching, which enables two peers behind NATs to establish direct communication by coordinating simultaneous outbound packets (LI; WANG; WANG, 2022), (PETIT-HUGUENIN et al., 2020), (KERANEN; HOLMBERG; ROSENBERG, 2018). This approach is particularly effective in NAT types that preserve endpoint mappings and is commonly used in decentralized applications due to its low latency and reduced reliance on centralized infrastructure (LI; WANG; WANG, 2022), (PETIT-HUGUENIN et al., 2020).

Recent advancements have also explored the use of modern transport protocols such as QUIC, which operates over UDP and integrates features such as connection migration and improved congestion control (IYENGAR; THOMSON, 2021). These properties make QUIC a promising candidate for NAT traversal in dynamic network conditions, especially when combined with P2P overlay mechanisms (IYENGAR; THOMSON, 2021).

Despite these techniques, NAT traversal remains a non-trivial challenge, particularly in large-scale decentralized networks with heterogeneous NAT behaviors (PETIT-HUGUENIN et al., 2020), (REDDY et al., 2020), (KERANEN; HOLMBERG; ROSENBERG, 2018). In such scenarios, hybrid approaches combining direct communication, relay mechanisms, and overlay routing are often required to ensure connectivity among all peers (PETIT-HUGUENIN et al., 2020), (REDDY et al., 2020), (KERANEN; HOLMBERG; ROSENBERG, 2018). These challenges directly motivate the development of more robust communication schemes in decentralized P2P systems (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).

## 2.3 POST-QUANTUM CRYPTOGRAPHY

The advancement of quantum computing poses a significant threat to widely used classical cryptographic systems. Algorithms such as RSA and Elliptic Curve Cryptography (ECC), which rely on the computational hardness of problems like integer factorization and discrete logarithms, can be efficiently broken using quantum algorithms such as Shor's algorithm (SHOR, 1997). This scenario has motivated the development of post-quantum cryptography (PQC), which aims to design cryptographic schemes secure against both classical and quantum adversaries (SHOR, 1997).

Among the various approaches to PQC, lattice-based cryptography has emerged as one of the most promising candidates due to its strong security guarantees and practical efficiency (REGEV, 2009), (BOS et al., 2017). These schemes are based on hard mathematical problems such as the Learning With Errors (LWE) and Module-

LWE problems, which remain computationally intractable even for quantum computers (REGEV, 2009), (BOS et al., 2017).

A prominent example of a lattice-based cryptographic scheme is CRYSTALS-Kyber, which has been selected by the National Institute of Standards and Technology (NIST) as a standard for post-quantum key encapsulation mechanisms (KEMs) (BOS et al., 2017), (National Institute of Standards and Technology, 2024a). Kyber enables two parties to securely establish a shared symmetric key over an insecure channel, without requiring prior secret exchange (BOS et al., 2017), (National Institute of Standards and Technology, 2024a).

The operation of Kyber is divided into three main phases: key generation, encapsulation, and decapsulation (BOS et al., 2017), (National Institute of Standards and Technology, 2024a). In the key generation phase, a public and private key pair is created based on structured lattice operations. During encapsulation, a sender uses the recipient's public key to generate a ciphertext and derive a shared secret. In the decapsulation phase, the recipient uses their private key to recover the shared secret from the ciphertext (BOS et al., 2017), (National Institute of Standards and Technology, 2024a).

In practical systems, Kyber is often combined with symmetric cryptographic algorithms such as AES-256 to provide efficient and secure data encryption (BOS et al., 2017). In this hybrid approach, Kyber is used to securely exchange a symmetric key, which is then used to encrypt the actual data payload. In modern protocol designs, this symmetric protection should preferably use authenticated encryption modes, so that confidentiality and integrity are provided together rather than relying on encryption alone (BOS et al., 2017), (GUERON; LANGLEY; LINDELL, 2019). This design leverages the efficiency of symmetric cryptography while ensuring quantum-resistant key exchange (BOS et al., 2017).

The adoption of post-quantum cryptography is particularly relevant in decentralized and privacy-oriented systems, where long-term data confidentiality and resistance to future attacks are critical (BOS et al., 2017), (National Institute of Standards and Technology, 2024a). As a result, integrating PQC mechanisms into P2P architectures represents an important step toward building secure communication infrastructures resilient to emerging computational paradigms (BOS et al., 2017), (National Institute of Standards and Technology, 2024a).

## 2.4 CRYPTOGRAPHIC HASH FUNCTIONS

Cryptographic hash functions are fundamental components in modern distributed and secure systems. A hash function maps input data of arbitrary size to a fixed-size

output, known as a hash or digest, in a deterministic and efficient manner.

For a hash function to be considered cryptographically secure, it must satisfy several key properties. First, it must be computationally infeasible to find two distinct inputs that produce the same output (collision resistance). Second, given a hash output, it should be infeasible to reconstruct the original input (preimage resistance). Third, small changes in the input should produce significantly different outputs, a property known as the avalanche effect.

In decentralized and peer-to-peer (P2P) systems, hash functions play a critical role beyond data integrity (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022). They are commonly used for content addressing, message identification, and efficient data verification (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022). In particular, hashing enables nodes to detect duplicate messages, avoid redundant transmissions, and mitigate network flooding (Mehr-Un-Nisa et al., 2019), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).

This property is especially relevant in gossip-based or broadcast-based communication models, where messages may propagate through multiple paths in the network (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022). By maintaining a record of previously seen message hashes, nodes can prevent infinite propagation loops and significantly reduce unnecessary bandwidth consumption (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).

Additionally, cryptographic hash functions are often used in Distributed Hash Tables (DHTs), where keys are mapped to nodes in a deterministic manner (STOICA et al., 2001), (MAYMOUNKOV; MAZIERES, 2002), (ROWSTRON; DRUSCHEL, 2001), (ZHAO; KUBIATOWICZ; JOSEPH, 2001), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022). This enables efficient lookup and routing in large-scale decentralized systems (STOICA et al., 2001), (MAYMOUNKOV; MAZIERES, 2002), (ROWSTRON; DRUSCHEL, 2001), (ZHAO; KUBIATOWICZ; JOSEPH, 2001), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).

In this work, SHA-512 is used in the prototype to derive compact identifiers for public keys, DHT records, and content references. The use of strong cryptographic hashing ensures both integrity and efficiency, contributing to the scalability and robustness of the proposed architecture.

## 2.5 DISTRIBUTED HASH TABLES (DHTS)

In the context of this work, Distributed Hash Tables (DHTs) are relevant not as a universal solution for every network function, but as an important theoretical foundation for scalable distributed resolution mechanisms (STOICA et al., 2001), (MAYMOUNKOV;

MAZIERES, 2002), (ROWSTRON; DRUSCHEL, 2001), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022). Classical DHT models demonstrate how structured overlay organization can support efficient lookup operations without requiring centralized coordination. Chord, for example, organizes the identifier space as a logical ring and maps keys to successor nodes (STOICA et al., 2001). Kademlia, on the other hand, introduces an XOR-based distance metric to organize node identifiers and guide lookup operations (MAYMOUNKOV; MAZIERES, 2002).

From a theoretical perspective, the importance of DHTs lies in their ability to provide structured and predictable resolution over large and dynamic peer sets (STOICA et al., 2001), (MAYMOUNKOV; MAZIERES, 2002), (ROWSTRON; DRUSCHEL, 2001). Rather than depending exclusively on flooding or unstructured dissemination, DHT-based approaches make it possible to associate identifiers with values through deterministic or semi-deterministic placement and retrieval rules. As a result, these models are often used as the foundation for scalable lookup services in decentralized environments (STOICA et al., 2001), (MAYMOUNKOV; MAZIERES, 2002), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).

In architectures that distinguish between physical communication endpoints and overlay-level logical identities, DHT principles may be applied in different ways depending on the purpose of each distributed structure. On the one hand, they may support logical routing by mapping virtual identifiers to information that helps initiate communication toward a destination. On the other hand, similar principles may also be adapted to maintain recoverable information about physical participants, content descriptors, semantic tags, and stable pointers.

For this reason, DHTs are treated in this work as a theoretical and architectural basis for the design of specialized distributed tables, rather than as a single monolithic mechanism responsible for all discovery and communication functions. This view is consistent with the broader literature, in which DHT-inspired designs are frequently adapted to the requirements of specific overlay systems while preserving the core benefits of structured lookup, scalability, and decentralization (STOICA et al., 2001), (MAYMOUNKOV; MAZIERES, 2002), (ROWSTRON; DRUSCHEL, 2001), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).

The proposed architecture adopts a Kademlia-inspired responsibility model because its XOR-distance principle is naturally compatible with cryptographic identifiers. In this model, both node identifiers and record keys are represented as hash-based values, and responsibility for a record is assigned to the  $K$  active peers whose identifiers are closest to the record key under the XOR distance metric. This model is not intended to reproduce the complete Kademlia protocol, but to adopt its core principle of proximity-based responsibility in a way that fits the proposed architecture.

This approach allows the same placement logic to be reused across different distributed structures, such as the Distributed Routing Table (DRT), the Distributed Tag Table (DTT), the Distributed Data Table (DDT), and the Distributed Pointer Table (DPT). Therefore, the DHT layer acts as a common low-level substrate for distributed routing, content discovery, content availability indexing, and stable pointer resolution.

## 2.6 KADEMLIA AND XOR-BASED ROUTING

Kademlia is a structured peer-to-peer DHT originally proposed by Maymounkov and Mazières. Its main contribution is the use of an XOR-based distance metric to organize the overlay identifier space and guide lookup operations (MAYMOUNKOV; MAZIERES, 2002). In this model, each node has a unique identifier, and the distance between two identifiers is computed by applying the bitwise XOR operation.

Given a node identifier  $node\_id$  and a DHT key  $dht\_key$ , the distance between them can be represented as:

$$distance(node\_id, dht\_key) = node\_id \oplus dht\_key$$

This distance metric is especially suitable for systems where both node identifiers and resource identifiers are derived from cryptographic hashes. Since hash outputs are uniformly distributed, the XOR metric provides a deterministic way to compare proximity between nodes and keys without requiring a centralized assignment authority.

In Kademlia-like systems, responsibility for a key is generally assigned to the nodes whose identifiers are closest to that key according to the XOR distance metric. This property makes the model useful for decentralized lookup, replication, and metadata distribution. Instead of assigning a record to a single node, the system may replicate the record among the  $K$  closest active nodes, improving availability and resilience against churn.

The architecture proposed in this work does not attempt to reproduce the complete Kademlia protocol. Instead, it adopts a Kademlia-inspired responsibility model based on XOR distance and proximity-based replication. This distinction is important because implementation details such as bucket maintenance, lookup parallelism, refresh policies, and churn handling may be adapted according to the specific needs of the system.

The resulting model is compatible with the hash-based identifiers used throughout this architecture, including physical node identifiers, virtual node identifiers, content hashes, tag hashes, and pointer records. As a result, the same placement logic can be reused across the Distributed Routing Table, Distributed Tag Table, Distributed Data Table, and Distributed Pointer Table.

### 3 SYSTEMATIC LITERATURE REVIEW METHODOLOGY

This work adopts a Systematic Literature Review (SLR) methodology in order to identify, analyze, and synthesize relevant studies related to decentralized peer-to-peer network architectures, anonymity mechanisms, and secure communication techniques.

#### 3.1 RESEARCH QUESTION

The review is guided by the following research question:

What decentralized peer-to-peer network architectures provide mechanisms for anonymous and secure communication?

#### 3.2 DATABASES

The literature search was conducted using the following scientific databases:

- Google Scholar
- IEEE Xplore
- ACM Digital Library
- arXiv

These platforms were selected due to their broad coverage of peer-reviewed publications in computer science and network research.

#### 3.3 SEARCH STRING

The following search query was used to retrieve relevant publications:

```
("peer-to-peer" OR "P2P")
AND ("overlay network" OR "overlay networks")
AND ("decentralized" OR "decentralized communication"
      OR "decentralized network")
AND ("anonymous" OR "anonymity" OR "privacy preserving"
      OR "secure communication")
AND ("routing" OR "peer discovery"
      OR "network architecture")
```

The query was applied to the selected databases to identify studies related to decentralized communication architectures and privacy-preserving network systems.

### 3.4 TYPES OF PUBLICATIONS

The review considered the following types of scientific publications:

- Journal articles
- Conference papers

### 3.5 INCLUSION CRITERIA

Publications were included in the review if they satisfied the following criteria:

- Published between 2015 and 2025
- Address peer-to-peer or decentralized network architectures
- Discuss privacy, anonymity, or secure communication mechanisms

### 3.6 EXCLUSION CRITERIA

Publications were excluded if they met any of the following conditions:

- Focus exclusively on blockchain-based systems
- Address Internet of Things (IoT) networks without relevance to P2P communication
- Are books, theses, or non-scientific sources

Classical foundational works published before 2015 were not treated as primary studies in the systematic review. They were used as background references when necessary to explain established concepts such as DHTs, structured overlays, anonymous routing, and NAT traversal. This distinction preserves the temporal scope of the SLR while still allowing the theoretical background to cite foundational protocols and systems.

#### 4 SELECTION PROCESS (PRISMA)

## PRISMA Flow Diagram

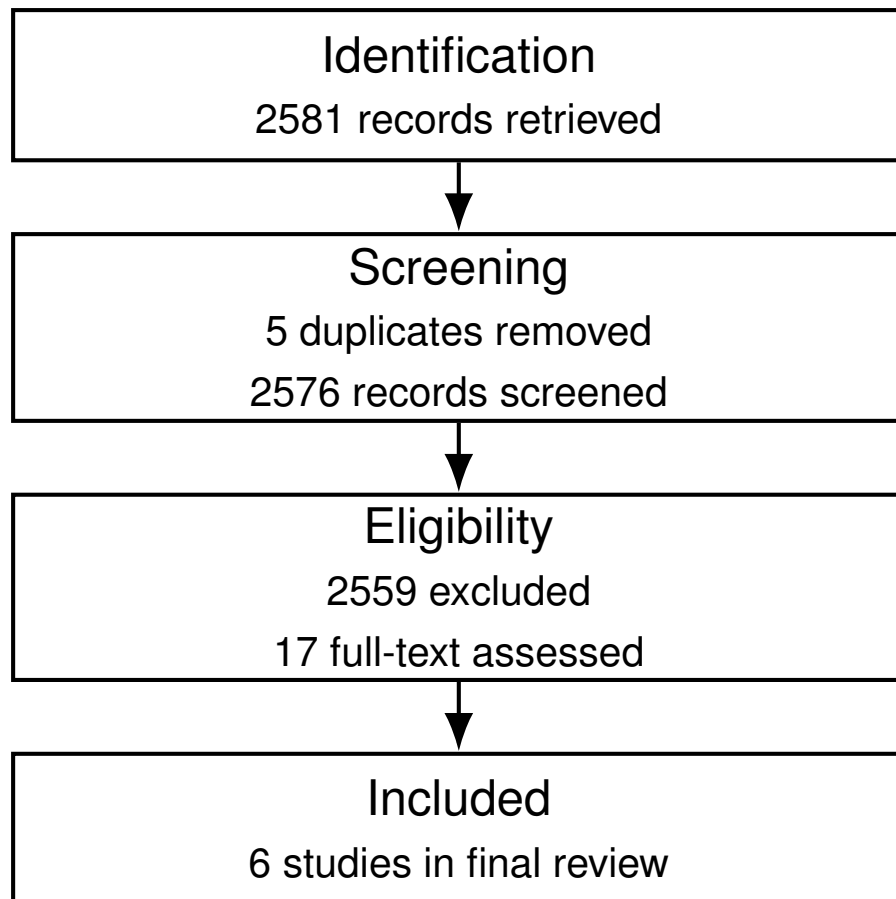


Figure 1 – PRISMA flow diagram.

The study selection process was conducted according to the PRISMA methodology, which is organized into four main stages: identification, screening, eligibility, and inclusion. First, the search strategy retrieved a total of 2581 records across the selected sources. After removing 5 duplicate records, 2576 unique studies remained for screening. Titles and abstracts were then examined, resulting in the exclusion of 2559 records. The remaining 17 studies were assessed through full-text reading for eligibility. After this final evaluation, 6 studies were considered aligned with the objectives and inclusion criteria of this systematic literature review and were therefore included in the final synthesis.

The small final set of six primary studies reflects the strict intersection imposed

by the review scope: decentralized P2P architecture, anonymous or privacy-preserving communication, secure communication mechanisms, and relevance to routing, peer discovery, NAT traversal, or persistence. Studies that addressed only one of these dimensions were therefore excluded from the final synthesis.

The process can be summarized as follows:

- **Identification:** 2581 records retrieved.
- **Screening:** 5 duplicates removed, leaving 2576 records screened.
- **Eligibility:** 2559 records excluded and 17 full-text studies assessed.
- **Included:** 6 studies selected for the final review.

## 5 DATA EXTRACTION AND ANALYSIS

After the selection phase, a total of six primary studies were included for in-depth analysis. In this stage, a structured data extraction process was conducted to systematically identify and compare the main characteristics, contributions, and limitations of each work.

### 5.1 DATA EXTRACTION PROCESS

For each selected study, relevant information was extracted according to pre-defined criteria aligned with the objectives of this research. The extraction focused on identifying architectural, networking, and security aspects of each proposal, including:

- The main problem addressed by the work;
- The type of overlay network employed (e.g., structured, unstructured, hybrid);
- The routing strategy adopted;
- Support for NAT traversal mechanisms;
- The level of anonymity provided;
- Data persistence capabilities;
- Scalability characteristics;
- The use (or absence) of post-quantum cryptographic techniques.

This structured approach enabled a consistent comparison across the selected studies and facilitated the identification of common patterns and research gaps.

### 5.2 COMPARATIVE ANALYSIS

Table 1 summarizes the main characteristics of the selected works based on the defined criteria.

Table 1 – Comparative Analysis of Selected Works

| Work                                                                                                                       | Problem Addressed                                                                                                  | Overlay Type                                                                | Routing Strategy                                                                        | NAT Traversal                    | Anonymity                   | Persistence | Scalability       | Post-Quantum Crypto |
|----------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|----------------------------------|-----------------------------|-------------|-------------------|---------------------|
| Spatial Path Selection (TOSIC; VICIC, 2022)                                                                                | High latency and poor topology distribution in anonymous networks                                                  | Unstructured / Onion-based overlay                                          | Geo-based path selection (geohash, hierarchical routing)                                | ✗                                | ✓ Strong                    | ✗           | ● Moderate        | ✗                   |
| A Fully-Distributed Scalable Peer-to-Peer Protocol for Byzantine-Resilient DHTs (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022) | Byzantine-resilient scalable DHT construction in sparse P2P networks                                               | Structured P2P overlay                                                      | Random walks and verified bit-fixing routing                                            | ✗                                | ✗                           | ✓           | ✓ High            | ✗                   |
| Comparative Analysis of Unstructured P2P Networks (Mehr-Un-Nisa et al., 2019)                                              | Performance limitations and inefficient search in unstructured networks                                            | Unstructured                                                                | Flooding / heuristic search                                                             | ✗                                | ● Medium (varies by system) | ● Limited   | ● Low to moderate | ✗                   |
| Unreachable Peers Communication Scheme (LI; WANG; WANG, 2022)                                                              | Communication between peers behind NAT/firewalls                                                                   | Structured P2P overlay                                                      | Node ID-based routing                                                                   | ✓ (relay + direct communication) | ● Limited                   | ✗           | ● Moderate        | ✗                   |
| Garlic Cast (QIAN et al., 2016)                                                                                            | Anonymous content sharing with low relay overhead and better resilience to churn                                   | Overlay-based anonymous content-sharing network (random-walk / proxy-based) | Random-walk proxy discovery with multi-path clove delivery                              | ✗                                | ✓ Strong (mutual anonymity) | ✗           | ● Moderate        | ✗                   |
| PK-Routing (MIRZAEI; FARAHANI; YAZDANI, 2018)                                                                              | Anonymous data communication without exposing real network identity, with flexible anonymity-efficiency trade-offs | Structured tree-based virtual P2P overlay (public-key addressing)           | Public-key / virtual-ID tree routing with optional relay, multicast, and hybrid schemes | ✗                                | ✓ Strong / configurable     | ✗           | ● Moderate        | ✗                   |

### 5.3 DISCUSSION OF FINDINGS

The comparative analysis highlights that the selected works address different aspects of decentralized network design, but none of them provide a comprehensive solution that integrates all critical requirements simultaneously.

Specifically, it can be observed that:

- Only one work explicitly addresses NAT traversal in a decentralized manner, indicating a significant gap in connectivity solutions for peers behind restrictive network environments;
- Anonymity is explored in some works, particularly through onion-based routing approaches, but often at the cost of increased latency and reduced performance;
- Structured approaches such as DHTs provide scalability and data location capabilities, yet they generally lack strong anonymity guarantees;
- Unstructured networks offer flexibility and robustness to churn, but suffer from inefficient search mechanisms and scalability limitations;
- None of the analyzed works incorporate post-quantum cryptographic mechanisms, revealing a critical gap considering the emerging threats posed by quantum computing.

These observations demonstrate that current solutions tend to focus on isolated challenges, such as routing optimization, scalability, or connectivity, without addressing the problem in an integrated manner.

This fragmentation reinforces the need for a unified architecture capable of combining anonymity, efficient NAT traversal, data persistence, scalability, and post-quantum security, which motivates the proposal presented in this work.

## 6 PROPOSED ARCHITECTURE

### 6.1 DESIGN RATIONALE

The proposed decentralized network layer was not directly derived from a single existing system. Instead, it was conceived as a hybrid architecture that integrates concepts and mechanisms inspired by multiple well-established paradigms in distributed systems and peer-to-peer networks (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).

The design draws from onion routing approaches to provide layered communication and enhanced anonymity (TOSIC; VICIC, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016), from unstructured peer-to-peer systems such as Gnutella to support flexible and resilient message propagation (Mehr-Un-Nisa et al., 2019), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), and from BitTorrent-like mechanisms to enable efficient data distribution and replication (Mehr-Un-Nisa et al., 2019). Additionally, concepts related to depth-oriented search strategies and decentralized resource discovery were considered to improve routing adaptability and network exploration (Mehr-Un-Nisa et al., 2019), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).

These elements were combined into a unified model based on a fully decentralized peer-to-peer architecture, structured in logical layers and supported by distributed hash tables (DHTs) for indexing, routing, and data organization (STOICA et al., 2001), (MAYMOUNKOV; MAZIERES, 2002), (ROWSTRON; DRUSCHEL, 2001), (ZHAO; KUBIATOWICZ; JOSEPH, 2001), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).

Rather than replicating any specific existing system, the proposed approach aims to synthesize the strengths of different paradigms while mitigating their individual limitations (LI; WANG; WANG, 2022), (TOSIC; VICIC, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016). In particular, the architecture seeks to balance anonymity, scalability, data persistence, and connectivity in heterogeneous network environments, including scenarios involving NAT traversal (LI; WANG; WANG, 2022), (TOSIC; VICIC, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).

This hybrid design reflects an intentional effort to move beyond isolated solutions and towards a more comprehensive and adaptable decentralized communication

framework (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHAANI; YAZDANI, 2018), (QIAN et al., 2016).

## 6.2 SYSTEM REQUIREMENTS

### 6.2.1 Functional Requirements

The functional requirements define the externally observable capabilities of the network.

Table 2 – Functional requirements of the proposed architecture.

| Requirement                     | Expected capability                                                                                                                                                                                                                                             |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| P2P communication and discovery | Peers communicate without application servers and dynamically learn reachable participants despite churn (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).                                                                                       |
| Heterogeneous connectivity      | Direct transports, NAT traversal extensions, and relay-assisted communication support peers under different reachability conditions (LI; WANG; WANG, 2022).                                                                                                     |
| Decentralized routing           | DHT-based resolution and hop-by-hop forwarding locate peers and deliver messages without a global network map (STOICA et al., 2001), (MAYMOUNKOV; MAZIERES, 2002), (ROWSTRON; DRUSCHEL, 2001), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).                      |
| Secure exchange and identity    | Cryptographic identities remain separate from physical addresses, and only authorized endpoints access protected content (MIRZAEI; FARAHAANI; YAZDANI, 2018), (QIAN et al., 2016), (BOS et al., 2017), (National Institute of Standards and Technology, 2024a). |
| Distributed persistence         | Data and metadata are stored, retrieved, and replicated across peers for availability and fault tolerance (Mehr-Un-Nisa et al., 2019), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).                                                         |
| Application support             | Stable service abstractions expose messaging, lookup, storage, and retrieval to decentralized applications (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).                                                                                                          |

### 6.2.2 Non-Functional Requirements

The quality requirements constrain how these capabilities should behave.

Table 3 – Non-functional requirements of the proposed architecture.

| Requirement                         | Architectural expectation                                                                                                                                                                                                                                                                  |
|-------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Scalability and efficiency          | Growth should not require global state or flooding, and routing overhead and latency should remain bounded (STOICA et al., 2001), (MAYMOUNKOV; MAZIERES, 2002), (ROWSTRON; DRUSCHEL, 2001), (TOSIC; VICIC, 2022), (Mehr-Un-Nisa et al., 2019), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022). |
| Churn and fault tolerance           | Joins, departures, and failures should degrade local availability rather than invalidate the entire network (TOSIC; VICIC, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).                                                                                         |
| Low centralization                  | Bootstrap and relays may assist connectivity, but no component should control identities, routes, or application state (LI; WANG; WANG, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).                                                                                    |
| Security and post-quantum readiness | Communication requires confidentiality, integrity, authenticity, and replaceable lattice-based primitives (MIRZAEI; FARAHANI; YAZDANI, 2018), (SHOR, 1997), (REGEV, 2009), (BOS et al., 2017), (National Institute of Standards and Technology, 2024a).                                    |
| Privacy orientation                 | Logical identities and indirect routes should reduce direct association with physical endpoints without claiming formal global anonymity (TOSIC; VICIC, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).                                                                    |
| Extensibility and interoperability  | Transport, routing, storage, and cryptographic modules should evolve independently while heterogeneous peers retain protocol compatibility (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).                                                                             |

## 6.3 CONCEPTUAL MODEL

### 6.3.1 Network Entities

The model separates execution, logical identity, distributed coordination, control traffic, and application content.

Table 4 – Core entities of the conceptual model.

| Entity             | Role                                                                                                                                                                                                                                                       |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Physical Node (PN) | Real process or device responsible for transports, physical sessions, discovery, NAT-related procedures, and relay forwarding (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).                                                          |
| Virtual Node (VN)  | Logical cryptographic identity hosted by a PN; one PN may host multiple VNs without publishing their local association (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).                                                                            |
| DHT and DHT Entry  | Distributed coordination substrate and its signed records for physical discovery, routes, content holders, pointers, and tags (STOICA et al., 2001), (MAYMOUNKOV; MAZIERES, 2002), (ROWSTRON; DRUSCHEL, 2001), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022). |
| Protocol Message   | Control-plane envelope used for discovery, lookup, session establishment, forwarding, storage, and maintenance (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).                                                    |
| Data Object        | Application bytes or metadata exchanged, referenced, replicated, or retrieved through the overlay (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).                                                     |

Privacy orientation follows from this separation and hop-by-hop forwarding: physical information is exposed only where connectivity requires it, virtual identities represent application participation, and intermediate nodes require local forwarding state rather than a complete end-to-end route (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).

### 6.3.2 Resource Representation

The overlay treats application resources as byte-oriented objects and assigns their semantics at the application layer (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022). Immutable objects use content hashes, enabling content-addressed lookup and integrity verification; communication destinations use hashed VN identifiers (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016). This uniform representation lets routing, discovery, and persistence operate without interpreting application data or exposing unnecessary physical context.

### 6.3.3 Identity Model

The dual-identity model decouples connectivity from logical participation (LI; WANG; WANG, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018). A PN key pair authenticates transport-level actions, physical sessions, endpoints, and relay coordination; a separate VN key pair authenticates overlay identity, route authorization, and application communication (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016). DHTs use SHA-512 identifiers derived from public keys for compact lookup, while complete keys remain available when signature verification requires them (STOICA et al., 2001), (MAYMOUNKOV; MAZIERES, 2002), (ROWSTRON; DRUSCHEL, 2001), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018). Encryption keys are session-specific: signed ephemeral KEM material establishes fresh symmetric secrets instead of statically reusing identity keys (MIRZAEI; FARAHANI; YAZDANI, 2018), (BOS et al., 2017), (National Institute of Standards and Technology, 2024a).

### 6.3.4 Overlay Abstraction

The overlay exposes logical identifiers, distributed resolution, and hop-by-hop forwarding over any available physical transport (STOICA et al., 2001), (MAYMOUNKOV; MAZIERES, 2002), (ROWSTRON; DRUSCHEL, 2001), (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018). Applications therefore use stable messaging and persistence abstractions without managing addresses, relay selection, or transport-specific behavior; each forwarding node processes only the local routing information required for its next action (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).

### 6.3.5 Kademlia-Inspired DHT Responsibility Model

The DHT uses Kademlia-inspired XOR responsibility because both node and resource identifiers are cryptographic hashes; it does not implement the complete Kademlia routing protocol.

In the proposed model, every DHT record is addressed by a deterministic key derived from a namespace and a logical identifier:

$$\text{dht\_key} = H(\text{namespace} \parallel \text{logical\_key})$$

The namespace identifies the table or protocol domain, while the logical key identifies a node, route, content object, tag, or stable pointer.

Responsibility for each DHT key is assigned to the  $K$  active peers whose identifiers are closest to that key according to the XOR distance metric:

$$\text{distance}(\text{node\_id}, \text{dht\_key}) = \text{node\_id} \oplus \text{dht\_key}$$

The  $K$  closest active peers store replicas and answer queries. This deterministic rule is shared by every namespace; bucket maintenance, parallel iterative lookup, and mature churn handling remain outside the current model.

Potentially large DTT and DDT collections may use a lightweight root and bounded shards to avoid oversized values and hotspots:

$$\text{root\_key} = H(\text{namespace} \parallel \text{logical\_key})$$

Each shard stores only a bounded portion of the full item list:

$$\text{shard\_key}_i = H(\text{namespace} \parallel \text{logical\_key} \parallel i)$$

Each shard is independently assigned by XOR distance. The root carries namespace, logical key, shard count, and update metadata; each shard carries its index and bounded item list. Sharding is an architectural extension for large collections rather than a claim of complete Kademlia behavior.

### 6.3.6 Core Distributed Structures

#### 6.3.6.1 Distributed Pointer Table (DPT)

In addition to the semantic discovery provided by the DTT, the architecture may also employ a lightweight indirection structure called the *Distributed Pointer Table* (DPT). Its purpose is not to store complete version history or detailed mutable metadata directly, but rather to provide a stable distributed resolution point through which peers may locate the most recent known state of a logical resource.

The DPT maps  $H(pk_{vn\_owner} \parallel title)$  to a signed record containing the owner key, stable title, resource type, modification time, and `target_ref`. The title is immutable because changing it creates another key; the target points to the latest state, commonly a DDT-resolvable content object. Owner signatures authorize updates, providing a stable rendezvous point without storing version history in the pointer record.

#### 6.3.6.2 Distributed Tag Table (DTT)

The *Distributed Tag Table* (DTT) is the semantic discovery structure of the proposed architecture. Its role is to provide an efficient distributed entry point for locat-

ing resources through normalized tags or other application-level descriptors, without requiring the requester to know an exact content hash in advance.

The DTT maps  $H(tag)$  to signed, expiring entries containing a resource identifier, resource kind, and publisher VN. Immutable identifiers proceed directly to DDT holder resolution; mutable identifiers first resolve through the DPT. Applications may combine tag results by intersection, union, ranking, or partial matching, while the DTT remains independent of content storage and provider discovery.

#### 6.3.6.3 Distributed Data Table (DDT)

The *Distributed Data Table* (DDT) is the distributed availability structure responsible for content-addressed object resolution in the proposed architecture. Its role is not to store the content object itself, but to maintain distributed references to virtual nodes that currently advertise possession of a given object.

The DDT maps  $H(content)$  to metadata and signed, expiring VN holder advertisements. The hash covers the exact object bytes; title, type, tags, and holder metadata are associated record fields unless encoded into the object itself. The DDT locates providers but never stores the content payload.

#### 6.3.6.4 Distributed Route Table (DRT)

The *Distributed Route Table* (DRT) is the distributed routing-assistance structure of the proposed architecture. Its role is to help requesters reach a target virtual node after that node has already been identified through other resolution stages.

The DRT maps  $H(pk_{vn})$  to expiring physical entry points. Each route entry carries the PN identifier, PN and VN authorization signatures, measured RTT, and expiration metadata. Entries are soft-state hints refreshed by maintenance; they initiate communication without exposing the complete physical path.

#### 6.3.6.5 Distributed Physical Nodes Table (DPNT)

The *Distributed Physical Nodes Table* (DPNT) is an auxiliary distributed support structure responsible for maintaining recoverable operational information about physical nodes. Its purpose is not to replace direct peer discovery, bootstrap procedures, or local neighbor views, but rather to extend them by preserving validated physical-node descriptors in a distributed form.

The DPNT maps  $H(pk_{pn})$  to a signed physical descriptor containing the public key, advertised endpoints and transports, reachability and relay capabilities, protocol compatibility, validation metadata, and status. It supplements bootstrap and local peer

exchange when a known PN is absent locally; it is not a global authoritative membership registry.

6.4 LAYERED ARCHITECTURE

Figure 2 summarizes how the proposed modules are organized across the physical, overlay, service, and application levels. The figure is presented after this brief introduction so that the layered view can be interpreted in relation to the modular design rationale described above.

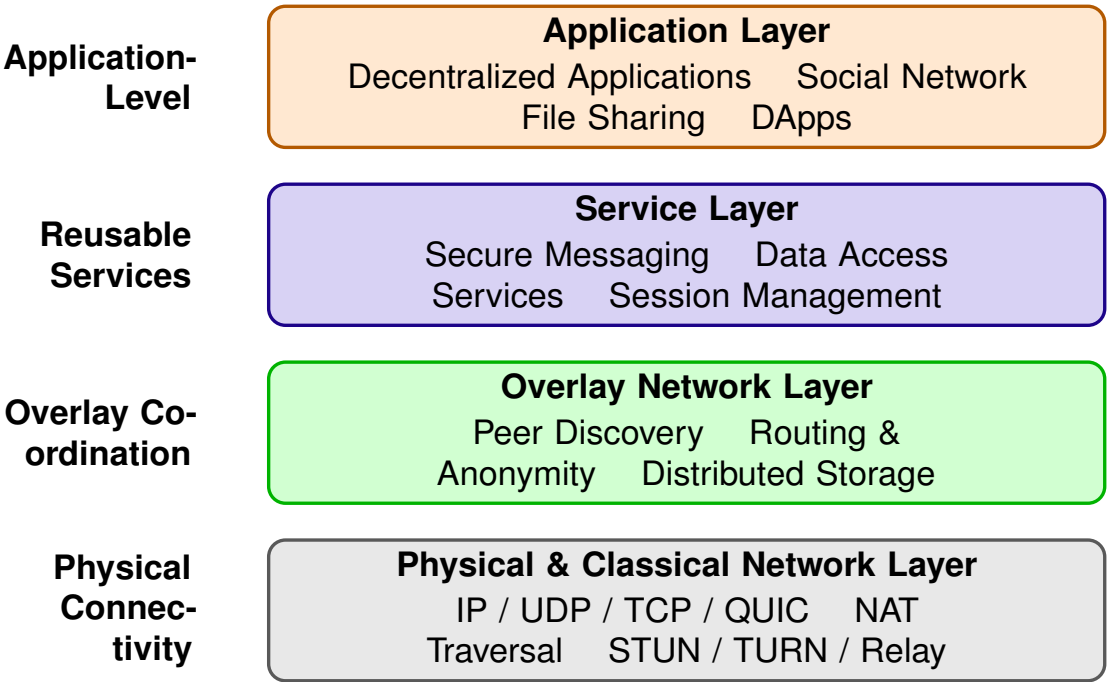


Figure 2 – Layered architecture.

Although the proposed system is defined as a modular architecture, its internal organization is described through a layered structure. Modularity refers to the separation of the system into distinct components that may evolve or be replaced independently, while layering provides a clearer structural view of how these components interact across different levels of abstraction. In this work, the layered organization is adopted to separate physical connectivity concerns, overlay coordination mechanisms, reusable service abstractions, and application-level functionalities.

6.4.1 Physical and Classical Network Layer

This layer is responsible for the underlying classical network communication between physical nodes (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022). It encompasses transport protocols such as TCP and UDP, as well as the mechanisms required to establish connectivity between devices, including

NAT traversal and relay-assisted communication (LI; WANG; WANG, 2022). In addition, this layer is also responsible for physical peer discovery, since the initial knowledge of reachable physical participants is obtained through bootstrap procedures, direct peer exchange, and other classical communication mechanisms that operate independently of overlay identities (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).

Its primary role is to provide a basic communication substrate between physical devices, without direct awareness of overlay-level logical identities, identifier-based routing, or application semantics (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022). When the available transport mechanism is based on UDP Hole Punching, this layer may also incorporate complementary reliability functions, such as delivery assurance and packet ordering, in order to provide a more stable communication channel despite the unordered and unreliable nature of UDP itself (LI; WANG; WANG, 2022).

#### **6.4.2 Overlay Network Layer**

The overlay network layer is built on top of the classical communication layer and provides decentralized interaction through logical identifiers (STOICA et al., 2001), (MAYMOUNKOV; MAZIERES, 2002), (ROWSTRON; DRUSCHEL, 2001), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018). Its main responsibility is to organize routing, lookup, and communication at the logical level, allowing nodes to interact through virtual identities rather than direct dependence on physical addressing (STOICA et al., 2001), (MAYMOUNKOV; MAZIERES, 2002), (ROWSTRON; DRUSCHEL, 2001), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018).

Unlike the classical layer, which is responsible for physical connectivity and physical peer discovery, the overlay layer operates over the communication substrate already established and focuses on identifier-based forwarding, logical resolution, and coordination among distributed entities (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016). This abstraction enables a consistent interaction model across the network while decoupling logical communication from the specific details of the underlying transport mechanisms (STOICA et al., 2001), (MAYMOUNKOV; MAZIERES, 2002), (ROWSTRON; DRUSCHEL, 2001), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018).

#### **6.4.3 Service Layer**

The service layer provides reusable functionalities on top of the overlay, such as distributed storage, messaging primitives, and data retrieval mechanisms (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).

This layer exposes higher-level abstractions that simplify the development of applications, allowing them to use network capabilities without handling routing or communication details directly (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).

#### 6.4.4 Application Layer

The application layer represents end-user systems built on top of the network, such as decentralized social platforms or communication tools (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).

Applications interact with the service layer to perform operations such as sending messages or storing data, without needing to manage lower-level networking or security mechanisms (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).

### 6.5 SYSTEM COMPONENTS

#### 6.5.1 Peer Management

The Peer Management component is responsible for maintaining the local representation of entities participating in the overlay network (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016). In the proposed architecture, participation is modeled through a strict separation between *physical nodes* and *virtual nodes*, so that logical identities in the overlay are decoupled from the underlying hosts that execute them (LI; WANG; WANG, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018).

A physical node may host multiple virtual nodes simultaneously, establishing a  $1 : N$  relationship between execution environment and logical identities. However, each virtual node is associated with only one physical host at a given time. This design allows a physical device to act as an execution substrate for multiple overlay identities, while preventing the network from learning explicit ownership relations between them (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).

The *physical node public key* ( $pk_{physical\_node}$ ) is not used as the main logical identity in the overlay. Instead, its role is limited to network access and reachability support (LI; WANG; WANG, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018). In particular, physical node identifiers may appear in routing-related structures only as entry points through which a given virtual node may be reached (LI; WANG; WANG, 2022). For example, the overlay may maintain associations of the form

$$H(pk_{virtual\_node}) \rightarrow \{pk_{physical\_node}^{(1)}, pk_{physical\_node}^{(2)}, \dots\},$$

where the listed physical nodes represent candidate entry points rather than explicit ownership relations (LI; WANG; WANG, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018).

In particular, such entry points do not necessarily correspond to the physical host currently responsible for the virtual node, but may instead denote third-party physical nodes that have agreed to act as reachable ingress points toward it.

The *virtual node public key* ( $pk_{virtual\_node}$ ) represents the primary logical identity of an entity in the overlay (MIRZAEI; FARAHANI; YAZDANI, 2018). To reduce storage and communication overhead, the network references virtual nodes through a derived identifier, denoted here as  $H(pk_{virtual\_node})$ , where  $H(\cdot)$  is a cryptographic hash function (STOICA et al., 2001), (MAYMOUNKOV; MAZIERES, 2002), (ROWSTRON; DRUSCHEL, 2001), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018). The complete public key does not need to be broadly exposed at the overlay level and may remain known only to parties that actually require it for cryptographic interaction (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).

VNs may be persistent or expiring and may migrate between hosts without publishing a permanent PN association (LI; WANG; WANG, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016). Peer Management maintains controlled local identities plus limited remote metadata: identifiers, candidate entry points, transports, reachability, capabilities, freshness, and validation state. Physical peers are learned through bootstrap, exchange, observation, and DPNT fallback, then periodically validated and pruned (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).

### 6.5.2 Connection Management

Connection management begins only after a physical peer has been discovered and a reachable transport endpoint has been selected. It then establishes the authenticated communication context used by higher-level protocols, while transport selection and cryptographic session establishment remain separate responsibilities.

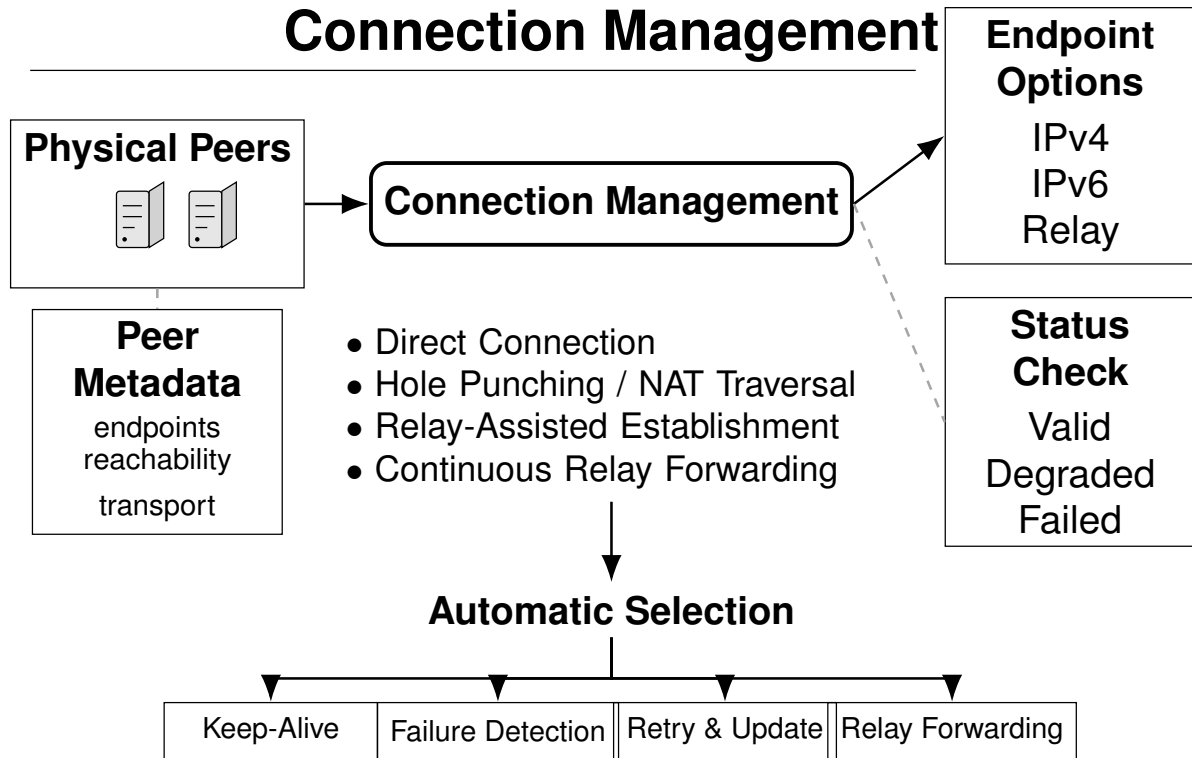


Figure 3 – Connection Management diagram.

Connection Management provides a common PN transport interface so overlay protocols remain independent of TCP, UDP, relay, or future connectivity mechanisms (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018). Endpoint metadata belongs only to PNs:

$$H(pk_{physical\_node}) \rightarrow \{endpoint_1, endpoint_2, \dots\},$$

where each endpoint may represent a candidate communication method such as IPv4, IPv6, direct transport ports, relay endpoints, or parameters relevant to NAT traversal and hole punching (LI; WANG; WANG, 2022).

Multiple candidates may coexist with reachability and validation state. Selection prefers lower-dependency methods while retaining fallback connectivity (LI; WANG; WANG, 2022):

1. direct connection;
2. hole punching or assisted NAT traversal;
3. relay-assisted establishment;
4. continuous relay forwarding.

Selection considers advertised support, reachability, and observed conditions. Maintenance performs keepalives, endpoint validation, failure detection, and retries;

another candidate is reconsidered when the active method fails. A relay may assist direct establishment or remain as the forwarding path when direct reachability is unavailable (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).

### 6.5.3 Routing Mechanism

The Routing Mechanism is responsible for enabling message delivery across the overlay without requiring global knowledge of the network topology (STOICA et al., 2001), (MAYMOUNKOV; MAZIERES, 2002), (ROWSTRON; DRUSCHEL, 2001), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018). In the proposed architecture, routing is identity-oriented and privacy-oriented: messages are routed toward virtual identities, while intermediate physical nodes should not learn the complete origin-destination relation of a communication (TOSIC; VICIC, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).

Two operational phases must be distinguished. The first phase is *route construction and publication*, in which a virtual node obtains one or more inbound route entries and publishes the corresponding reachability information in the Distributed Routing Table (DRT). The second phase is *route resolution and execution*, in which another peer queries the DRT, selects one of the available entry points, and sends traffic through the already established route state. This distinction avoids treating the DRT as a table of complete end-to-end paths. The DRT stores candidate entry points and authorization metadata, while the actual forwarding state remains local to the physical nodes participating in each route (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018).

Routing resolution begins with a lookup in the DRT. More specifically, the DRT maps the identifier of a virtual node, represented as  $H(pk_{virtual\_node})$ , to a set of candidate physical entry points capable of receiving traffic toward that virtual destination (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018). Thus, the DRT does not store full hop sequences, nor does it prescribe the algorithm used to construct them. It only exposes the distributed information needed for a requester to start communication toward a reachable virtual identity (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018).

## Distributed Routing Table (DRT)

The DRT indirectly maps a virtual node identifier to physical entry points authorized to receive its traffic.

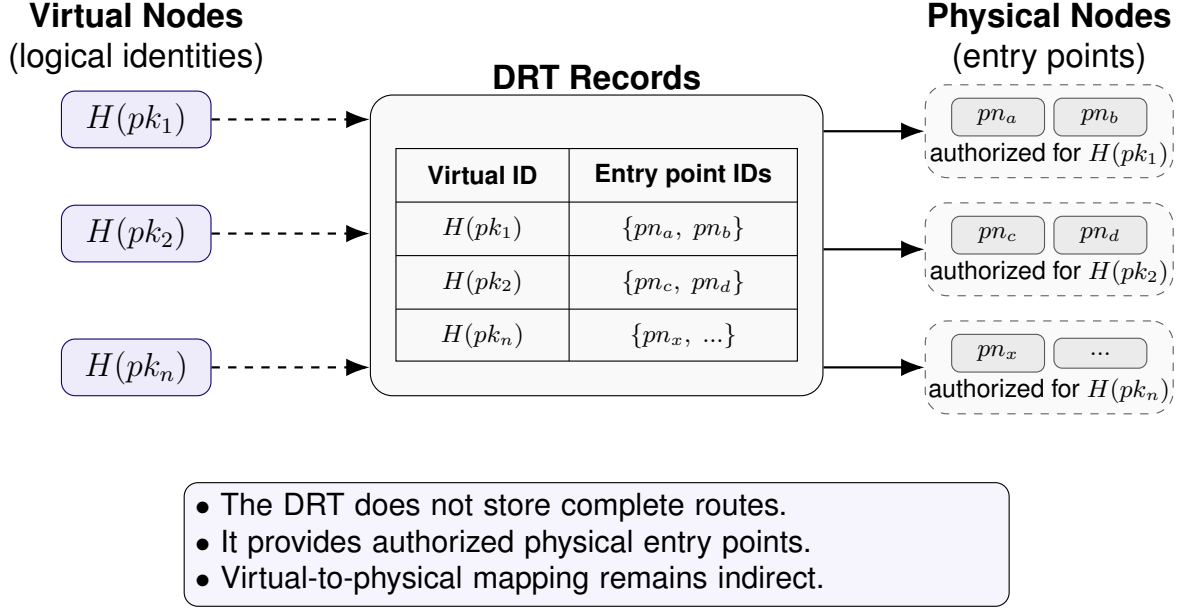


Figure 4 – Distributed Routing Table.

Formally, the DRT may contain records of the form

$$H(pk_{virtual\_node}) \rightarrow \{H(pk_{physical\_node}^{(1)}), H(pk_{physical\_node}^{(2)}), \dots\},$$

where the listed physical nodes are entry points authorized to receive traffic destined to the corresponding virtual identity (LI; WANG; WANG, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018). This preserves the indirect relation between virtual and physical entities, since the network never exposes an explicit and permanent binding between them (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).

## Strategy-Based Route Construction

The route protocol keeps the lifecycle stable. The active route strategy chooses each next physical hop and defines when construction should finish.

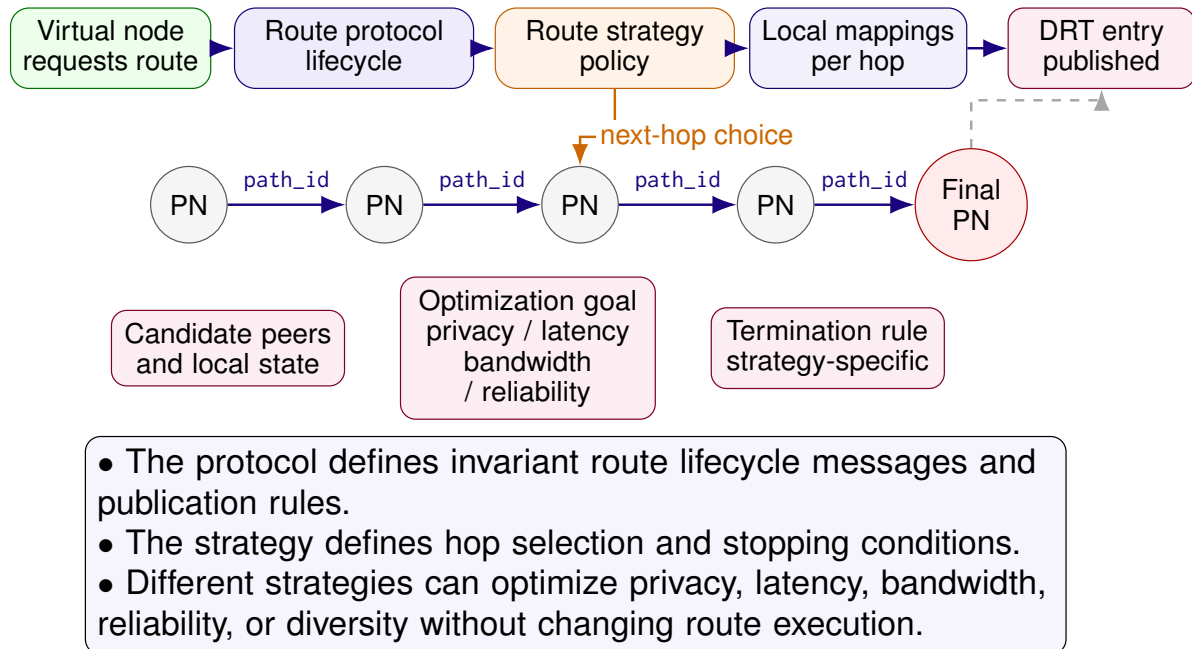
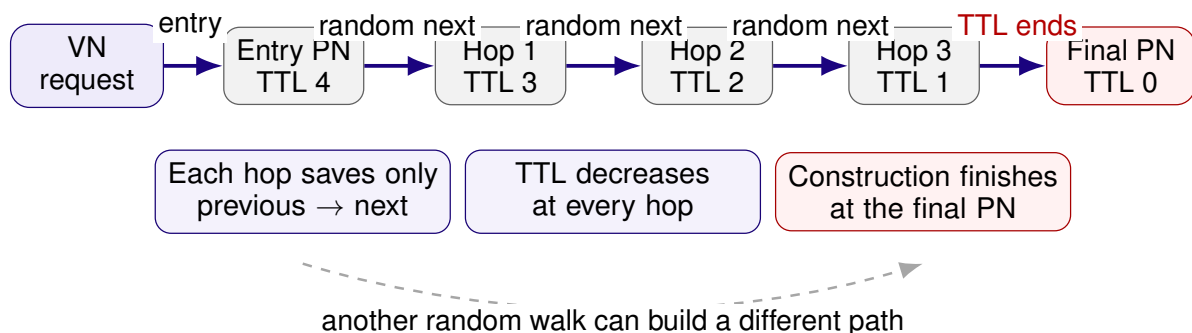


Figure 5 – Generic route construction through a route strategy.

## TTL-Based Hop-by-Hop Route Example

A route is built incrementally: each physical node stores only its local mapping and chooses the next hop while TTL remains valid.



- Each hop chooses one next physical node.
- TTL decreases at every forwarding step.
- The route emerges during forwarding.

Figure 6 – Example of hop-by-hop route construction with a TTL-based strategy.

Route construction proceeds hop by hop until a final PN accepts the entry-point role. The protocol defines lifecycle, authorization, validation, publication, and execution; the strategy selects each next hop and its termination rule. Figure 6 shows *random\_walk\_ttl*, where each hop consumes a temporal budget, but other strategies may use hops, latency, bandwidth, reliability, diversity, or preselected onion-like paths (TOSIC; VICIC, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).

Each intermediate PN remaps the received *path\_id* to local state and knows only adjacent forwarding actions. The final intermediate hop learns the final PN, not the destination VN; the final PN validates authorization and publishes the signed DRT entry (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016). Once active, *ROUTE\_DATA* forwarding is a local mapping operation independent of the construction strategy.

Content discovery first resolves providers through DTT/DPT/DDT and then routes toward a holder VN. TTL and *seen\_hash* are MVP safeguards, while future strategies may use other termination and deduplication controls. Route failure is detected by timeout and triggers another route or entry point rather than automatic mid-route repair (LI; WANG; WANG, 2022), (Mehr-Un-Nisa et al., 2019), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).

#### 6.5.4 Route Strategy Abstraction

The route lifecycle is stable, while a replaceable strategy selects hops, termination conditions, and optimization goals for privacy, latency, bandwidth, stability, or diversity (TOSIC; VICIC, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016):

$$route\_construction = route\_lifecycle + route\_strategy$$

This makes TTL random walk one policy rather than a routing-layer requirement and allows applications to trade latency, capacity, stability, path diversity, and predictability without changing route publication or execution.

Table 5 summarizes examples of route strategies that may be supported by the architecture.

✓ Implemented and validated    ◆ Architectural extension.

Table 5 – Examples of route strategies supported by the architectural model.

| Status | Strategy                    | Architectural purpose                                                                                            |
|--------|-----------------------------|------------------------------------------------------------------------------------------------------------------|
| ✓      | Random walk by TTL          | Uses a temporal budget to create dynamic, less predictable paths.                                                |
| ◆      | Random walk by maximum hops | Bounds route depth without guaranteeing latency.                                                                 |
| ◆      | Minimum bandwidth           | Requires minimum capacity for content transfer or streaming.                                                     |
| ◆      | Minimum latency             | Prioritizes low observed delay for interactive traffic.                                                          |
| ◆      | Onion-like routing          | Preselects hops and layered forwarding information, requiring greater prior peer knowledge (TOSIC; VICIC, 2022). |
| ◆      | Reliability-based routing   | Uses availability and failure history for long-lived sessions.                                                   |
| ◆      | Reputation-based routing    | Uses behavioral scores while requiring protection against reputation manipulation.                               |
| ◆      | Network diversity routing   | Reduces concentration across networks, providers, or administrative domains.                                     |
| ◆      | Cost-balanced routing       | Balances latency, bandwidth, reliability, and diversity through a weighted score.                                |

A generic cost-balanced route selection policy can be represented as a weighted function:

$$score(route) = w_1 \cdot latency + w_2 \cdot bandwidth^{-1} + w_3 \cdot failure\_rate + w_4 \cdot concentration$$

Lower scores represent preferred routes. Applications choose the weights: messaging may favor latency and diversity, while content transfer may favor bandwidth and reliability. Because policy remains isolated, strategies can be compared experimentally without changing physical forwarding, virtual identity, or application interfaces.

### 6.5.5 Discovery Module

## Peer Discovery Process

Public peers bootstrap the network, learn about private peers, and share reachability metadata for relay or hole-punch attempts.

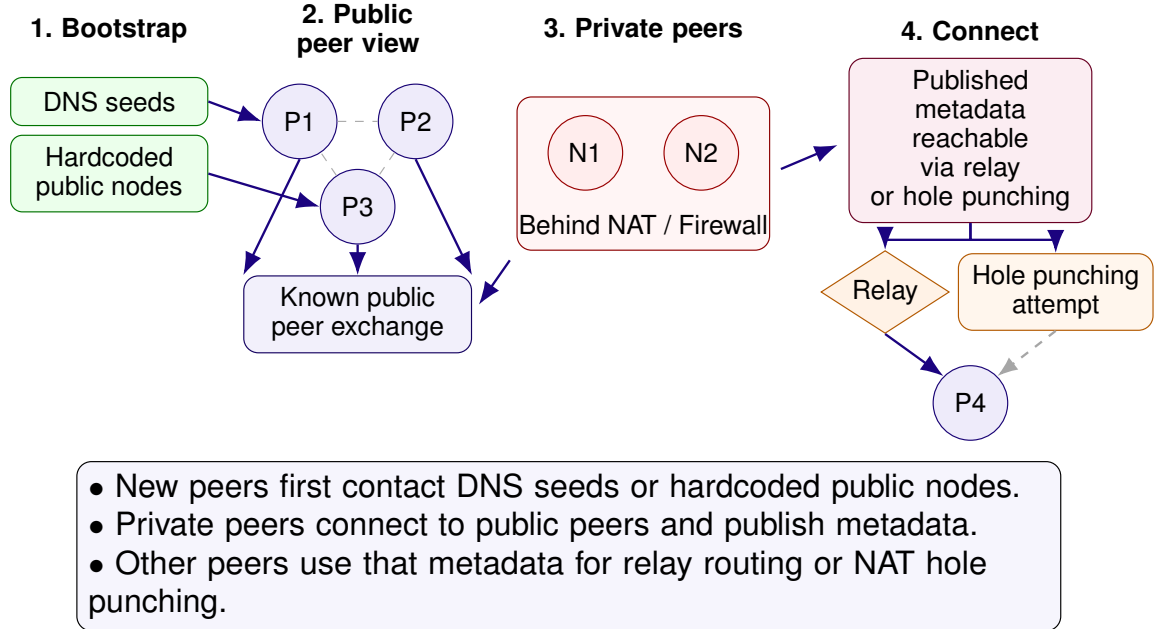


Figure 7 – Peer discovery process.

Discovery learns physical peers, not virtual identities or routes. Public bootstrap endpoints provide initial contacts; direct observation and peer exchange expand each node's local view; maintenance validates freshness, reachability, transports, relay support, latency, and endpoint alternatives (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016). Indirectly learned descriptors remain untrusted until validation.

The DPNT complements this local process. Validated PN descriptors may be published for later recovery when local endpoint information is missing, but bootstrap and peer exchange remain the primary discovery paths. DPNT data is best-effort soft state, periodically renewed and expired rather than treated as authoritative membership (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).

This separation keeps responsibilities clear: Discovery acquires PN candidates, Peer Management validates and stores them, Connection Management selects a transport, and Routing resolves VNs through the DRT.

### 6.5.6 Cryptographic Module

The Cryptographic Module protects physical and virtual sessions plus distributed records while preserving the PN/VN identity boundary (LI; WANG; WANG, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016), (BOS et al., 2017), (National Institute of Standards and Technology, 2024a). Long-term ML-DSA keys authenticate identities, handshakes, and signed records; signed ephemeral ML-KEM material establishes fresh session secrets; AES-256-GCM-SIV protects payloads and selected public metadata through AAD; SHA-512 derives identifiers and content references (BOS et al., 2017), (National Institute of Standards and Technology, 2024a), (National Institute of Standards and Technology, 2024b), (GUERON; LANGLEY; LINDELL, 2019). The suite is fixed for interoperability in the MVP but remains replaceable at the architectural level.

Figure 8 connects these mechanisms to their operational roles. SHA-512 derives compact physical- and virtual-node identifiers; long-term ML-DSA keys authenticate identities, handshake material, and signed records; ephemeral ML-KEM material establishes a fresh shared secret when a physical or virtual session is created; and AES-256-GCM-SIV protects subsequent session payloads while AAD authenticates selected public envelope fields. Publication proof-of-work is applied before DHT writes and complements signatures by imposing a generation cost without replacing authenticity checks.

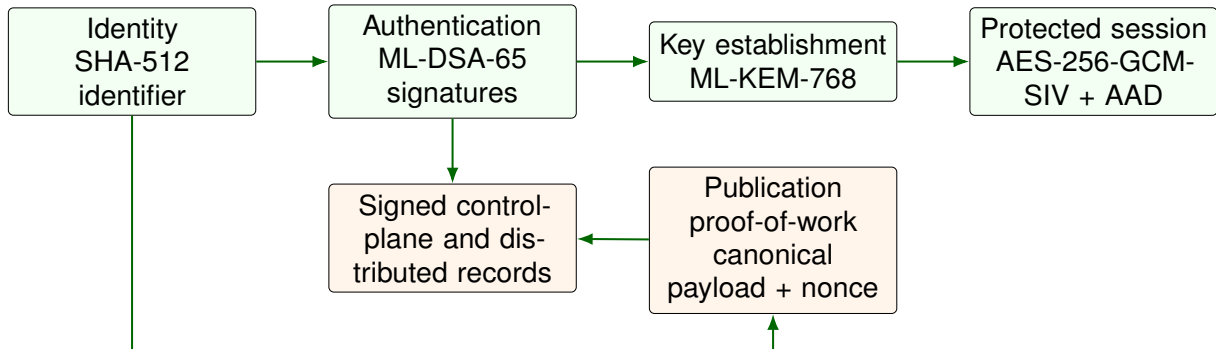


Figure 8 – Security mechanisms and the stages at which they are applied.

The same authenticated handshake runs over direct or relayed transports. Physical sessions protect adjacent peers; virtual sessions protect communicating VNs end to end. Freshness fields, nonces, message identifiers, and duplicate controls provide replay resistance. Relays and storage peers process only the layer required by their role and do not receive virtual-session plaintext (LI; WANG; WANG, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).

### 6.5.7 Storage Management

Storage Management handles immutable, content-addressed application objects. A content change produces a new hash; DPT pointers represent mutable logical state without changing stored objects (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).

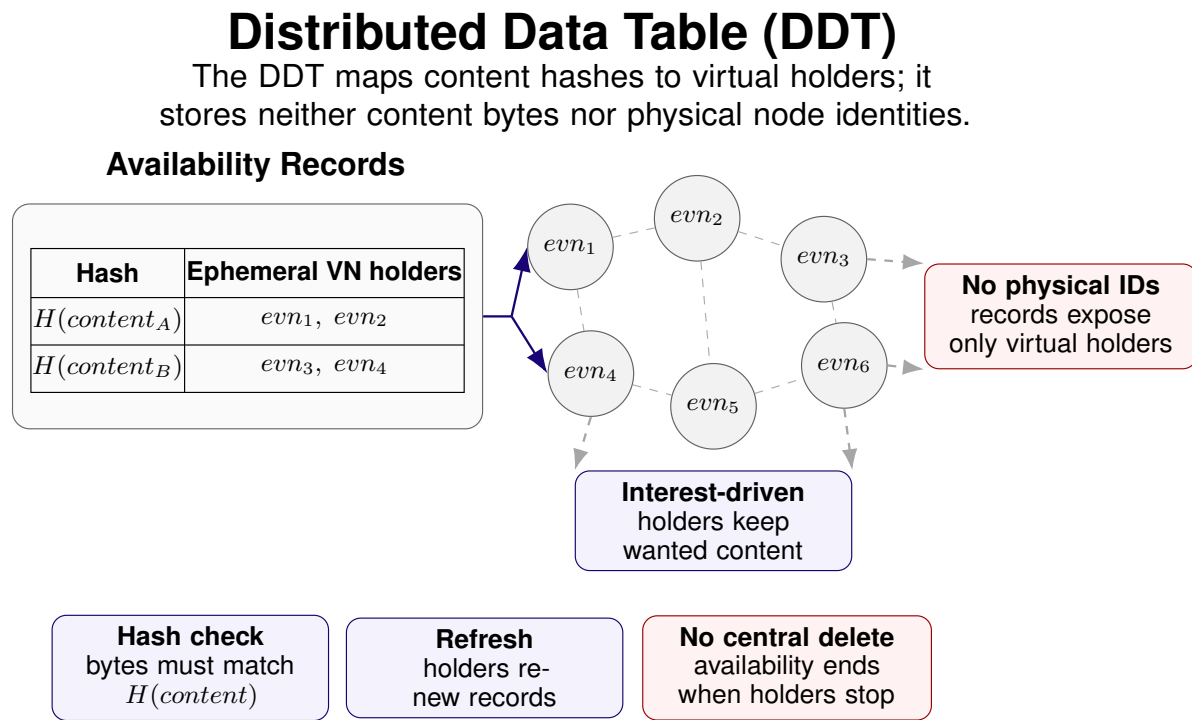


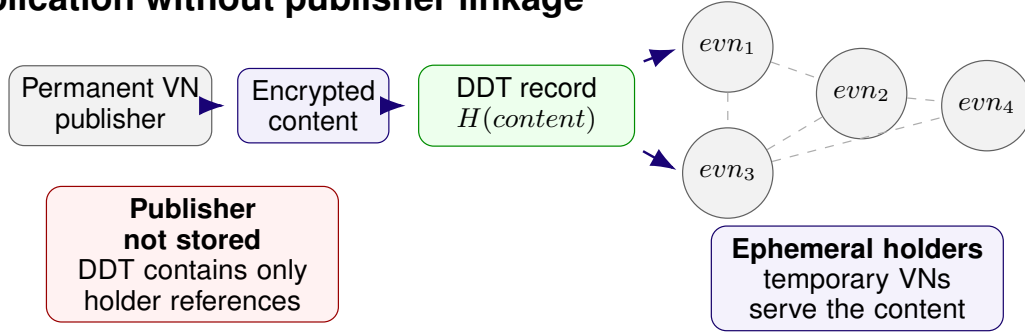
Figure 9 – Distributed Data Table.

The DDT advertises expiring holder VNs rather than physical hosts or content bytes (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016). Replication is interest-driven: consumers that retain an object may become providers, while uninterested peers incur no storage obligation (Mehr-Un-Nisa et al., 2019), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022). Public objects remain readable; protected objects require application-managed keys even though transport is always session-protected (QIAN et al., 2016), (BOS et al., 2017), (National Institute of Standards and Technology, 2024a).

## Ephemeral Virtual Nodes

Temporary holder identities avoid permanent publisher linkage.

### Publication without publisher linkage



### Expiration and renewal

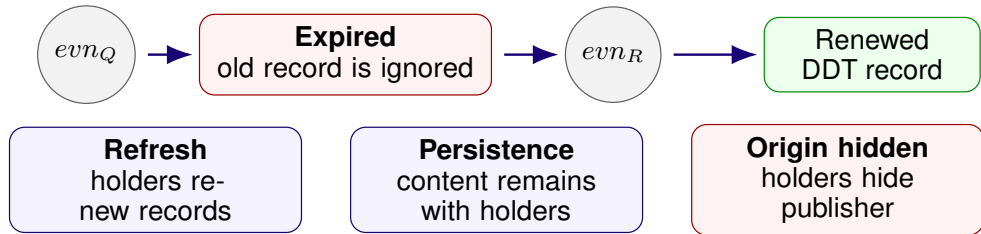


Figure 10 – Ephemeral Virtual Nodes.

Holder records are renewed or expire, keeping discovery aligned with active providers. Retrieved bytes must match their content hash. The network cannot guarantee global deletion after replication; an object disappears only when holders remove it and advertisements expire (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016). Location and access control remain separate: the DDT locates holders, while applications define who can interpret protected content.

## 6.6 COMMUNICATION PROTOCOLS

Communication follows four transport-independent stages: discover and validate PNs; establish authenticated physical or virtual sessions; resolve a VN or content provider and obtain a route; exchange protected data (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016). Fixed message semantics, explicit identifiers, bounded validity, signatures, and freshness controls support interoperability under churn and replay resistance. Control messages coordinate the network; data messages carry application payloads. Every participant receives only the metadata required by its local role.

### 6.6.1 Peer Discovery Protocol

Bootstrap returns public PN endpoints only. After an authenticated physical session is established, one-hop discovery exchanges bounded samples of validated peers and advertised transport, reachability, relay, and entry-point capabilities (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016). Candidates enter the local table only after liveness validation; background maintenance refreshes usable peers and removes repeated failures.

The DPNT stores renewable descriptors for already validated PNs and is queried only when local information is missing. It complements rather than replaces bootstrap and peer exchange, never carries VN information, and remains best-effort soft state rather than global membership (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).

### 6.6.2 Connection Establishment

Reachability alone is insufficient: a session becomes active only after authenticated establishment of a fresh symmetric key (LI; WANG; WANG, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (BOS et al., 2017), (National Institute of Standards and Technology, 2024a). Transport selection prefers direct methods and falls back to assisted traversal or relay, but does not change handshake semantics.

The responder returns a signed ephemeral ML-KEM public key; the initiator verifies it, encapsulates a secret, signs the response, and sends the ciphertext; the responder decapsulates and confirms with encrypted control traffic. The procedure runs independently for adjacent PNs and communicating VNs, producing hop-by-hop and end-to-end keys. Bounded retries and keepalives prevent stale establishment state; persistent failure expires the session and requires a new attempt (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (BOS et al., 2017), (National Institute of Standards and Technology, 2024a).

### 6.6.3 Routing and Lookup

DRT lookup maps a destination VN identifier to authorized PN entry points; DDT lookup maps a content hash to holder VNs (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016). Content retrieval therefore resolves a holder in the DDT and then resolves that VN in the DRT. Neither table returns a complete route.

Construction remaps *path\_id* at every hop, so intermediates retain only incoming/outgoing local mappings and adjacent peers (MIRZAEI; FARAHANI; YAZDANI,

2018), (QIAN et al., 2016). The MVP's TTL bounds construction, not later reuse. A failed active route is replaced through a new lookup rather than repaired locally. This separates distributed resolution from private local forwarding state and supports messaging and content without flooding.

#### 6.6.4 Data Exchange Protocol

Data Exchange uses an active route and session; it does not discover peers, construct routes, or establish keys. The same encrypted virtual-session channel carries messages and byte-range content responses, while physical hops see only their protected forwarding layer (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016), (BOS et al., 2017), (National Institute of Standards and Technology, 2024a).

Reliable envelopes add message identifiers, sequence numbers, acknowledgments, duplicate suppression, ordered buffering, and bounded retransmission at physical and virtual session scopes. Routes and sessions may be reused while healthy. Content is accepted only when its recomputed hash matches the requested identifier; timeout, invalid data, or route failure triggers another holder, route, or session. Public content relies on session protection, while protected objects additionally require an application-level content key.

### 6.7 SECURITY MODEL

The security model treats a signing public key as a self-defining identity and separates persistent identity material from authenticated ephemeral sessions (MIRZAEI; FARAHANI; YAZDANI, 2018), (BOS et al., 2017), (National Institute of Standards and Technology, 2024a).

#### 6.7.1 Identity and Authentication

PN, persistent VN, and optional ephemeral VN identities use independent long-term signing keys (LI; WANG; WANG, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016). Possession of the private key proves control without a certificate authority. Mutual session authentication binds signed ephemeral KEM material to that identity:

$$pk_{sign} \rightarrow \text{Sign}(pk_{kex}) \rightarrow \text{Session Key}$$

After signature validation and encrypted confirmation, the session key authenticates traffic within that session. PN and VN identities remain independent so connectivity-

level presence is not published as application identity (LI; WANG; WANG, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).

### 6.7.2 Encryption Mechanisms

AES-256-GCM-SIV protects post-handshake traffic with confidentiality, integrity, and authentication (MIRZAEI; FARAHANI; YAZDANI, 2018), (BOS et al., 2017), (National Institute of Standards and Technology, 2024a), (GUERON; LANGLEY; LINDELL, 2019). Fresh nonces are tracked per key. AAD covers visible fields that must remain stable, such as physical headers, route direction, virtual session identifiers, encryption flags, and control-message types; remapped hop-local *path\_id* values are excluded from end-to-end AAD. Independent physical and virtual layers protect each hop and the final application payload, while optional object encryption protects stored private content.

### 6.7.3 Post-Quantum Cryptography

The model uses post-quantum public-key primitives and classical symmetric encryption: ML-DSA signs identities and control artifacts, ML-KEM establishes secrets, and AES protects bulk traffic (SHOR, 1997), (REGEV, 2009), (BOS et al., 2017), (National Institute of Standards and Technology, 2024a), (National Institute of Standards and Technology, 2024b), (GUERON; LANGLEY; LINDELL, 2019). It distinguishes:

- a persistent signing key pair used as the long-term identity of the node;
- an ephemeral asymmetric key exchange pair generated per session;
- optional ephemeral data identities for temporary objects;
- a symmetric session key derived from the authenticated key exchange.

The MVP fixes this suite for compatibility; future versioned suites may replace algorithms without changing the identity/session roles.

### 6.7.4 Key Management

Long-term keys are generated and stored locally and remain stable until intentionally replaced; session KEM pairs and symmetric keys are fresh and independent, including across concurrent sessions (MIRZAEI; FARAHANI; YAZDANI, 2018), (BOS et al., 2017), (National Institute of Standards and Technology, 2024a). Revocation uses control messages and reannouncement in the MVP. Private content may use separate object keys. Without backup, loss of a signing private key means loss of that identity; recovery phrases and mature distributed revocation remain future work.

### 6.7.5 Abuse Control and Sybil Resistance

Open P2P systems cannot eliminate Sybil, Eclipse, flooding, or routing manipulation without authority or resource cost (DOUCEUR, 2002). The architecture therefore combines complementary local mitigations rather than claiming perfect prevention. Optional identity admission may require a nonce satisfying  $H(pk_{pn} \parallel nonce) < target$ , making bulk identifier generation more expensive (BAUMGART; MIES, 2007).

Table 6 – Layered abuse and Sybil mitigations.

| Mitigation        | Purpose                                                                                                                                             |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| Origin diversity  | Score IP, subnet, IPv6 prefix, AS, endpoint reuse, and behavior; prefer throttling over absolute rejection because legitimate peers may share NATs. |
| DHT-set diversity | Prevent one apparent origin from dominating the $K$ responsible peers, accepting slightly farther but more diverse candidates.                      |
| Signed records    | Treat DHT peers as storage providers, not authorities: DRT, DDT, DPT, and DTT entries require the relevant owner signatures.                        |
| Expiration        | Renew legitimate soft state instead of granting arbitrary deletion authority to responsible peers.                                                  |
| Redundant lookup  | Compare parallel or partially disjoint results to reduce Eclipse-style influence (BAUMGART; MIES, 2007), (CASTRO et al., 2002).                     |
| Flood control     | Apply rate limits, token buckets, bounded queues, and higher cost to writes, advertisements, and repeated lookups (HEINANEN; GUERIN, 1999).         |
| Bounded resources | Cap pending requests, bytes, connections, record sizes, namespace writes, and shard updates.                                                        |

A simplified local abuse policy may be represented as:

$$\begin{aligned}
 \text{score}(\text{peer}) = & w_1 \cdot \text{invalid\_signatures} \\
 & + w_2 \cdot \text{request\_rate} \\
 & + w_3 \cdot \text{failed\_lookups} \\
 & + w_4 \cdot \text{origin\_concentration} \\
 & + w_5 \cdot \text{malformed\_messages}
 \end{aligned}$$

If the score exceeds predefined thresholds, the node may reduce the peer's priority, delay responses, require additional proof-of-work, ignore selected requests, or temporarily quarantine the peer. This decision remains local to each node and does not require a global reputation system.

Together, these controls raise the computational, network, and bandwidth cost of abuse without claiming immunity to Sybil or flooding attacks.

### 6.7.6 Publication Proof-of-Work

In addition to identity-level controls, DHT writes may require lightweight proof-of-work so that publication remains inexpensive to verify but costly to flood.

For each DHT publication, the record key is first derived according to the standard DHT key derivation rule:

$$\text{record\_key} = H(\text{namespace} \parallel \text{logical\_key})$$

The publisher must then find a nonce such that the hash of the record key concatenated with the nonce satisfies the required difficulty condition:

$$P = H(\text{record\_key} \parallel \text{payload} \parallel \text{nonce})$$

The publication proof is considered valid only if the resulting hash satisfies the target condition:

$$P < \text{target}$$

Equivalently, this condition may be interpreted as requiring the proof hash to contain at least  $N$  leading zero bits:

$$\text{leading\_zero\_bits}(P) \geq N$$

The resulting record carries the payload, proof nonce, signature, and expiration:

$$\text{record\_key} \rightarrow \{\text{payload}, \text{nonce}, \text{signature}, \text{expires\_at}\}$$

Responsible peers recompute the hash and check the target before accepting or refreshing route advertisements, holder announcements, tags, pointers, or shards. Proof-of-work complements, rather than replaces, signatures, expiration, rate limits, and diversity constraints.

The proof-of-work rule used in the implementation should be interpreted as a refinement of the generic architectural model. While this section describes publication work abstractly as a hash over the record key, payload, and nonce, the implemented version derives the hashed material from a canonical payload representation, namespace

information, the physical DHT key, and, when required, the relevant identity context. This avoids ambiguity when records aggregate multiple independently signed entries while preserving the same architectural purpose: making DHT writes inexpensive to verify but more costly to flood.

## 6.8 DISTRIBUTED STORAGE

The storage model combines content-addressed immutable objects with distributed indexes for discovery. Peers are not forced to retain arbitrary third-party data: ordinary content is stored and served by nodes that retrieved it and chose to keep it, while critical DHT metadata is actively replicated (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016). Content may be advertised through ephemeral virtual nodes, separating provider availability from a long-term physical identity (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).

### 6.8.1 Data Distribution Strategy

Every object is an opaque byte sequence identified by its exact hash. The DDT stores descriptors and expiring holder advertisements rather than the object bytes:

$$H(\text{content}) \rightarrow \{\text{title}, \text{type}, \text{tags}[], \text{holders} : \{(H(pk_{vn}^{eph_1}), \text{exp}_1), \dots\}\}$$

A publisher stores the object locally and may use a dedicated ephemeral virtual node to advertise and serve it. Placement is opportunistic: DHT proximity determines who replicates the index record, not who must retain the content. Transfers support byte ranges, and private objects may retain an application-level encryption layer so that holders can serve bytes without reading their plaintext (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016), (BOS et al., 2017), (National Institute of Standards and Technology, 2024a).

### 6.8.2 Replication Mechanisms

Table 7 – Storage replication policies.

| Data            | Replication                                                                                               | Availability consequence                                                                             |
|-----------------|-----------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| Content objects | Consumer-driven: downloaders that retain an object may advertise it through a new ephemeral virtual node. | Popular objects naturally gain holders; unpopular objects may disappear when the last holder leaves. |
| DHT metadata    | Actively maintained by the $K$ responsible physical nodes.                                                | Discovery and routing records are re-replicated as responsibility changes.                           |

Replicas of the same content hash are equivalent; there is no primary copy. The MVP trusts holder advertisements and verifies the downloaded bytes by hash. Proofs of Retrievability remain a possible extension (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).

### 6.8.3 Data Consistency

Content is immutable: an update creates new bytes, a new hash, and a new DDT entry. Consequently, consistency is established by hash validation rather than mutable replica coordination. Applications represent evolution through signed pointers, manifests, or causal links to new objects. Holder metadata may be temporarily stale; requesters try alternative providers until an advertisement expires or all candidates fail (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).

### 6.8.4 Data Lifecycle and Expiration

Object bytes have no protocol-level expiration and remain available while at least one peer retains them. DDT advertisements and other soft state do expire and must be renewed, preventing obsolete providers from remaining indefinitely. Each peer controls its local retention policy, so removing one copy is not global deletion and the network cannot guarantee erasure after dissemination. The model therefore offers decentralized persistence, not absolute permanence (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).

### 6.8.5 Content Discovery and Resolution Model

Content discovery separates semantic search, mutable identity, and availability:

## Content Discovery and Resolution Model (Two-Stage)

A lightweight process connecting semantic discovery to current content availability using two distributed tables.

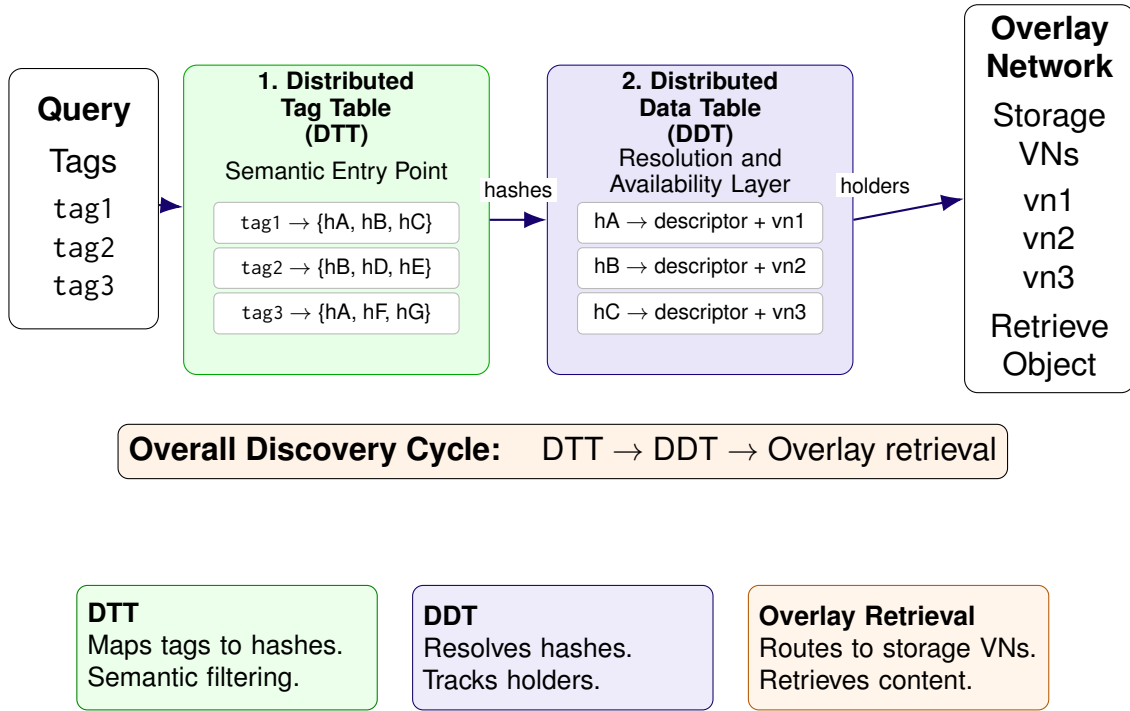


Figure 11 – Content discovery and resolution model.

The DTT maps normalized tags to resource identifiers. An immutable identifier can be resolved directly in the DDT; a mutable identifier first resolves a signed DPT record whose `target_ref` identifies the latest immutable state. The DDT then returns the virtual holders of that hash, and the DRT provides routes to a selected holder. The two resolution paths are therefore:

$$DTT \rightarrow DDT \quad \text{or} \quad DTT \rightarrow DPT \rightarrow DDT$$

This separation keeps content immutable while supporting semantic discovery and mutable application state (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).

### 6.9 RESILIENCE AND FAULT TOLERANCE

Failure and churn are normal operating conditions. The architecture protects critical discovery metadata more strongly than ordinary content: DHT records are replicated and repaired, routes and sessions have alternatives, while content availability

still depends on voluntary holders (TOSIC; VICIC, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).

### 6.9.1 Node Churn Handling

A physical-node failure also makes its hosted virtual nodes unavailable, but their distributed references disappear only when advertisements expire. Timeouts, failed keepalives, broken paths, and missing renewals indicate unavailability without proving permanent departure. This ambiguity limits direct correlation between physical-node activity and virtual-node visibility. Sessions may move to another published route if one becomes available before their liveness window expires (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).

### 6.9.2 Redundancy Strategies

Table 8 – Resilience mechanisms by network state.

| State                 | Resilience mechanism                                                                                       |
|-----------------------|------------------------------------------------------------------------------------------------------------|
| DHT metadata          | Store each record at the $K$ closest physical nodes and re-replicate when the responsible set changes.     |
| Virtual reachability  | Publish multiple independently constructed DRT routes and select among available entry points.             |
| Physical connectivity | Keep several validated peers and support direct TCP, UDP, or relay-assisted endpoints.                     |
| Content               | Allow each downloader that retains valid bytes to become another holder; durability remains demand-driven. |

Redundancy is intentionally local and bounded: it improves availability without requiring global consensus or unlimited replication (TOSIC; VICIC, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).

### 6.9.3 Failure Recovery

Recovery uses existing protocol state rather than a centralized coordinator. Failed physical sessions are re-established through another advertised transport; failed virtual delivery retries another DRT route; stale DHT replicas are replaced when responsibility is recomputed; and content lookup tries another holder. Persistent identities and local state allow a returning peer to resume participation, while expired advertisements must be republished (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016).

#### 6.9.4 Network Rebalancing

Physical nodes periodically recompute the  $K$  peers closest to each DHT key. A node that enters the responsible set receives a validated replica; one that leaves it may drop the record after successful handoff. Large aggregate records may be divided into independently verifiable shards and later compacted as load decreases. Rebalancing is incremental, preserving service while ownership follows the evolving keyspace (STOICA et al., 2001), (MAYMOUNKOV; MAZIERES, 2002), (ROWSTRON; DRUSCHEL, 2001), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022).

### 6.10 OPERATIONAL WORKFLOWS

The following workflows summarize how peers join, disappear, resolve content, and exchange messages through the layered architecture (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).

#### 6.10.1 Node Join Procedure

1. Generate or restore the physical and virtual signing identities.
2. Contact hard-coded bootstrap endpoints or DNS seeds and obtain initial physical-node information.
3. Validate peers and determine which local transports can be advertised; private nodes establish and publish relay reachability.
4. Build a useful local peer view, then create and publish DRT routes for each local virtual node.
5. Gradually assume DHT responsibilities according to XOR proximity.

A stateful rejoin reuses identities and cached knowledge but renews expired presence. A node may remain physical-only, although overlay communication requires at least one virtual identity with published routes (LI; WANG; WANG, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018).

#### 6.10.2 Node Departure Procedure

Departure requires no global leave transaction. Once a peer stops responding, physical sessions fail, its virtual routes become unusable, and unrenewed advertisements expire. Replicated DHT metadata is repaired as responsibility changes; content survives only if another holder remains. A returning node may reuse its persistent identity

and republish its soft state. This passive model tolerates abrupt failures and avoids precise global tracking of online intervals (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).

### 6.10.3 Data Lookup Process

A requester follows one of two initial paths: a known hash queries the DDT directly, while semantic lookup uses the DTT and, for mutable resources, resolves the signed DPT `target_ref`. It then:

1. obtains the final content hash and queries the DDT;
2. selects an unexpired virtual holder;
3. resolves that holder's entry points through the DRT;
4. establishes a virtual session and requests the full object or byte ranges;
5. tries another holder or route after failure;
6. verifies the assembled bytes against the expected hash; and
7. optionally retains the object and publishes itself as another holder.

Private content follows the same lookup but remains application-encrypted. Cached DTT, DPT, DDT, and DRT results may reduce repeated network queries (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (QIAN et al., 2016), (BOS et al., 2017), (National Institute of Standards and Technology, 2024a).

### 6.10.4 Message Routing Process

Application data is carried inside a protected virtual envelope, which is itself forwarded through physical route-data envelopes:

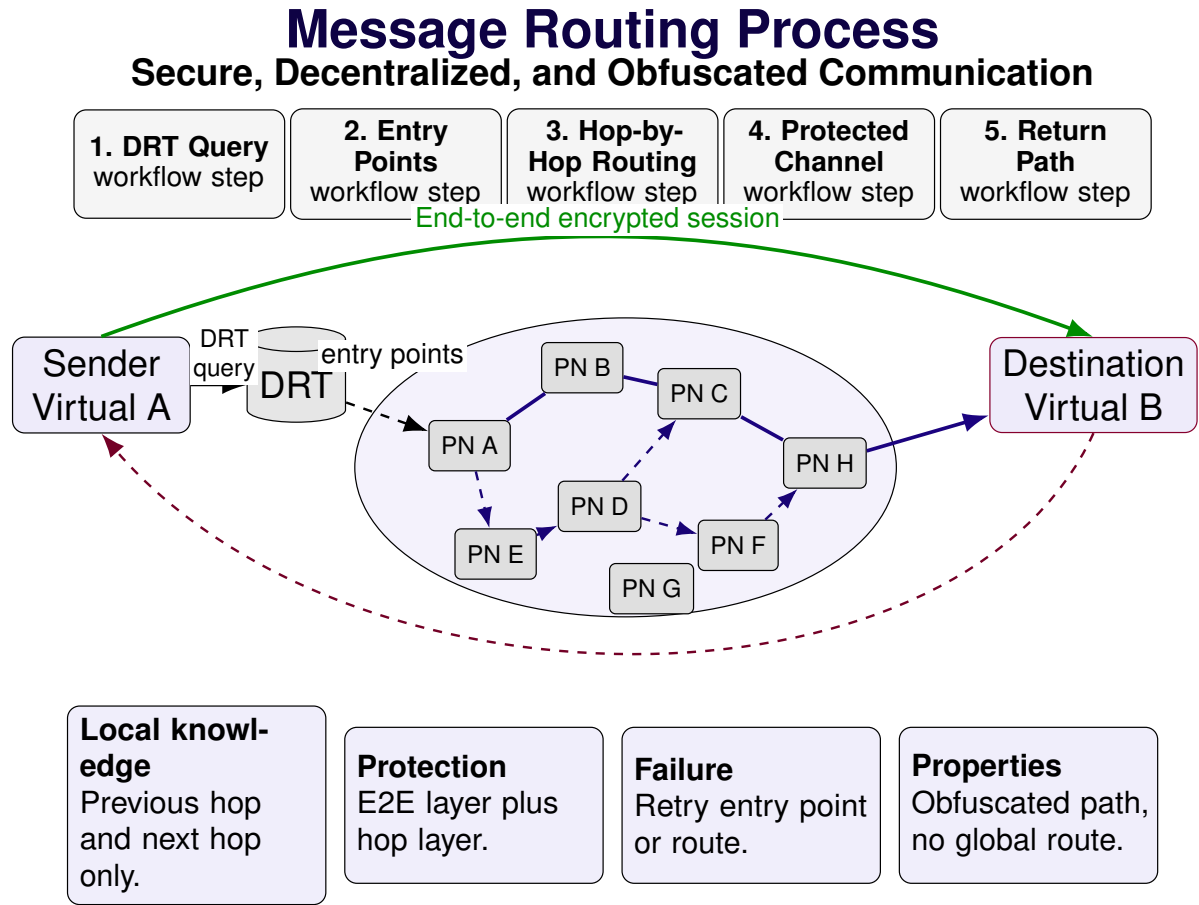


Figure 12 – Message routing process.

The sender queries the DRT for the destination virtual-node identifier, chooses an entry point, and forwards by `path_id`. Each physical hop resolves only its local previous/next mapping; it does not receive the complete path. Hop sessions protect physical links, while the independent virtual session protects the end-to-end payload. Authenticated associated data binds public routing metadata that must not be modified (LI; WANG; WANG, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (BOS et al., 2017), (National Institute of Standards and Technology, 2024a).

Reliable session payloads add identifiers, sequence numbers, acknowledgments, duplicate suppression, ordered delivery, and bounded retransmission. A broken path causes selection of another published route rather than reversal or continuation of the partial path. Replies independently resolve a route toward the original sender, reducing path symmetry and preserving the architecture's anonymity-oriented routing properties (TOSIC; VICIC, 2022), (AUGUSTINE; CHATTERJEE; PANDURANGAN, 2022), (MIRZAEI; FARAHANI; YAZDANI, 2018), (QIAN et al., 2016).

## 7 IMPLEMENTATION

This chapter presents the *AnonNetCore Python MVP*, an experimental implementation that validates physical/virtual identity separation, DHT-based discovery, route construction, end-to-end virtual sessions, content exchange, and external application integration. The source is available at <[https://github.com/andersonsimioni/AnonNetCore\\_py](https://github.com/andersonsimioni/AnonNetCore_py)>, with an institutional mirror at <[https://codigos.ufsc.br/andersonsimioni/anonnnetcore\\_py](https://codigos.ufsc.br/andersonsimioni/anonnnetcore_py)>. The prototype targets functional integration and observability in controlled environments; it is not presented as a production-ready or Internet-scale network.

### 7.1 IMPLEMENTATION METHODOLOGY

The work followed an applied, exploratory, and incremental methodology: physical communication and discovery were implemented first, followed by sessions, distributed tables, routes, virtual protocols, content transfer, the local API, and the social PoC. Each stage was validated through observable traces and integrated smoke tests before the next layer was added.

The implementation was developed specifically for this work rather than adapted from an existing peer-to-peer core. The author designed and implemented the physical and virtual node models, transport-adapter integration, bootstrap and peer-validation flows, distributed-table protocols, route construction and execution, physical and virtual sessions, relay forwarding, reliability mechanisms, local API, observability tools, integration smokes, and social PoC. External software was used only as supporting infrastructure: Python and its standard library provide the runtime, SQLAlchemy and SQLite provide persistence, cryptography provides authenticated symmetric encryption, OpenSSL exposes the selected post-quantum primitives, websockets provides the local event channel, and Docker supplies reproducible test clusters. These libraries implement general-purpose primitives; the network architecture, protocol composition, record models, route lifecycle, and application integration are contributions of this project.

Validation combines real core instances, Docker clusters, DHT publication/query, route and session flows, content transfer, API/WebSocket access, and debug-state inspection. The external HTML/JavaScript PoC consumes only the public local API, demonstrating application integration without importing internal modules. Logs, a debug endpoint, WebSocket events, and the Debug Console provide runtime observability.

The controlled MVP uses Python, SQLite, TCP, experimental fragmented UDP, relay-assisted forwarding, SHA-512 identifiers, ML-KEM/ML-DSA, AES-256-GCM-SIV

with AAD, DHT publication proof-of-work, and lightweight reliable-session delivery. Complete NAT traversal, QUIC, full Kademlia routing tables, mature anti-abuse mechanisms, and formal security auditing remain future work; earlier planned choices such as Kyber-1024 and SHA3-512 were superseded by the implemented suite.

## 7.2 IMPLEMENTATION SCOPE

The validated scope covers physical nodes over TCP, experimental UDP, and relay transport; bootstrap and physical information validation; authenticated physical sessions with keepalive and reliable acknowledgments; replicated multi-namespace DHT records with XOR responsibility and publication proof-of-work; automatic DRT route publication; reliable end-to-end virtual sessions and messages; byte-range content transfer; local HTTP/WebSocket access; and the social PoC with profiles, posts, feeds, and direct messages.

The implementation does not claim complete protection against coordinated malicious peers, large-scale Sybil attacks, global traffic correlation, or Internet-scale churn. These aspects require additional hardening and broader experimentation.

## 7.3 TECHNOLOGY STACK

The MVP was implemented in Python to prioritize development speed, testability, and observability. The final architecture, however, remains independent of the language and can be reimplemented in a lower-level language such as C++ or Rust in future versions.

Table 9 summarizes the main implementation choices.

Table 9 – Main technologies used in the experimental MVP.

| Component              | Implementation choice                                                                                                                     |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| Transport              | TCP sockets with length-prefixed frames, experimental UDP datagrams with compact binary fragmentation, and relay-assisted TCP forwarding. |
| Protocol envelope      | JSON-based protocol envelope for physical and virtual messages.                                                                           |
| Local persistence      | SQLite through SQLAlchemy.                                                                                                                |
| Content storage        | Filesystem storage, with metadata persisted in the local database.                                                                        |
| Identity identifier    | SHA-512(public_key) for both physical and virtual nodes.                                                                                  |
| Post-quantum KEM       | ML-KEM-768 when available through the cryptographic backend.                                                                              |
| Post-quantum signature | ML-DSA-65 when available through the cryptographic backend.                                                                               |
| Symmetric encryption   | AES-256-GCM-SIV for authenticated encrypted session payloads with AAD for selected public metadata.                                       |
| Local API              | HTTP API on 127.0.0.1:18080 and WebSocket events on 127.0.0.1:18081/v1/events.                                                            |
| Testing environment    | Local processes and Docker-based clusters.                                                                                                |
| Application PoC        | Local HTML/JavaScript social interface using the local API.                                                                               |

## 7.4 IMPLEMENTED ARCHITECTURE

The implemented system follows the layered model proposed in this work. The physical layer represents real processes and network connections. The virtual layer represents logical identities hosted by physical nodes. Distributed tables provide discovery, routing, and content location. A local API exposes the core functionality to external applications. Finally, the social PoC validates that an application can use the network layer without depending on internal modules.

Figure 13, adapted from the execution flow used in the project presentation, connects these layers from the perspective of an application request. A virtual node first resolves distributed physical and route information, the route protocol and active strategy establish hop-by-hop forwarding state, and a virtual session then carries protected application messages to the destination virtual node.

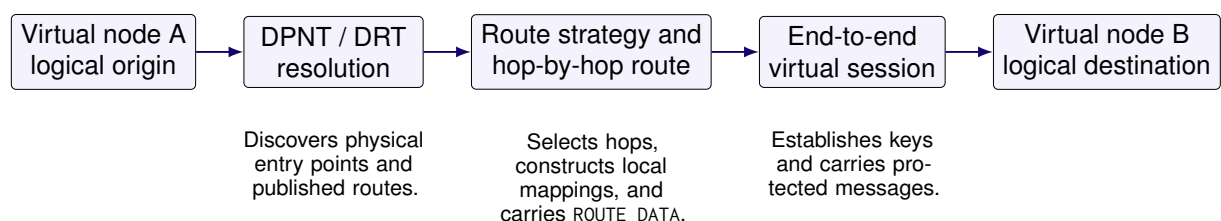


Figure 13 – From distributed discovery to virtual application communication.

Figure 14 shows the implemented architecture.

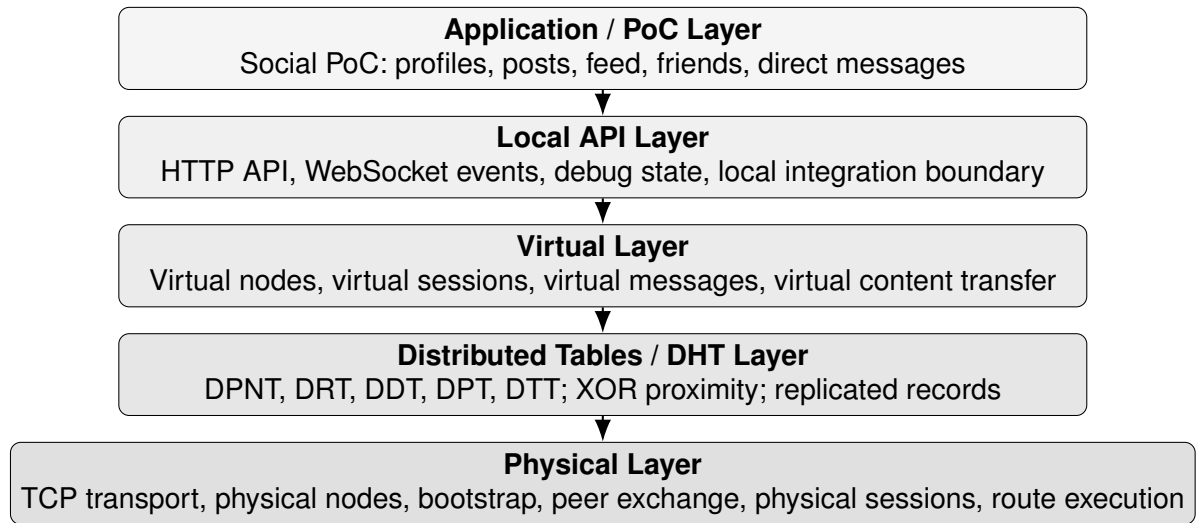


Figure 14 – Implemented layered architecture of the AnonNetCore Python MVP.

The layers are not independent applications. They are internal abstractions inside the core. The social PoC is the only external component and communicates with the core through the local API. This separation is important because it demonstrates that the network layer can support decentralized applications without exposing internal protocol modules directly.

#### 7.4.1 Physical Layer

The physical layer is responsible for communication between real processes. Each running core instance represents one physical node. A physical node opens a TCP listener, receives frames, decodes protocol envelopes, processes physical message types, maintains peer information, and participates in DHT and routing workflows.

Each physical node is identified by the hash of its public key:

$$physical\_node\_id = SHA512(pk_{physical}) \quad (7.1)$$

The physical layer implements the following responsibilities:

- TCP listener initialization and frame parsing;
- experimental UDP listener initialization, datagram fragmentation, and reassembly;
- relay-assisted physical forwarding through selected public relay-capable nodes;
- direct peer communication through physical envelopes;
- bootstrap from known endpoints;

- exchange of physical node descriptors;
- validation of reachable peers;
- physical session establishment and keepalive;
- DHT publish/query forwarding;
- route construction and route execution.

The MVP no longer uses TCP as the only transport. TCP remains the primary validated transport, but the implementation also includes an experimental UDP transport and a relay-assisted TCP transport. UDP hole punching, STUN/TURN integration, QUIC, and mature Internet-scale NAT traversal remain future implementation targets.

Encrypted physical payloads use AES-256-GCM-SIV with a 96-bit nonce and a 128-bit authentication tag. The implementation generates a fresh nonce for each encryption operation and rejects nonce reuse for the same key within the running process. Associated authenticated data (AAD) binds the encrypted payload to the public physical envelope header, excluding only the `payload_encrypted` marker. This prevents visible routing and session metadata that must remain outside the ciphertext from being altered without detection.

#### 7.4.2 Relay-Assisted Physical Transport

The implemented relay mechanism is modeled as a physical transport adapter named `relay_tcp`. It does not replace the physical session protocol and does not create an application-level shortcut. Instead, it encapsulates an ordinary physical packet inside a relay envelope and transports it through a public physical node that already has active physical sessions with the participants.

In the current MVP, a node can act as a physical relay only when it is public, TCP-enabled, and configured with relay service enabled. Private physical nodes do not open public TCP or UDP listeners for other cores. They still initialize the available transport adapters locally, but they only advertise a physical endpoint after selecting a public relay. This selection is performed by the `Physical_Relay_Maintenance_Runtime`, which periodically searches for relay-capable peers, opens or refreshes a physical session with the selected relay, and publishes a local endpoint of the following form:

```
{  
  "transport": "relay_tcp",  
  "host": relay_host,  
  "port": relay_port,  
}
```

```

    "metadata": {
        "relay_physical_node_id": relay_id,
        "target_physical_node_id": private_node_id
    }
}

```

This endpoint is then included in physical-node information exchange and in DPNT records. Therefore, other peers do not need to know a direct address for the private node. They only need to know which relay endpoint can be used to reach it.

When a peer sends data to a relay endpoint, the local transport layer builds a `PHYSICAL_RELAY_DATA` message containing the target physical node identifier and the encapsulated packet bytes. Before sending this relay message, the sender establishes or reuses a normal physical session with the relay. The relay accepts the message only if the sender's physical session is active. It then checks whether it has an active physical session with the target physical node. If such a session exists, the relay forwards another `PHYSICAL_RELAY_DATA` message through that target session. When the target node receives a relay message addressed to its own physical identifier, the payload is injected into the local `relay_tcp` adapter and processed as an ordinary inbound physical packet.

The resulting path can be summarized as:

$$A \leftrightarrow R \leftrightarrow B \quad (7.2)$$

where  $A$  is the sender,  $R$  is the relay-capable public physical node, and  $B$  is the private target. Both  $A \leftrightarrow R$  and  $B \leftrightarrow R$  are ordinary authenticated physical sessions. The relay forwards encrypted physical packets and is not treated as the final application endpoint. Higher-level payloads remain layered normally, so a virtual message may still be represented as a virtual envelope inside a physical route packet inside a relay packet:

$$\text{relay\_packet}(\text{physical\_packet}(\text{virtual\_envelope})) \quad (7.3)$$

This design keeps relay behavior inside the physical transport layer and preserves the same protocol flow used by direct communication. It also makes the relay mechanism observable through the same peer discovery, session, DPNT, and route-building logic used by the rest of the core.

### 7.4.3 Virtual Layer

The virtual layer represents logical identities hosted by physical nodes. A single physical node may host multiple virtual nodes. This design separates the network

addressable process from the application-level identity.

Each virtual node is identified by the hash of its public key:

$$virtual\_node\_id = SHA512(pk_{virtual}) \quad (7.4)$$

In the social PoC, one virtual node corresponds to one social profile.

Virtual nodes are responsible for publishing route entries, establishing virtual sessions, sending virtual messages, signing DHT records, and advertising content. Virtual sessions are end-to-end sessions between virtual identities and are transported through physical routes using `ROUTE_DATA`.

#### 7.4.4 DHT and Distributed Tables

The implementation uses a generic local table named `dht_record`. Each distributed record has a logical namespace, a logical key, a physical DHT key, and a canonical JSON payload. The DHT key is computed as:

$$key = SHA512(namespace \parallel " | " \parallel logical\_key) \quad (7.5)$$

Responsibility is based on XOR distance between the physical node identifier and the DHT key:

$$distance = XOR(physical\_node\_id, dht\_key) \quad (7.6)$$

The  $K$  physical nodes with the lowest distances are considered responsible for the key. In the current MVP, the default replication factor is:

$$K = 3 \quad (7.7)$$

Before a DHT publication is accepted, the publisher must include lightweight proof-of-work nonces. The current implementation does not treat every DHT record as a single opaque blob. Instead, proof-of-work is attached to the semantic payloads that may be independently published, merged, or replicated. This is important because some namespaces aggregate lists of entries or holders, and peers may receive only a fragment of a larger logical record.

The payload types that currently carry a `pow_nonce` are:

- `Dpnt_Record_Payload`;
- `Drt_Route_Entry_Record`;

- Ddt\_Holder\_Record;
- Dtt\_Entry\_Record;
- Dpt\_Record\_Payload.

For example, a DRT record may contain multiple route entries, but the proof is validated per `Drt_Route_Entry_Record`. Similarly, DDT and DTT records validate individual holders or entries. The canonical proof material is built from the namespace, the physical DHT key, the canonical payload without its nonce, and any required parent identity context. The implemented proof can be summarized as:

$$\text{proof} = \text{SHA512}(\text{canonical\_payload\_material} \parallel \text{nonce}) \quad (7.8)$$

The publication is accepted only when the proof hash satisfies the configured number of leading zero bits:

$$\text{leading\_zero\_bits}(\text{proof}) \geq \text{network\_pow\_difficulty\_bits} \quad (7.9)$$

This mechanism does not provide a complete anti-abuse system, but it introduces a measurable computational cost before records can be inserted or refreshed in the distributed tables. Responsible nodes can verify the proof cheaply before storing or merging the payload, while publishers must search for a valid nonce before starting publication.

Table 10 summarizes the implemented namespaces.

Table 10 – Implemented DHT namespaces.

| Namespace | Name                             | Purpose                                                                                                                                         |
|-----------|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| dpnt      | Distributed Physical Nodes Table | Stores validated physical node descriptors, including public key, endpoints, transport capabilities, status, and signature.                     |
| drt       | Distributed Route Table          | Stores route entries that map a virtual node to one or more physical entry points capable of delivering traffic to that virtual node.           |
| ddt       | Distributed Data Table           | Stores content holder announcements, mapping a content hash to virtual nodes that currently possess the content.                                |
| dpt       | Distributed Pointer Table        | Stores mutable signed pointers owned by virtual nodes. In the social PoC, it points to the most recent profile state object.                    |
| dtb       | Distributed Tag Table            | Provides a model for distributed tag-to-resource association. The parser and merge logic exist, but it is not part of the main social PoC flow. |

The implemented DHT supports publication, query, merge, deduplication, and maintenance routines. When a record with an existing key is received, the system attempts to merge it according to the namespace semantics. For instance, DRT records aggregate route entries, DDT records aggregate holders, and DPT records prefer the most recent valid pointer signed by the owning virtual node.

## 7.5 BOOTSTRAP AND PEER DISCOVERY

The bootstrap mechanism provides the first known endpoints of the network. In the current MVP, bootstrap endpoints are configured through `CoreConfig.bootstrap_public_endpoints`. The default development setup uses two bootstrap ports:

19001, 19002 (7.10)

The bootstrap host is selected from environment variables or from local network detection. The implemented flow is:

1. the core initializes its local physical identity;
2. the TCP listener is started;
3. bootstrap endpoints are loaded;
4. endpoints that point to the local node itself are ignored;

5. the node sends `PHYSICAL_NODE_INFO_REQUEST`;
6. received peers are persisted locally;
7. validation runtimes test reachability;
8. peer exchange spreads known physical nodes;
9. validated physical nodes are published in the DPNT.

Figure 15 summarizes the main implemented flows, including bootstrap, route creation, virtual session establishment, and DHT publish/query.

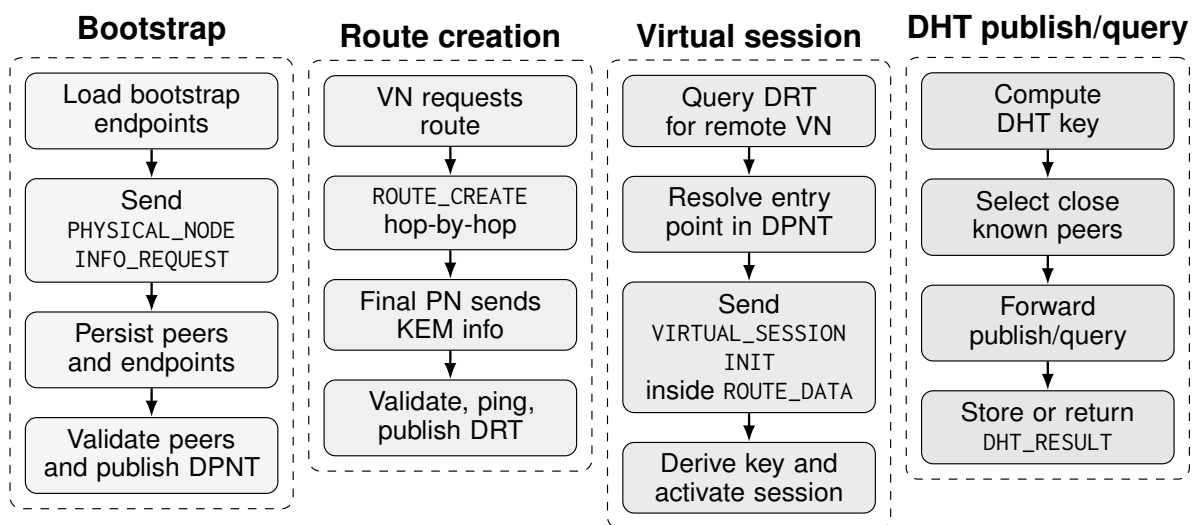


Figure 15 – Main implemented flows: bootstrap, route creation, virtual session establishment, and DHT publish/query.

## 7.6 ROUTE CONSTRUCTION AND ROUTE EXECUTION

Routes allow virtual nodes to be reached through physical nodes without exposing the full route to every participant. The route construction process is performed by physical nodes, although it is requested on behalf of a virtual node.

### 7.6.1 Route Strategy Abstraction

The implementation follows the strategy abstraction defined in Section 6.5.4: the protocol controls construction, validation, DRT publication, failure propagation, and later execution, while the strategy selects physical hops and defines its termination and optimization rules. Consequently, new selection policies do not require changes to the route message lifecycle.

After publication, execution is strategy-independent. Every hop processes `ROUTE_DATA` by resolving its local `path_id` mapping; it does not rerun the construction policy or

require the complete route. Figure 16 summarizes this separation, while Section 7.6.2 presents the implemented message sequence once, together with the execution flow.

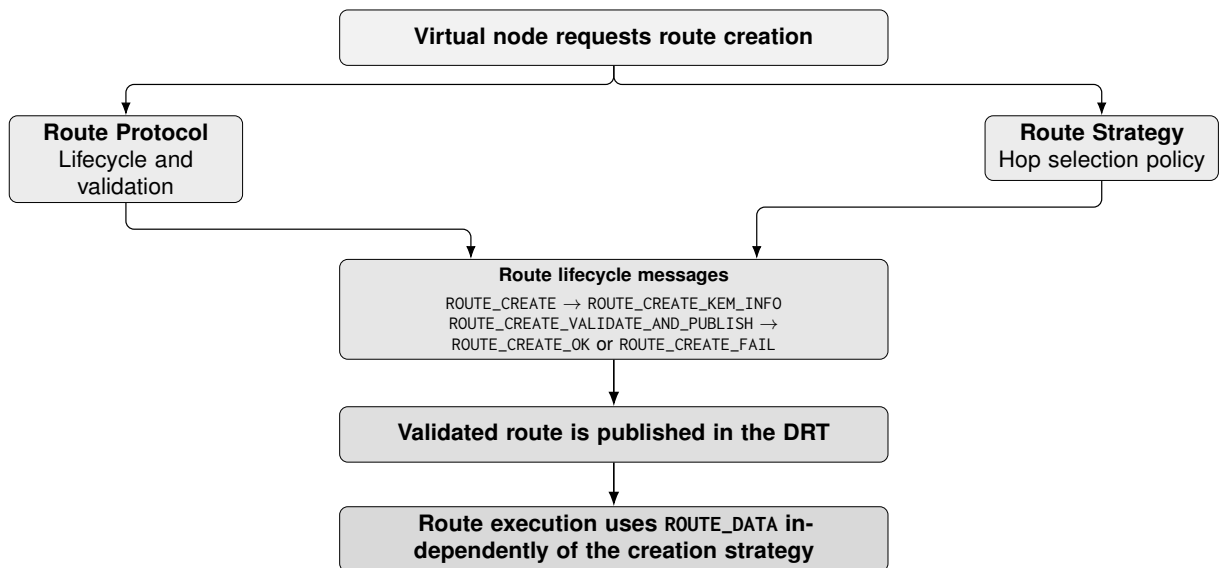


Figure 16 – Separation between the route protocol lifecycle and the route strategy used to select physical hops.

The current MVP primarily implements the `random_walk_ttl` strategy. In this strategy, the virtual node starts route creation with an expected temporal budget, expressed in milliseconds. Each intermediate physical node forwards the `ROUTE_CREATE` message to another physical node while the route remains within the expected construction constraints. The TTL value is therefore interpreted as a strategy-specific construction budget, not as a fixed rule of the routing layer. When the final physical node is reached, the route is validated through `ROUTE_CREATE_PING` and `ROUTE_CREATE_PONG`. The observed round-trip time is compared against the expected budget, including an acceptable error window. If the measured RTT is valid, the final physical node signs the route information and publishes the route entry in the DRT. If validation fails, `ROUTE_CREATE_FAIL` is propagated back through the route so intermediate hops can discard pending route state instead of waiting for timeout expiration.

This strategy is useful for the MVP because it is simple, dynamic, and capable of creating relatively unpredictable paths without requiring global knowledge of the network. It also fits the experimental nature of the prototype, since it can be evaluated through local clusters, route logs, DRT records, and smoke tests. However, the same route lifecycle can be reused by future strategies that terminate by hop count, bandwidth threshold, reliability score, or onion-like path definition.

The MVP implements only `random_walk_ttl`; the additional policies and cost dimensions are already summarized in Table 5. They remain architectural extension points rather than implemented alternatives. This distinction keeps the common route lifecycle

reusable without presenting future strategies as experimental results.

### 7.6.2 Hop-by-Hop Route Construction and Execution

The MVP implements the route strategy `random_walk_ttl`. Hop-by-hop construction means that the initiator does not install a complete globally visible path. Instead, each physical hop receives a route creation request, applies the active strategy to decide whether and where to forward it, creates a local `path_id` mapping, and forwards only the information required by the next hop. The strategy is responsible for selecting the next hop and deciding when the route should stop; the execution layer is responsible only for forwarding later packets through the local mappings created during construction.

The implemented route build message sequence is:

```
ROUTE_CREATE
→ ROUTE_CREATE_KEM_INFO
→ ROUTE_CREATE_VALIDATE_AND_PUBLISH
→ ROUTE_CREATE_PING
→ ROUTE_CREATE_PONG
→ ROUTE_CREATE_OK or ROUTE_CREATE_FAIL
```

(7.11)

During route creation, each intermediate physical node stores only the local mapping required to forward packets. For example, an intermediate hop stores a mapping from a received `path_id` to a generated `path_id`, together with the next physical forwarding action. It does not need to know the full path or the final virtual identity. This design preserves the separation between physical routing and virtual semantics. The TTL used by the implemented strategy limits construction and validation for `random_walk_ttl`, but packet execution after publication depends on local route state rather than on recomputing the random walk. If the final validation fails, `ROUTE_CREATE_FAIL` follows the same reverse notification role as `ROUTE_CREATE_OK`, but causes pending mappings to be rejected instead of activated.

After a route is created, virtual envelopes are transported inside physical `ROUTE_DATA` messages. Conceptually, the physical envelope has the following structure:

```
physical_envelope {
  message_type: ROUTE_DATA,
  payload: {
    path_id,
    direction,
    virtual_session_id,
```

```
        virtual_envelope_ciphersed: bool,  
        virtual_envelope  
    }  
}
```

The `virtual_envelope_ciphersed` field is a boolean flag indicating whether the serialized virtual envelope carried in `virtual_envelope` is encrypted. It is not a second copy of the payload. When the flag is true, the field contains an encrypted serialized virtual envelope; when the flag is false, the field carries a plaintext control envelope allowed by the current protocol stage. This distinction is useful during session establishment and debugging, while ordinary post-session application traffic is expected to use the encrypted form.

The route execution handler resolves the local `path_id`, decides whether the packet must be forwarded or delivered locally, and re-encapsulates virtual responses when necessary. It does not interpret application-level semantics. Its role is only to move virtual envelopes through physical routes.

## 7.7 VIRTUAL SESSIONS AND VIRTUAL MESSAGES

Virtual sessions are end-to-end sessions between virtual nodes. They are established over routes previously published in the DRT. The initiator first resolves the remote virtual node in the DRT, obtains a physical entry point, resolves that physical node in the DPNT, and then sends a virtual session initialization envelope through `ROUTE_DATA`.

Figure 17 summarizes when the implemented cryptographic mechanisms are applied. Every adjacent pair of physical peers establishes an independent hop-by-hop session key, while the communicating virtual nodes establish a separate end-to-end session key. Signed ephemeral ML-KEM material is used during each establishment flow, ML-DSA authenticates the exchanged identity and key material, and AES-256-GCM-SIV subsequently protects the corresponding physical or virtual session payloads.

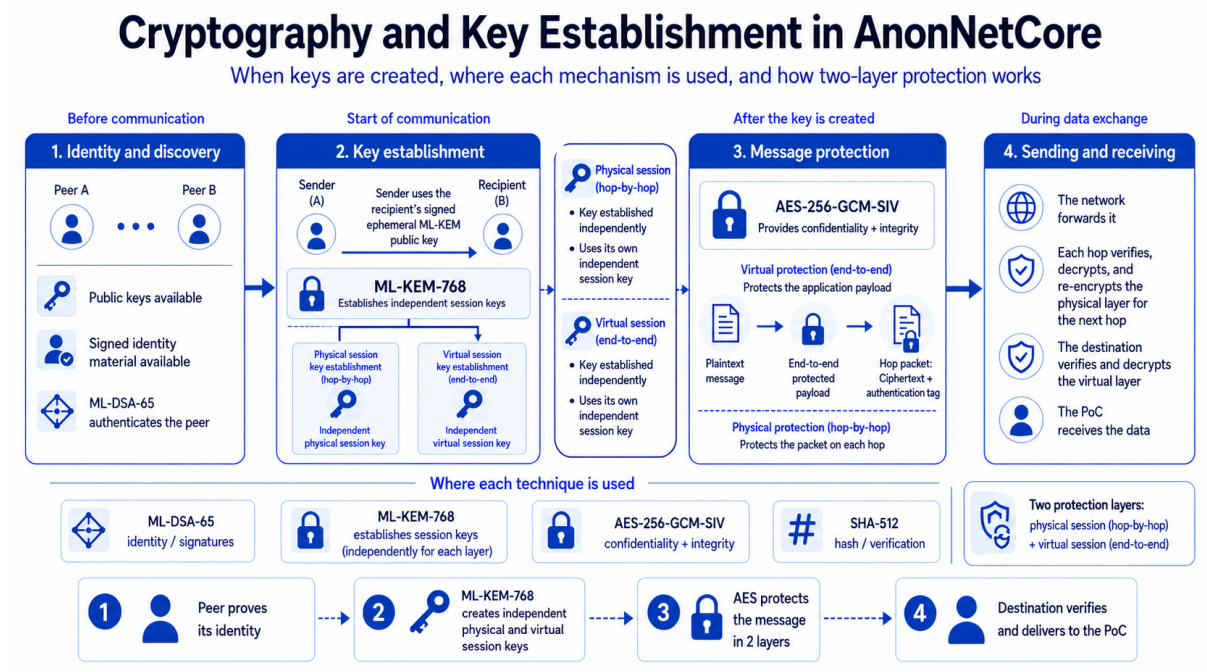


Figure 17 – Cryptographic identity, independent physical and virtual key establishment, and two-layer message protection in the implemented Core.

The implemented virtual session message types are:

- VIRTUAL\_SESSION\_INIT;
- VIRTUAL\_SESSION\_INIT\_OK;
- VIRTUAL\_SESSION\_KEY\_CONFIRM;
- VIRTUAL\_SESSION\_READY;
- VIRTUAL\_SESSION\_KEEPLIVE;
- VIRTUAL\_SESSION\_KEEPLIVE\_ACK;
- VIRTUAL\_SESSION\_CLOSE.

After the session is established, application messages are sent through:

VIRTUAL\_SESSION\_DATA (7.12)

In the social PoC, direct messages use the application message type:

social.direct\_message (7.13)

## 7.8 RELIABLE SESSION PAYLOADS

The current MVP implements a lightweight reliability layer on top of active physical and virtual sessions. This mechanism is not a replacement for TCP reliability and does not attempt to solve all route-level failures. Its purpose is narrower: to provide ordered delivery, duplicate suppression, acknowledgments, and bounded retransmission for protected session payloads in a way that can also support less reliable transports in future iterations.

Reliable transmission is implemented with two envelope families:

- `PHYSICAL_SESSION_RELIABLE_DATA` and `PHYSICAL_SESSION_RELIABLE_ACK` for hop-by-hop physical sessions;
- `VIRTUAL_SESSION_RELIABLE_DATA` and `VIRTUAL_SESSION_RELIABLE_ACK` for end-to-end virtual sessions.

Each reliable payload wraps an inner protocol message:

```
{
  "reliable_message_id": "...",
  "sequence_number": 1,
  "inner_message_type": "...",
  "inner_payload": { ... },
  "created_at": "...",
  "attempt": 1
}
```

The session manager maintains in-memory reliability state per session, including the next outbound sequence number, the next expected inbound sequence number, pending outbound messages, buffered out-of-order inbound messages, and already delivered message identifiers. When a reliable payload arrives, the receiver immediately prepares an acknowledgment containing the received message identifier and sequence number. Duplicate messages are acknowledged again but not delivered twice. Out-of-order messages are buffered until the missing previous sequence numbers arrive, preserving ordered delivery to the inner protocol handler.

Retransmission is handled by `Session_Runtime`. The runtime periodically scans pending reliable messages and resends messages whose retry deadline has expired. Physical sessions use a fixed retry interval because they represent adjacent hop-by-hop communication. Virtual sessions use a dynamic retry interval derived from the route

RTT when available, bounded by minimum and maximum configuration values. This distinction is necessary because an end-to-end virtual session may traverse a longer route than a direct physical session.

For virtual reliable payloads, an acknowledgment may also carry synchronous inner virtual responses produced while processing the delivered message. This keeps the protocol flow compact without bypassing the normal virtual-session processing path: the response is still represented as a virtual envelope and is dispatched by the virtual session handler after the reliable acknowledgment is processed.

The reliable state is runtime-only. It is not persisted as durable storage, because it represents live session delivery state and depends on active session keys, active routes, and currently reachable peers. If a session is lost, higher layers must establish a new session or retry the application-level operation.

## 7.9 PROTOCOL MESSAGE TYPES

Table 11 summarizes the main implemented protocol families and message types.

Table 11 – Implemented protocol families and message types.

| Layer    | Protocol family    | Message types                                                                                              | Purpose                                                               |
|----------|--------------------|------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|
| Physical | Ping               | PING<br>PONG                                                                                               | Validates basic reachability and measures RTT between physical nodes. |
| Physical | Physical node info | PHYSICAL_NODE_INFO_REQUEST<br>PHYSICAL_NODE_INFO_RESPONSE                                                  | Obtains identity and advertised endpoints of physical nodes.          |
| Physical | Peer exchange      | PHYSICAL_NODE_INFO_EXCHANGE_REQUEST<br>PHYSICAL_NODE_INFO_EXCHANGE_RESPONSE<br>PHYSICAL_NODE_INFO_ANNOUNCE | Spreads known physical nodes beyond the initial bootstrap list.       |

| Layer    | Protocol family           | Message types                                                                                                                                                                                       | Purpose                                                                                                                                      |
|----------|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| Physical | Physical session          | PHYSICAL_SESSION_INIT<br>PHYSICAL_SESSION_INIT_OK<br>PHYSICAL_SESSION_KEY_CONFIRM<br>PHYSICAL_SESSION_READY<br>PHYSICAL_SESSION_KEEPAIVE<br>PHYSICAL_SESSION_KEEPAIVE_ACK<br>PHYSICAL_SESSION_CLOSE | Creates and maintains secure hop-by-hop sessions between adjacent physical nodes.                                                            |
| Physical | Physical reliable session | PHYSICAL_SESSION_RELIABLE_DATA<br>PHYSICAL_SESSION_RELIABLE_ACK                                                                                                                                     | Wraps physical-session payloads with sequence numbers, acknowledgments, duplicate suppression, ordered delivery, and bounded retransmission. |
| Physical | Physical relay            | PHYSICAL_RELAY_DATA                                                                                                                                                                                 | Encapsulates a physical packet through a relay-capable public physical node using active physical sessions with the sender and target.       |
| Physical | DHT                       | DHT_PUBLISH<br>DHT_QUERY<br>DHT_RESULT                                                                                                                                                              | Publishes and retrieves distributed records across DPNT, DRT, DDT, DPT, and DTT namespaces.                                                  |
| Physical | Route build               | ROUTE_CREATE<br>ROUTE_CREATE_KEM_INFO<br>ROUTE_CREATE_VALIDATE_AND_PUBLISH<br>ROUTE_CREATE_PING<br>ROUTE_CREATE_PONG<br>ROUTE_CREATE_OK<br>ROUTE_CREATE_FAIL                                        | Creates physical routes that allow virtual nodes to publish entry points in the DRT.                                                         |

| Layer    | Protocol family          | Message types                                                                                                                                                                                                                | Purpose                                                                                                                                                             |
|----------|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Physical | Route execute            | ROUTE_DATA                                                                                                                                                                                                                   | Carries virtual envelopes through already created physical routes.                                                                                                  |
| Virtual  | Virtual session          | VIRTUAL_SESSION_INIT<br>VIRTUAL_SESSION_INIT_OK<br>VIRTUAL_SESSION_KEY_CONFIRM<br>VIRTUAL_SESSION_READY<br>VIRTUAL_SESSION_KEEPALIVE<br>VIRTUAL_SESSION_KEEPALIVE_ACK<br>VIRTUAL_SESSION_CLOSE                               | Establishes and maintains end-to-end sessions between virtual nodes.                                                                                                |
| Virtual  | Virtual reliable session | VIRTUAL_SESSION_RELIABLE_DATA<br>VIRTUAL_SESSION_RELIABLE_ACK                                                                                                                                                                | Wraps virtual-session payloads with ordered delivery, acknowledgments, retransmission, and optional synchronous inner virtual responses carried by acknowledgments. |
| Virtual  | Virtual message          | VIRTUAL_SESSION_DATA                                                                                                                                                                                                         | Transports application-level messages over an active virtual session.                                                                                               |
| Virtual  | Virtual content          | VIRTUAL_CONTENT_INFO_REQUEST<br>VIRTUAL_CONTENT_INFO_RESPONSE<br>VIRTUAL_CONTENT_RANGE_REQUEST<br>VIRTUAL_CONTENT_RANGE_RESPONSE<br>VIRTUAL_CONTENT_NOT_FOUND<br>VIRTUAL_CONTENT_RANGE_DENIED<br>VIRTUAL_CONTENT_RANGE_ERROR | Retrieves content metadata and byte ranges from virtual content holders.                                                                                            |

## 7.10 LOCAL PERSISTENCE MODEL

Local persistence is implemented with SQLite through SQLAlchemy. Persistent state includes identities, endpoints, DHT records, route metadata, content metadata, and operational logs. Runtime-only state such as active sockets, handshake state,

cryptographic session secrets, temporary buffers, and reliable-session delivery windows remains in memory.

The main persistence rule is that stable identifiers and metadata are persisted, while active cryptographic material, live network resources, pending reliable acknowledgments, in-flight retransmission state, and ordered-delivery buffers are not persisted without protection.

Table 12 summarizes the main persistence entities.

Table 12 – Summary of local persistence entities.

| Entity                        | Purpose                                                                         |
|-------------------------------|---------------------------------------------------------------------------------|
| local_physical_node_identity  | Stores the local physical node identity and key material.                       |
| local_virtual_node_identity   | Stores virtual identities hosted by the local physical node.                    |
| remote_physical_node_identity | Caches known remote physical nodes and operational metadata.                    |
| node_endpoint                 | Stores advertised endpoints for known physical nodes.                           |
| remote_virtual_node_identity  | Caches known remote virtual nodes.                                              |
| dht_record                    | Stores local replicas of distributed records from DPNT, DRT, DDT, DPT, and DTT. |
| route_resolution              | Stores local route mappings required by initiators, hops, and final endpoints.  |
| content_object                | Stores metadata for locally persisted content objects.                          |
| content_advertisement         | Tracks content announcements published by local virtual nodes in the DDT.       |
| seen_hash                     | Avoids repeated processing of already seen packets, messages, or objects.       |
| local_event_log               | Stores operational events for debug and audit support.                          |

This model supports the core requirement of the architecture: distributed references use compact hash-based identifiers, while complete public keys remain available when signatures or identity reconstruction are required.

## 7.11 LOCAL API

The local API exposes core functionality to external applications. The API is local by default and is intended to be consumed by desktop applications, local HTML interfaces, scripts, and integration tests.

The HTTP API is exposed at:

`http://127.0.0.1:18080` (7.14)

The WebSocket event stream is exposed at:

`ws://127.0.0.1:18081/v1/events` (7.15)

The API includes endpoints for health checks, debug snapshots, virtual node management, virtual signatures, DHT publish/query operations, virtual sessions, virtual messages, content storage, content downloads, and event subscriptions. Long operations, such as DHT publication from the user interface, can be executed through asynchronous jobs.

Table 13 summarizes the main API groups.

Table 13 – Summary of implemented local API groups.

| API group          | Purpose                                                                                   |
|--------------------|-------------------------------------------------------------------------------------------|
| Status and debug   | Provides <code>/health</code> , <code>/v1/status</code> , and <code>/debug/state</code> . |
| Virtual nodes      | Creates and lists local VNs and registers known remote VNs.                               |
| Virtual signatures | Allows applications to sign payloads with local VN keys and verify VN signatures.         |
| DHT                | Computes DHT keys, publishes records, queries records, and manages publish jobs.          |
| Virtual sessions   | Opens, lists, sends messages through, and closes virtual sessions.                        |
| Virtual messages   | Subscribes to application message types and reads received messages.                      |
| Content            | Stores local content, retrieves metadata, reads byte ranges, and publishes DDT holders.   |
| Downloads          | Starts and tracks virtual content downloads.                                              |
| WebSocket events   | Emits events such as received virtual messages and content publication updates.           |

The API does not replace the internal protocols. Instead, it calls the corresponding services and protocol clients inside the core. This design allows external applications to use the network without depending on internal Python modules.

From the application's perspective, this boundary turns distributed networking operations into a small local service interface. Profile publication invokes content storage followed by DDT and DPT publication; feed reconstruction invokes DPT resolution, DDT holder discovery, and content download; direct messaging invokes DRT resolution and virtual-session establishment; and WebSocket events deliver incoming messages without requiring the browser to poll protocol state. The application therefore benefits from

peer discovery, route construction, persistence, cryptographic sessions, and transport heterogeneity while remaining independent of their internal implementations.

Figure 18 provides a didactic view of this boundary. An external application submits a request through the local HTTP API, receives asynchronous events through WebSocket, and delegates identity protection, route resolution, session establishment, persistence, and forwarding to the Core. The destination Core reverses the layered processing and delivers the resulting application event locally, while intermediate physical peers process only the forwarding and hop-protection information required at their respective positions.

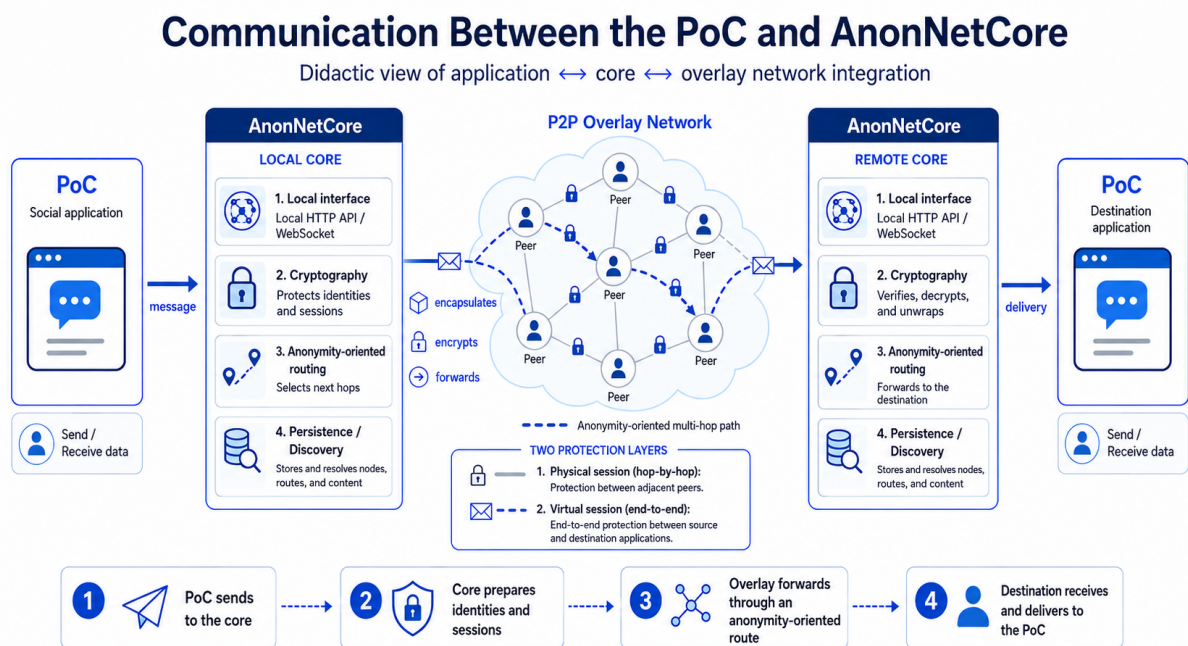


Figure 18 – Integration flow between the social PoC, the local Core, the physical overlay route, and the destination application.

## 7.12 SOCIAL PROOF OF CONCEPT

A social proof of concept was implemented to validate the architecture from the perspective of an external application. The PoC is a local HTML/JavaScript interface that communicates with the core through the local API.

The social model is intentionally simple: in the social PoC, one virtual node corresponds to one social profile.

A profile state contains name, biography, profile photo, friend list, posts, and local metadata. The current profile state is stored as a local content object. The virtual node then publishes itself as a holder of that content in the DDT and updates a DPT pointer to reference the most recent state object.

The DPT logical key used by the PoC is:

$$\text{logical\_key} = \text{virtual\_node\_id}|\text{profile} \quad (7.16)$$

The physical DHT key is derived from this logical key:

$$\text{dpt\_key} = \text{SHA512}(\text{dpt}|\text{virtual\_node\_id}|\text{profile}) \quad (7.17)$$

The DPT record contains:

$$\text{target\_ref} = \text{content\_id} \quad (7.18)$$

Since the DPT is signed by the owner virtual node, other peers can verify that the pointer belongs to the claimed profile. During validation, peers compute  $\text{SHA512}(\text{pk\_virtual\_node\_owner})$  and confirm that it matches the `virtual_node_id` used in the DPT key. To read a friend's profile, the application queries the friend's DPT, validates the pointer, resolves the referenced content in the DDT, downloads the content from a holder, and renders the posts in the feed.

Direct messages use virtual sessions. The application requests a session with a friend virtual node, the core resolves the friend's DRT entry, resolves the entry point in the DPNT, establishes the virtual session through `ROUTE_DATA`, and sends the message using `VIRTUAL_SESSION_DATA` with the application type `social.direct_message`.

### 7.13 EXPERIMENTAL EXECUTION

The implementation includes three main execution scripts: one to run a local core, one to run the social PoC with cluster and Debug Console support, and one to run the official integration smokes. The following commands are representative examples:

```
.\.venv\Scripts\python.exe scripts\run_core.py
```

```
.\.venv\Scripts\python.exe scripts\run_poc.py 10
```

```
.\.venv\Scripts\python.exe scripts\run_smokes.py 10
```

The experimental environment validates execution with multiple local or containerized nodes. The Debug Console and the `/debug/state` endpoint were used to inspect peers, sessions, DHT records, route state, runtime status, and potential operational inconsistencies.

## 7.14 VALIDATION AND TEST RESULTS

The MVP was validated through smoke tests and the social PoC. The tests are flow-oriented: instead of only testing isolated functions, they instantiate real cores and exercise protocol interactions across the implemented layers.

Table 14 summarizes the main validation tests.

Table 14 – Implemented smoke tests and validation goals.

| Status | Test                                 | Validation goal                                                                                                                                                                                                                                                                                                                                  |
|--------|--------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ✓      | <code>core_full_flow_smoke.py</code> | Validates the core as an integrated system, including Docker cluster startup, randomized transport topology, bootstrap, physical peer validation, DHT publish/query, route maintenance, DRT resolution, virtual sessions, virtual messages, content transfer, local API access, WebSocket events, reliable delivery, and debug-state inspection. |
| ✓      | <code>poc_full_flow_smoke.py</code>  | Validates the external social PoC through the public local API, including creation of two virtual-node profiles, DPT/DDT publication, friendship, remote profile resolution, feed construction from a friend's posts, direct messages, and browser-facing JavaScript service behavior.                                                           |
| ✓      | <code>scripts/run_smokes.py</code>   | Orchestrates the official smoke suite, resets local cluster data, starts the requested number of nodes, prints the randomized topology, runs the core and PoC flows in order, reports pass/fail status, and stops the cluster after execution.                                                                                                   |

Table 15 summarizes the functional outcomes observed during validation.

Table 15 – Summary of experimental validation results.

| Experiment                          | Observed result | Interpretation                                                                                                             |
|-------------------------------------|-----------------|----------------------------------------------------------------------------------------------------------------------------|
| Local physical cluster execution    | ✓               | Multiple physical nodes were instantiated and connected through the bootstrap process.                                     |
| Virtual node creation               | ✓               | The core was able to create and maintain independent virtual identities.                                                   |
| DRT route publication               | ✓               | Virtual node reachability information was published and resolved through DHT records.                                      |
| Virtual session establishment       | ✓               | Virtual endpoints established sessions over physical route structures.                                                     |
| Virtual message delivery            | ✓               | Application messages were delivered through the virtual session pipeline.                                                  |
| Content advertisement and retrieval | ✓               | Content objects were stored, advertised in the DDT, resolved, and downloaded using byte ranges.                            |
| Social PoC integration              | ✓               | The external application consumed the local API to create profiles, posts, friendships, feed entries, and direct messages. |

Together, the results cover the complete prototype path: cluster bootstrap and physical validation; virtual identity and route maintenance; DRT/DPNT resolution; virtual sessions and messages; byte-range content publication and retrieval; external PoC access through the local API; and runtime inspection of peers, sessions, DHT records, routes, and warnings. Figures 19, 20, 21, and 22 retain one representative artifact for each validation surface. The complete screenshot set remains available in the repository under `thesis/assets/img`.

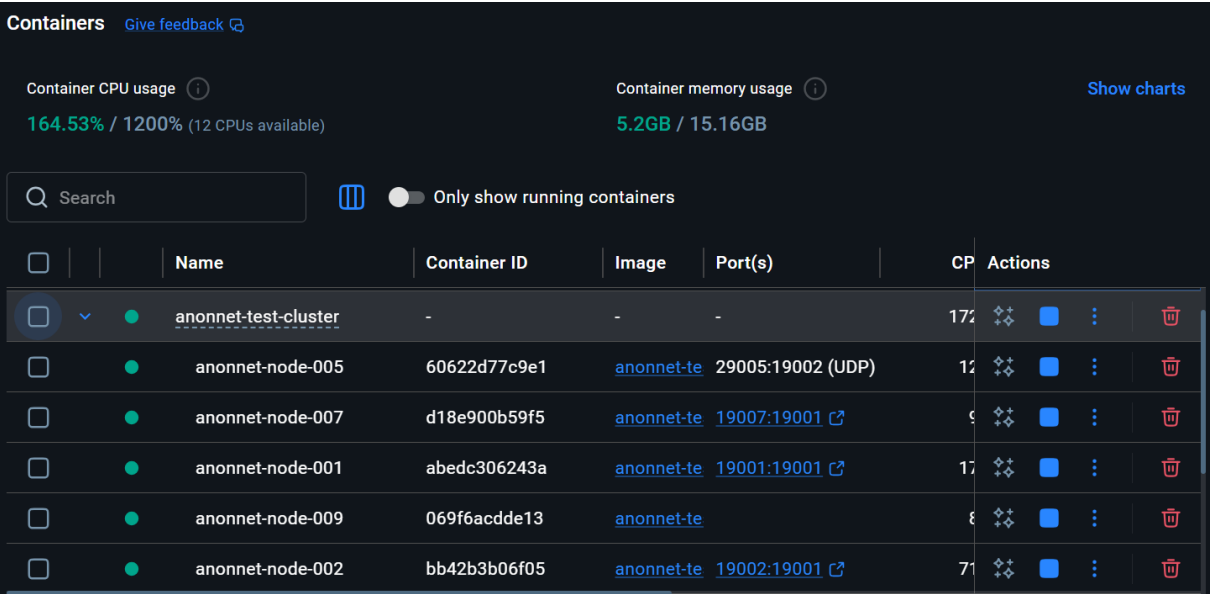


Figure 19 – Cluster execution console showing multiple physical nodes running in the experimental Docker environment.

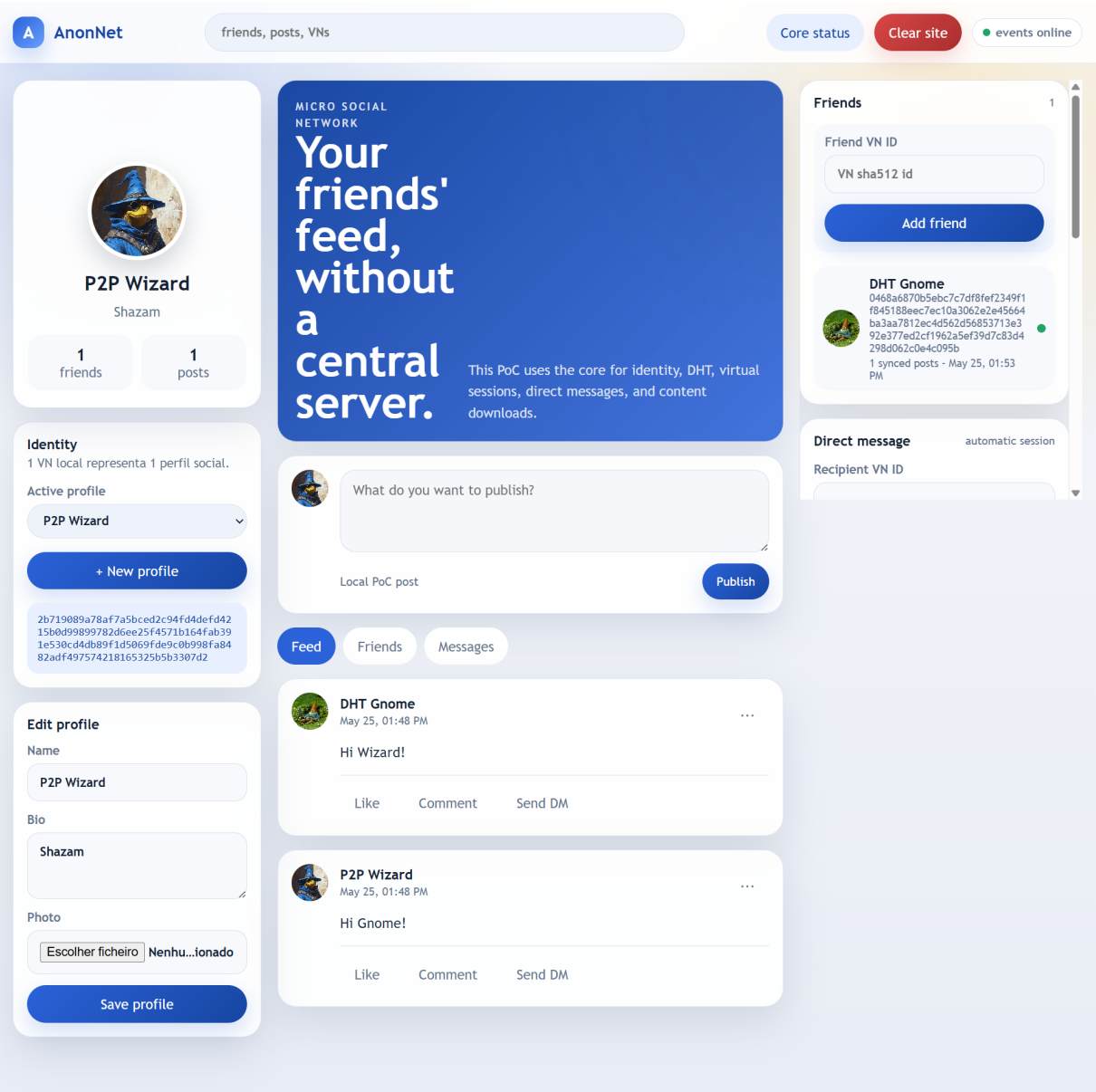


Figure 20 – Social proof of concept showing profile state, friends, and feed synchronization through the local API.

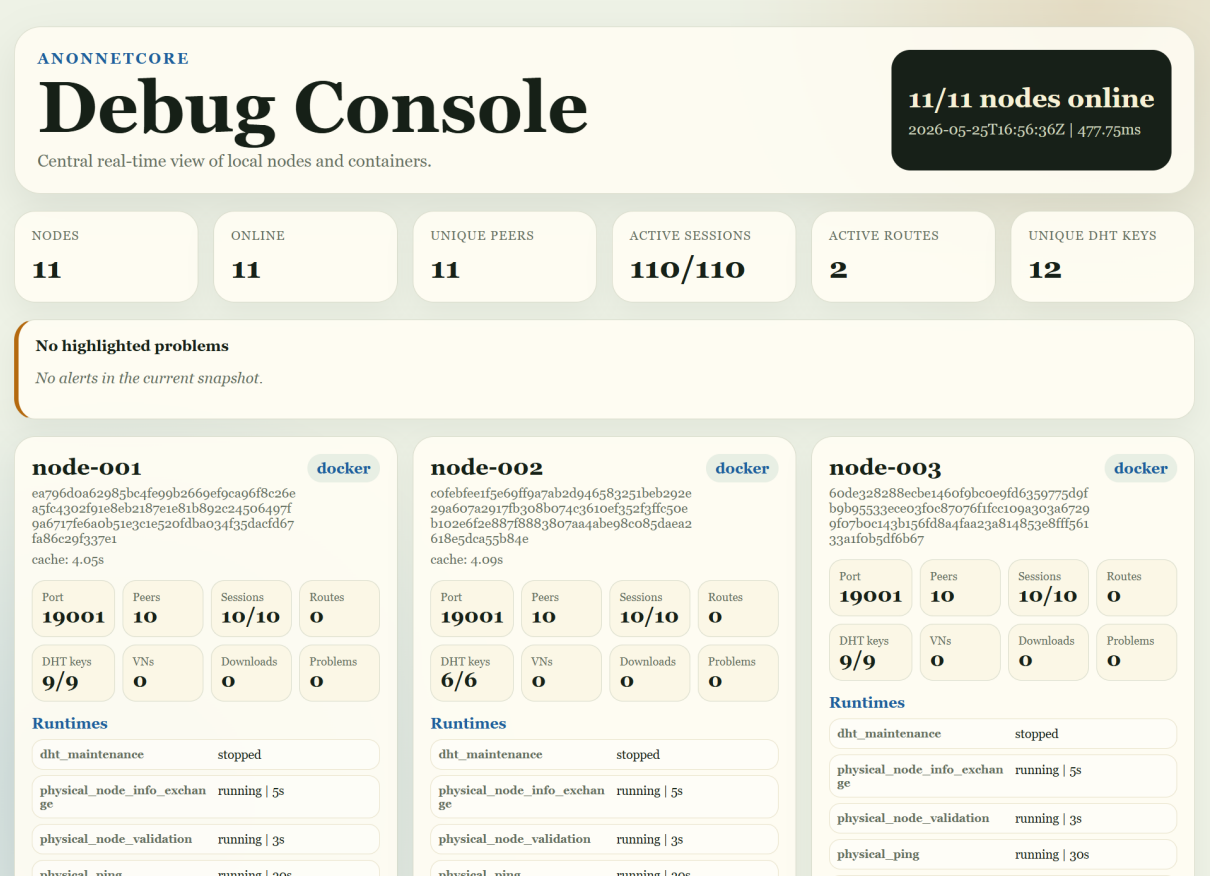


Figure 21 – Debug Console overview used to observe the experimental execution.

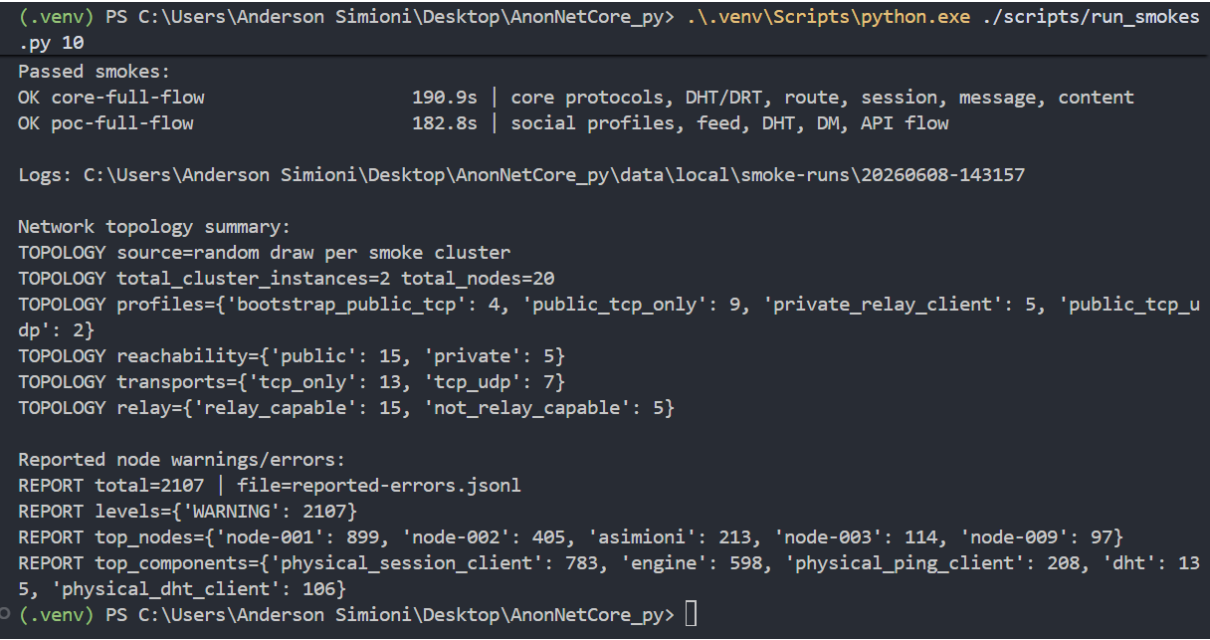


Figure 22 – Final smoke-test summary showing the successful integrated Core and social PoC validation flows.

## 7.15 COMPARISON WITH RELATED SYSTEMS AND CONCEPTS

The implementation is not intended to reproduce an existing system directly. Instead, it combines concepts from structured DHTs, peer-to-peer overlays, anonymous routing, and post-quantum cryptography into a prototype tailored to the proposed architecture.

Table 16 summarizes the main comparisons.

Table 16 – Comparison between the implemented MVP and related systems or concepts.

| Reference concept                                                                                             | Similarity                                                                                                                                                         | Difference in the MVP                                                                                                                                                                               |
|---------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Kademlia / DHTs (MAY-MOUNKOV; MAZIERES, 2002)                                                                 | The MVP uses hash-based keys, XOR distance, and responsibility by proximity.                                                                                       | The implementation does not yet implement a complete Kademlia routing table, iterative lookup procedure, bucket management, or mature churn handling.                                               |
| Peer-to-peer overlay networks (STOICA et al., 2001), (MAYMOUNKOV; MAZIERES, 2002), (ROWSTRON; DRUSCHEL, 2001) | The architecture separates logical overlay behavior from physical network connections and allows nodes to exchange resources without a central application server. | The prototype currently runs in controlled local or Docker environments and still requires future work for Internet-scale churn, NAT traversal, and large-scale peer discovery.                     |
| Tor / onion routing (DINGLELINE; MATHEWSON; SYVERSON, 2004)                                                   | The route layer also separates intermediate forwarding from final virtual semantics, and intermediate hops do not need to know the complete route.                 | The MVP does not implement Tor's complete onion routing design, directory authority model, circuit construction protocol, relay ecosystem, or mature resistance against global traffic correlation. |

Continued on next page.

Table 16 – Comparison between the implemented MVP and related systems or concepts (continued).

| Reference concept                                                                                                                                              | Similarity                                                                                                                                                                     | Difference in the MVP                                                                                                                                                                                                                                             |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| I2P / garlic routing (ALOTHMAN et al., 2018)                                                                                                                   | The architecture shares the goal of hiding communication relationships through indirect overlay routing and separation between network-level and application-level identities. | The MVP does not implement I2P's garlic routing model, tunnel management, network database, lease sets, or mature anonymity mechanisms, focusing instead on validating virtual identities, DHT-based discovery, and route construction in a simplified prototype. |
| Content-addressed systems (BENET, 2014)                                                                                                                        | Content is addressed by hash and announced through distributed references.                                                                                                     | The MVP separates pointer resolution, holder discovery, and virtual content transfer through DPT, DDT, and virtual sessions rather than reproducing IPFS directly.                                                                                                |
| Post-quantum cryptography (BOS et al., 2017), (National Institute of Standards and Technology, 2024a), (National Institute of Standards and Technology, 2024b) | The implementation uses post-quantum KEM and signature algorithms when available, combined with symmetric encryption for session payloads.                                     | The prototype has not undergone formal cryptographic audit, side-channel analysis, or production hardening of key storage and cryptographic lifecycle management.                                                                                                 |

The main contribution of the implementation is the integration of these ideas into a layered architecture in which physical nodes, virtual identities, DHT records, route entries, sessions, and application objects are explicitly separated.

## 7.16 KNOWN LIMITATIONS AND FUTURE WORK

The MVP validates the core architecture, but several components require further development before the system can be considered robust in public networks. Table 17 summarizes the main limitations and future work.

Table 17 – Current limitations and future work.

| Area               | Current limitation                                                                                                                                                                                             | Future work                                                                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| NAT traversal      | The MVP includes TCP, experimental UDP transport, and relay-assisted physical forwarding, but it does not implement complete NAT traversal.                                                                    | Implement UDP hole punching, STUN/TURN support, relay hardening, and later QUIC-based transport.                                                         |
| DHT routing        | The DHT uses XOR proximity and replication but does not yet include a complete Kademlia-style routing table.                                                                                                   | Implement buckets, iterative lookup, routing table refresh, churn handling, and more complete proximity search.                                          |
| Anti-abuse         | The MVP includes lightweight DHT publication proof-of-work, but there is no mature defense against Sybil attacks, flooding, spam, or coordinated malicious peers.                                              | Add adaptive proof-of-work, peer scoring, rate limits per peer and namespace, invalid signature penalties, and origin concentration detection.           |
| Scalability        | Tests currently focus on local processes and Docker clusters.                                                                                                                                                  | Execute long-running experiments with larger clusters, churn, latency, packet loss, and heterogeneous nodes.                                             |
| Security hardening | The MVP has not undergone formal security audit and local private key protection still requires strengthening.                                                                                                 | Harden key storage, review cryptographic lifecycles, add secure local access control, and perform external audits.                                       |
| Route privacy      | The route mechanism reduces local knowledge at each hop, but does not provide full protection against global correlation.                                                                                      | Study stronger onion-like encapsulation, cover traffic, route rotation, and traffic analysis resistance.                                                 |
| Reliability        | The MVP implements basic reliable session payloads with sequence numbers, acknowledgments, duplicate detection, ordered delivery, and retransmission, but it does not yet include advanced transport behavior. | Add congestion control, adaptive backoff, route failover, session migration policies, and broader reliability coverage for additional control protocols. |

Continued on next page.

Table 17 – Current limitations and future work (continued).

| Area                | Current limitation                                                                              | Future work                                                                                                                                |
|---------------------|-------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| DTT and large lists | The DTT is modeled but not central to the current PoC, and large DDT/DTT lists are not sharded. | Implement sharding, pagination, tag indexing, expiration policies, and garbage collection for large distributed records.                   |
| Application PoC     | The social PoC stores profile photos as data URLs and downloads complete friend states.         | Store media as separate content objects, add pagination, access control, privacy settings, moderation, and efficient feed synchronization. |

These limitations do not invalidate the MVP. Instead, they define the boundary between an experimental prototype and a production-grade decentralized network.

## 7.17 CHAPTER SUMMARY

This chapter presented the implementation of the proposed architecture as a functional experimental MVP. The prototype implements a physical layer, virtual layer, TCP transport, experimental UDP transport, relay-assisted physical forwarding, DHT namespaces, route construction, route execution, reliable physical and virtual session payloads, content transfer, local HTTP/WebSocket API, and a social proof of concept.

The implementation demonstrates that the proposed separation between physical nodes and virtual nodes is feasible, that distributed records can be used to locate physical nodes, virtual routes, content holders, and mutable pointers, and that an external application can use the network through a local API. The smoke tests and social PoC validate the main protocol flows expected from the architecture, including route maintenance, DHT publication proof-of-work, route-based virtual sessions, direct messages, content download by byte ranges, and runtime observability.

At the same time, the implementation remains intentionally limited. It is a research prototype designed to validate architecture and integration, not a hardened public network. Future work should focus on complete NAT traversal, relay hardening, complete Kademlia-style routing, stronger anti-abuse mechanisms, scalability tests, reliability under churn and packet loss, cryptographic hardening, formal auditing, and more realistic application-level privacy controls.

## 8 CONCLUSION

This work presented the design, implementation, and experimental validation of a decentralized peer-to-peer overlay network architecture focused on secure communication, data persistence, and support for decentralized applications. The proposed architecture was motivated by the limitations of centralized communication models, the increasing relevance of privacy-preserving systems, and the need to investigate network designs that combine peer-to-peer communication, distributed discovery, content-addressed storage, routing abstraction, and post-quantum cryptographic mechanisms.

The main objective of the project was achieved through the development of the *AnonNetCore Python MVP*, an experimental prototype that implements the core concepts proposed throughout this work. The implemented system separates physical nodes, which represent real network processes, from virtual nodes, which represent logical overlay identities. This separation allowed the architecture to decouple transport-level connectivity from application-level identity, enabling multiple virtual identities to coexist over the same physical node and allowing applications to interact with the network through a higher-level abstraction.

The implementation also demonstrated that distributed tables can be used as a common foundation for multiple discovery and persistence functions. The DPNT was used to describe and locate physical nodes; the DRT was used to publish entry points for virtual nodes; the DDT was used to announce content holders; and the DPT was used to maintain mutable signed references to application state. Together, these structures provided a practical mechanism for peer discovery, route resolution, content lookup, and application-level state publication. The MVP also introduced lightweight publication proof-of-work for DHT payloads, adding an explicit computational cost to distributed writes while preserving cheap verification by responsible peers.

Another important result was the implementation of route creation and route execution mechanisms. The prototype showed that virtual communication can be transported through physical routes using path identifiers and encapsulated virtual envelopes. In this model, intermediate physical nodes only need to process local forwarding information, while virtual sessions remain logically end-to-end between virtual identities. This validates the feasibility of the proposed separation between the physical routing layer and the virtual communication layer.

The physical communication layer also advanced beyond the original TCP-only validation path. The MVP includes TCP transport, experimental UDP transport with compact binary fragmentation, and relay-assisted TCP forwarding for private physical

nodes. These mechanisms are still experimental, but they validate the modular transport abstraction expected by the architecture and show that direct and relay-assisted communication can be handled without changing the higher-level virtual-session model.

The security and delivery mechanisms were also partially validated. The prototype uses post-quantum key-establishment and signature primitives when available, AES-256-GCM-SIV authenticated encryption, associated authenticated data for selected public metadata, and reliable session payloads with sequence numbers, acknowledgments, duplicate detection, ordered delivery, and bounded retransmission. These mechanisms do not make the prototype production-ready, but they demonstrate that the architecture can combine hop-by-hop protection, end-to-end virtual sessions, and practical delivery controls in a single protocol stack.

The local API and the social proof of concept further demonstrated that the implemented core can be consumed by external applications without requiring direct access to internal protocol modules. The social PoC used virtual nodes as profiles, DPT records as mutable profile pointers, DDT records for content holder discovery, DRT records for virtual node reachability, and virtual sessions for direct messages. This integration provided evidence that the proposed architecture can support decentralized application workflows beyond isolated protocol tests.

The experimental validation was performed through smoke tests, integration scripts, local clusters, debug snapshots, and the social proof of concept. These tests exercised the main protocol flows of the MVP, including bootstrap, peer validation, heterogeneous transport execution, DHT publication and query, route maintenance, virtual session establishment, reliable virtual message delivery, content transfer by byte ranges, HTTP/WebSocket API usage, and social profile synchronization. Therefore, although the prototype was evaluated in a controlled environment, it successfully demonstrated the functional viability of the proposed architecture.

However, the implementation must be understood as an experimental MVP rather than a production-ready network. Several limitations remain. The current prototype does not yet implement complete NAT traversal, QUIC transport, mature relay infrastructure, full Kademlia-style routing tables, robust anti-abuse mechanisms, Sybil resistance, reputation systems, large-scale churn handling, formal cryptographic auditing, or complete protection against global traffic correlation. These limitations are expected for a research prototype and define the boundary between the validated architectural model and a deployable public network.

Even with these limitations, the project contributes a concrete and extensible architecture for decentralized overlay networks. Its main contribution is not only the implementation of individual mechanisms, but the integration of physical nodes, virtual

nodes, transport adapters, relay forwarding, distributed tables, proof-of-work-protected DHT writes, route construction, virtual sessions, reliable payload delivery, content transfer, local APIs, WebSocket events, observability tools, and an application-level proof of concept into a single coherent experimental system. This integration demonstrates that the proposed model can serve as a foundation for future research and development in secure decentralized applications.

Future work should focus on hardening the prototype and extending it toward real-world deployment conditions. The most relevant directions include implementing UDP hole punching, hardening relay fallback, adding QUIC support, replacing the simplified DHT behavior with a complete Kademlia-like routing layer, evolving publication proof-of-work and peer scoring, improving protection against Sybil and flooding attacks, executing larger-scale experiments with churn and packet loss, strengthening local key protection, separating media objects from social profile state, and performing a formal security review.

In conclusion, this work showed that it is feasible to design and implement a modular decentralized overlay architecture that supports secure communication, distributed discovery, persistent content references, virtual identities, and decentralized application integration. The resulting MVP validates the core architectural hypothesis of the project and establishes a practical foundation for future iterations aimed at scalability, robustness, stronger anonymity, and production-grade security.

## DECLARATION OF ARTIFICIAL INTELLIGENCE USE

Artificial Intelligence tools were used as auxiliary support during the development of this work, exclusively to assist with the revision, organization, standardization, and formal improvement of texts, code fragments, technical documentation, and  $\text{\LaTeX}$  structures.

The purpose of using these tools was to improve clarity, cohesion, formatting, technical terminology, and academic style. These tools were not used to replace the intellectual authorship of the work, define the results, conduct the research, create the scientific proposal, or generate technical content without validation by the author.

All conceptual, architectural, methodological, and technical decisions presented in this work were defined, reviewed, and validated by the author. The final content, including explanations, models, design choices, code, analyses, and conclusions, remains the full responsibility of the author.

## REFERENCES

- ALOTHMAN, O. et al. An empirical study of the i2p anonymity network and its censorship resistance. In: *Proceedings of the Internet Measurement Conference*. [S.l.]: ACM, 2018. p. 379–390. 89
- AUGUSTINE, J.; CHATTERJEE, S.; PANDURANGAN, G. A fully-distributed scalable peer-to-peer protocol for byzantine-resilient distributed hash tables. In: *Proceedings of SPAA 2022*. New York: ACM, 2022. p. 87–98. 14, 15, 16, 18, 19, 26, 28, 29, 30, 31, 32, 35, 36, 37, 38, 39, 40, 43, 45, 47, 48, 49, 50, 54, 55, 56, 57, 58, 59, 60
- BAUMGART, I.; MIES, S. S/kademlia: A practicable approach towards secure key-based routing. In: *2007 International Conference on Parallel and Distributed Systems*. [S.l.]: IEEE, 2007. p. 1–8. 52
- BENET, J. *IPFS: Content Addressed, Versioned, P2P File System*. 2014. ArXiv:1407.3561. 89
- BOS, J. et al. *CRYSTALS–Kyber: A CCA-Secure Module-Lattice-Based Key-Encapsulation Mechanism*. 2017. Cryptology ePrint Archive, Paper 2017/634. 16, 17, 29, 30, 32, 46, 47, 49, 50, 51, 54, 59, 60, 89
- CASTRO, M. et al. Secure routing for structured peer-to-peer overlay networks. In: *Proceedings of OSDI 2002*. [S.l.]: USENIX Association, 2002. 52
- DINGLELINE, R.; MATHEWSON, N.; SYVERSON, P. Tor: The second-generation onion router. In: *Proceedings of the 13th USENIX Security Symposium*. [S.l.]: USENIX Association, 2004. p. 303–320. 88
- DOUCEUR, J. R. The sybil attack. In: *Peer-to-Peer Systems: IPTPS 2002*. [S.l.]: Springer, 2002. p. 251–260. 52
- GUERON, S.; LANGLEY, A.; LINDELL, Y. *AES-GCM-SIV: Nonce Misuse-Resistant Authenticated Encryption*. [S.l.], 2019. 17, 46, 51
- HEINANEN, J.; GUERIN, R. *A Single Rate Three Color Marker*. [S.l.], 1999. 52
- IYENGAR, J.; THOMSON, M. *QUIC: A UDP-Based Multiplexed and Secure Transport*. [S.l.], 2021. 16
- KERANEN, A.; HOLMBERG, C.; ROSENBERG, J. *Interactive Connectivity Establishment (ICE)*. [S.l.], 2018. 15, 16
- LI, G.; WANG, C.; WANG, H. Unreachable peers communication scheme in decentralized networks based on peer-to-peer overlay approaches. *Future Internet*, v. 14, n. 10, p. 290, 2022. 15, 16, 26, 28, 29, 30, 31, 32, 35, 36, 37, 38, 39, 40, 41, 43, 45, 46, 48, 49, 50, 51, 57, 58, 60
- MAYMOUNKOV, P.; MAZIERES, D. Kademlia: A peer-to-peer information system based on the xor metric. In: *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*. Berlin: Springer, 2002. p. 53–65. 14, 18, 19, 20, 28, 29, 30, 31, 32, 36, 38, 40, 58, 88

- Mehr-Un-Nisa et al. Comparative analysis of unstructured p2p file sharing networks. In: *Proceedings of ICISDM 2019*. New York: ACM, 2019. p. 148–153. 14, 18, 26, 28, 29, 30, 43, 47
- MIRZAEI, R.; FARAHANI, R.; YAZDANI, N. Public-key based routing in internet for anonymous data communication. In: *2018 9th International Symposium on Telecommunications*. [S.l.]: IEEE, 2018. p. 58–64. 26, 28, 29, 30, 31, 32, 36, 37, 38, 39, 40, 41, 43, 46, 47, 48, 49, 50, 51, 54, 57, 58, 59, 60
- National Institute of Standards and Technology. *FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard*. Gaithersburg, 2024. 17, 29, 30, 32, 46, 47, 49, 50, 51, 54, 59, 60, 89
- National Institute of Standards and Technology. *FIPS 204: Module-Lattice-Based Digital Signature Standard*. Gaithersburg, 2024. 46, 51, 89
- PETIT-HUGUENIN, M. et al. *Session Traversal Utilities for NAT (STUN)*. [S.l.], 2020. 15, 16
- QIAN, C. et al. Garlic cast: Lightweight and decentralized anonymous content sharing. In: *2016 IEEE 22nd International Conference on Parallel and Distributed Systems*. [S.l.]: IEEE, 2016. p. 216–223. 26, 28, 29, 30, 31, 32, 36, 37, 38, 40, 41, 43, 45, 46, 47, 48, 49, 50, 51, 54, 55, 56, 57, 58, 59, 60
- REDDY, T. et al. *Traversal Using Relays around NAT (TURN)*. [S.l.], 2020. 15, 16
- REGEV, O. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, v. 56, n. 6, p. 1–40, 2009. 16, 17, 30, 51
- ROWSTRON, A.; DRUSCHEL, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: *Proceedings of Middleware 2001*. Heidelberg: Springer, 2001. p. 329–350. 14, 18, 19, 28, 29, 30, 31, 32, 36, 38, 40, 58, 88
- SHOR, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, v. 26, n. 5, p. 1484–1509, 1997. 16, 30, 51
- STOICA, I. et al. Chord: A scalable peer-to-peer lookup protocol for internet applications. In: *Proceedings of the ACM SIGCOMM 2001 Conference*. New York: ACM, 2001. p. 149–160. 14, 18, 19, 28, 29, 30, 31, 32, 36, 38, 40, 58, 88
- TOSIC, A.; VICIC, J. Spatial path selection and network topology optimisation in p2p anonymous routing protocols. *Journal of Web Engineering*, v. 21, p. 97–118, 2022. 26, 28, 30, 40, 43, 44, 57, 60
- ZHAO, B. Y.; KUBIATOWICZ, J.; JOSEPH, A. D. *Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing*. [S.l.], 2001. 14, 18, 28

## **A SCIENTIFIC ARTICLE DERIVED FROM THIS WORK**

This appendix presents the scientific article derived from this undergraduate thesis, prepared according to the Brazilian Computer Society (SBC) conference template.

# AnonNetCore: A Modular Peer-to-Peer Overlay for Secure Decentralized Applications

Anderson Simioni Assunção<sup>1</sup>, Carlos Becker Westphall<sup>1</sup>

<sup>1</sup>Department of Informatics and Statistics – Federal University of Santa Catarina (UFSC)  
Florianópolis – SC – Brazil

anderson.simioni22@gmail.com, carlosbwestphall@gmail.com

**Abstract.** *Centralized services simplify deployment but concentrate identity, communication, and data control. This paper presents AnonNetCore, a modular peer-to-peer overlay architecture that separates physical connectivity from virtual identities and application communication. The Python prototype integrates peer discovery, TCP and experimental UDP transports, relay-assisted forwarding, distributed metadata structures, strategy-based route construction, end-to-end virtual sessions, content-addressed persistence, post-quantum cryptographic primitives, authenticated encryption, reliable payload delivery, and a local HTTP/WebSocket API. Two integration smokes and a social-network proof of concept exercise the complete flow from cluster bootstrap to route publication, virtual messaging, content retrieval, and external application access. The results demonstrate functional viability in controlled local and Docker environments while identifying NAT traversal, large-scale evaluation, anti-abuse mechanisms, and formal security analysis as future work.*

**Keywords—** *peer-to-peer networks, overlay networks, decentralized applications, distributed hash tables, post-quantum cryptography*

## 1. Introduction

Centralized online services place communication, identity, and application state under a small number of administrative domains. This model is operationally convenient, but it also introduces single points of failure, censorship opportunities, broad metadata visibility, and dependence on permanent service providers. Peer-to-peer (P2P) systems distribute these responsibilities, yet practical deployments must still address dynamic peer discovery, heterogeneous reachability, route construction, distributed persistence, secure sessions, and application integration.

Classical overlays such as Chord, Kademlia, Pastry, and Tapestry demonstrate scalable key-based lookup and distributed responsibility [Stoica et al., 2001], [Maymounkov and Mazieres, 2002], [Rowstron and Druschel, 2001], [Zhao et al., 2001]. However, a complete application platform also needs to distinguish the machine that provides connectivity from the logical identity that participates in the application. It must locate private peers, represent mutable and immutable data, establish protected communication, and avoid exposing complete routes to every intermediate participant.

This work presents *AnonNetCore*, an experimental modular overlay in which physical nodes (PNs) provide connectivity and virtual nodes (VNs) provide logical identities. The main contribution is the integration of transport adapters, relay forwarding, distributed metadata, route strategies, virtual sessions, reliable delivery, content transfer, observability,

and an application-facing API into a coherent prototype. The architecture is anonymity-oriented: it limits the routing knowledge required by each hop, but does not claim formal anonymity or protection against a global traffic observer.

The remainder of this paper summarizes the literature basis, architecture, implementation, experimental validation, and limitations of the prototype. The complete source code and monograph are available at [https://github.com/andersonsimioni/AnonNetCore\\_py](https://github.com/andersonsimioni/AnonNetCore_py). The source code is also mirrored in the UFSC institutional repository at [https://codigos.ufsc.br/anderson.simioni/anonnetcore\\_py](https://codigos.ufsc.br/anderson.simioni/anonnetcore_py).

## **2. Research Basis and Method**

The architectural proposal was informed by a systematic literature review covering decentralized P2P architecture, privacy-oriented communication, secure mechanisms, routing, peer discovery, NAT traversal, and persistence. The search retrieved 2,581 records from IEEE Xplore, ACM Digital Library, ScienceDirect, Scopus, Web of Science, and SpringerLink. After duplicate removal, screening, eligibility analysis, and the strict intersection of the review dimensions, six primary studies remained in the final synthesis [Li et al., 2022, Tosic and Vicic, 2022, Mehr-Un-Nisa et al., 2019, Augustine et al., 2022, Mirzaei et al., 2018, Qian et al., 2016].

The selected studies expose complementary concerns rather than a complete common stack. Li et al. address unreachable peers and overlay-assisted communication [Li et al., 2022]; Tosic and Vicic examine topology and path selection for anonymous routing [Tosic and Vicic, 2022]; Augustine et al. study Byzantine-resilient DHT operation [Augustine et al., 2022]; Mirzaei et al. investigate public-key-oriented routing [Mirzaei et al., 2018]; and Qian et al. combine decentralized lookup with anonymous content sharing [Qian et al., 2016]. The prototype uses these concerns as design inputs and validates their integration rather than reproducing any single prior protocol.

Development followed an iterative design-and-build method. Architectural abstractions were first represented as protocols and persistence models, then validated through real multi-process flows. Integration smokes, Docker clusters, debug snapshots, and a social application were used to expose failures across layers. This methodology prioritizes functional architectural validation; it is not a performance benchmark or a formal security proof.

## **3. Proposed Architecture**

### **3.1. Layer and Identity Separation**

AnonNetCore separates physical connectivity, overlay coordination, reusable services, and applications, as summarized in Figure 1. A PN represents a running core process, owns transport endpoints, and authenticates hop-level actions. A VN is an independent logical identity hosted by a PN and authenticates route authorization, virtual sessions, and application messages. One PN may host multiple VNs, while the overlay addresses them through SHA-512 identifiers derived from their public keys.

The separation prevents applications from depending directly on sockets or physical addresses. A virtual identity can publish several route entry points, and another application can communicate with it without receiving the complete physical path or a permanent VN-to-host ownership mapping.

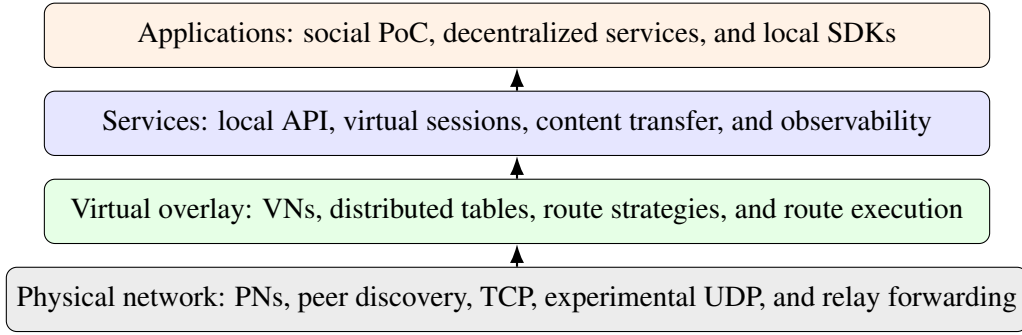


**Figure 1. Layered organization of AnonNetCore.**

### 3.2. Distributed Resolution Structures

The architecture organizes distributed state into five logical namespaces. Records are placed according to XOR proximity inspired by Kademlia and replicated among the  $K$  closest responsible PNs. Table 1 summarizes their roles.

**Table 1. Distributed structures used by the architecture.**

| Name | Resolution key             | Purpose                                                                                     |
|------|----------------------------|---------------------------------------------------------------------------------------------|
| DPNT | Physical-node identifier   | Recover validated PN descriptors, advertised transports, endpoints, and relay capabilities. |
| DRT  | Virtual-node identifier    | Resolve signed, expiring PN entry points for communication with a VN.                       |
| DDT  | Content hash               | Locate VNs that advertise possession of immutable content bytes.                            |
| DPT  | Owner VN and logical title | Maintain a signed mutable pointer to the latest immutable application state.                |
| DTT  | Normalized tag             | Map semantic tags to immutable or mutable resource identifiers.                             |

The DHT layer validates signatures, expiration, namespace-specific payloads, and lightweight proof-of-work before accepting publication. The network-wide difficulty is configured consistently, while the publisher computes the nonce before dissemination. Canonical payload serialization ensures that each peer verifies the same bytes. DHT replication and maintenance follow changing responsibility, whereas content durability remains holder-driven.

### 3.3. Route Construction and Execution

Routing is divided into two protocols. Route construction establishes hop-local mappings and publishes an authorized DRT entry. Route execution later forwards `ROUTE_DATA` using only the local mapping associated with a `path_id`. The protocol lifecycle is stable, while a pluggable strategy selects the next hop and termination condition. The implemented strategy is `random_walk_ttl`, but the separation permits hop-count, latency, bandwidth, reliability, diversity, or onion-like preselection strategies.

The final PN validates VN authorization, cryptographic nonces, and route timing through a route-level ping/pong exchange. Success publishes the route and confirms

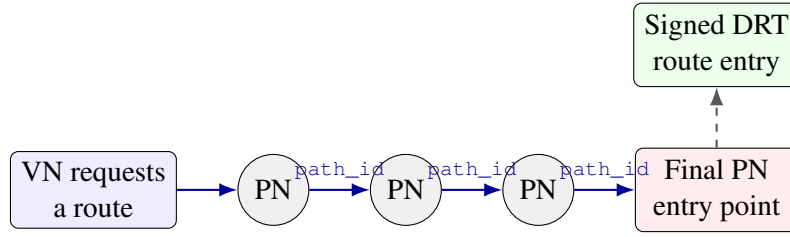


**Figure 2. Hop-by-hop route construction and DRT publication.**

it to the participating hops; failure explicitly releases pending state. During execution, intermediate PNs know only their adjacent forwarding actions. Virtual envelopes are nested inside physical route envelopes, preserving the distinction between hop-by-hop connectivity and end-to-end application communication.

### 3.4. Sessions, Connectivity, and Security

Physical sessions protect adjacent PNs, while virtual sessions protect communication between VNs. Session establishment uses ML-KEM-768 when available, and identities use ML-DSA-65 signatures when supported by the cryptographic backend [National Institute of Standards and Technology, 2024a], [National Institute of Standards and Technology, 2024b]. Session payloads use AES-256-GCM-SIV authenticated encryption, with selected public metadata bound as associated authenticated data [Gueron et al., 2019]. Reliable envelopes add sequence numbers, acknowledgments, duplicate suppression, ordered delivery, and bounded retransmission. Physical retries use hop-level timing; virtual retries account for route RTT.

All transport adapters exist in each core, but configuration determines which listeners and endpoints are advertised. Public PNs may expose TCP, UDP, or both. Private PNs maintain authenticated TCP sessions with relay-capable public PNs and advertise relay reachability. Relay payloads encapsulate physical packets without interpreting the virtual content. Experimental UDP uses compact binary fragmentation and transport-level keepalive state. Complete STUN, TURN, ICE, and UDP hole punching remain future work [Petit-Huguenin et al., 2020, Reddy et al., 2020, Keranen et al., 2018].

## 4. Prototype Implementation

The MVP is implemented in Python 3.12 using `asyncio`. SQLAlchemy and SQLite persist identities, peer descriptors, endpoints, distributed records, route mappings, content metadata, and operational events. Live sockets, session secrets, retransmission windows, and in-flight buffers remain runtime-only. The cryptographic backend provides SHA-512 identities, post-quantum primitives when available, and AES-256-GCM-SIV session protection.

Protocol handlers process bootstrap, peer information exchange, physical and virtual sessions, DHT publication and queries, route construction, route execution, relay forwarding, reliable payloads, virtual messages, and byte-range content transfer. Maintenance runtimes refresh peer validation, physical sessions, relays, DHT responsibility, virtual routes, and session keepalives.

A local HTTP API exposes VN management, signatures, distributed publication and query, virtual sessions, messages, content storage, and downloads. WebSocket events

deliver incoming application messages without polling. The API remains a boundary over the normal protocol flow; it does not bypass DRT resolution or route execution when communicating between VNs hosted by the same core.

## **5. Experimental Validation**

### **5.1. Environment and Test Flows**

Validation uses local processes and Docker clusters. Two bootstrap PNs are always public TCP nodes. Remaining containers receive a randomized topology across public TCP, public UDP, dual-transport, and private relay-assisted profiles. Cluster state, local databases, and logs are reset before each execution. Timeouts scale with cluster size so the same tests can run with different node counts while allowing peer validation and route publication to mature.

Two official integration smokes cover the system. The core full-flow smoke exercises bootstrap, heterogeneous transport topology, peer validation, DHT publication/query, route maintenance, DRT resolution, virtual sessions, virtual messages, reliable delivery, byte-range content transfer, HTTP, WebSocket, and debug-state inspection. The PoC full-flow smoke creates two VNs, publishes profile state through DDT and DPT, establishes friendship, resolves a friend's state, reconstructs the feed, and exchanges direct messages through virtual sessions.

### **5.2. Observed Results**

Table 2 reports the functional outcomes represented by the repository evidence. These results establish completion of the tested flows; they do not provide latency, throughput, anonymity, or internet-scale claims.

The social PoC provides an application-level demonstration of the same protocol stack. One VN corresponds to one profile. A profile update creates a new immutable content object, advertises its holder in the DDT, and updates the VN-signed DPT pointer. Friends resolve the DPT, locate a DDT holder, download the state, and build a feed. Direct messages use active virtual sessions and WebSocket delivery. Figure 3 shows the browser interface used during validation.

The integrated tests found and guided corrections in endpoint advertisement, route maturity, DHT replication, relay maintenance, timeout separation, UDP framing, local multi-VN operation, and shutdown behavior. The resulting prototype demonstrates that the layered abstractions can operate together under controlled heterogeneous topologies.

## **6. Limitations and Future Work**

AnonNetCore remains an educational research MVP. It does not implement complete NAT traversal, mature relay infrastructure, QUIC, a complete Kademlia bucket implementation, strong Sybil resistance, adaptive anti-flooding controls, congestion control, or formal cryptographic verification. The route design reduces per-hop knowledge but does not prevent global traffic correlation. Local key storage requires hardening, and the experimental UDP adapter requires broader testing under loss, reordering, and constrained MTUs.

The evaluation is functional and controlled. Future experiments should measure latency, throughput, route-construction cost, DHT convergence, relay overhead, packet loss,

**Table 2. Functional validation outcomes.**

| <b>Flow</b>       | <b>Outcome</b> | <b>Validated behavior</b>                                                                           |
|-------------------|----------------|-----------------------------------------------------------------------------------------------------|
| Physical cluster  | Passed         | Bootstrap and validation formed usable peer views across randomized node profiles.                  |
| DHT lifecycle     | Passed         | Signed records were published, queried, replicated, and exposed through the local API.              |
| Route lifecycle   | Passed         | Local VNs obtained routes, final PNs validated them, and DRT entries became resolvable.             |
| Virtual sessions  | Passed         | VNs established end-to-end sessions over physical route mappings and maintained keepalives.         |
| Reliable messages | Passed         | Ordered delivery, acknowledgment, duplicate suppression, and retransmission were exercised.         |
| Content transfer  | Passed         | Objects were advertised in the DDT, resolved, downloaded by byte ranges, and hash-checked.          |
| Social PoC        | Passed         | Two profiles published state, synchronized posts, formed friendship, and exchanged direct messages. |

churn, and scalability over larger and geographically distributed deployments. Security work should include threat modeling, external audit, adaptive publication costs, peer scoring, stronger route encapsulation, and application-level access control.

## 7. Conclusion

This paper presented AnonNetCore, a modular P2P overlay that separates physical connectivity from virtual identity and integrates distributed discovery, route construction, protected sessions, reliable payloads, content persistence, heterogeneous transports, and external application access. The Python MVP and social PoC validate the complete functional path from bootstrap to decentralized application interaction in local and Docker environments.

The primary contribution is architectural integration rather than a new isolated routing or cryptographic primitive. Physical and virtual identities, five distributed resolution structures, pluggable route strategies, hop-local forwarding, end-to-end virtual sessions, relay-assisted connectivity, and a local API coexist behind explicit module boundaries. The results support the feasibility of this organization while clearly delimiting the additional work required for production deployment, formal anonymity analysis, and internet-scale operation.

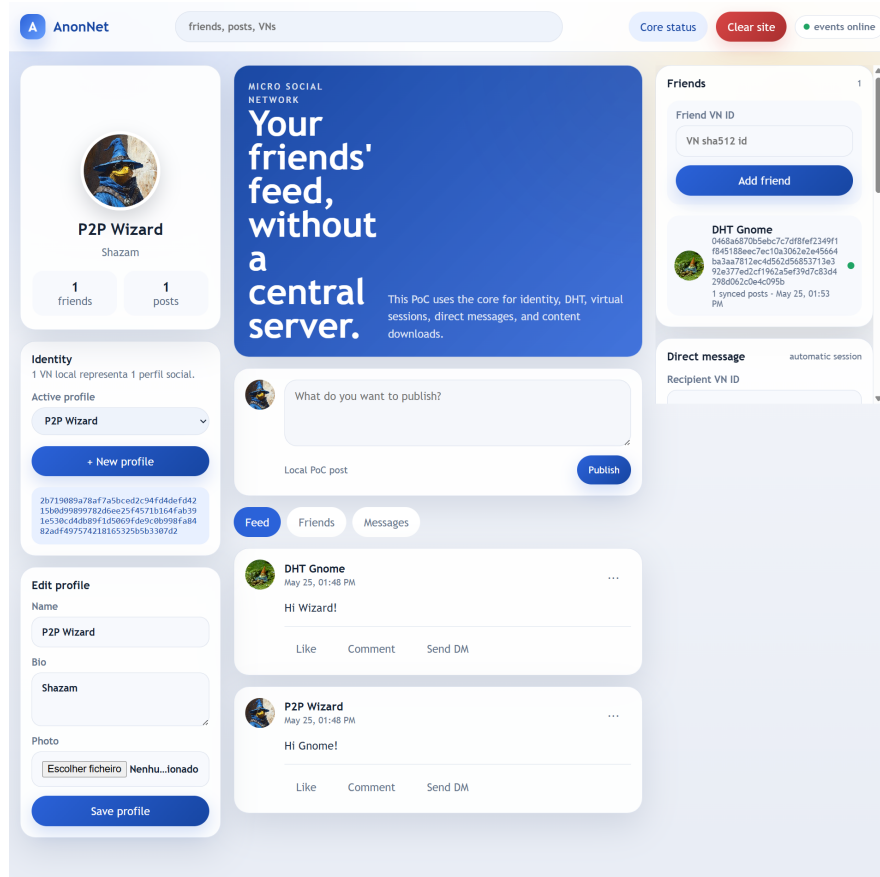


**Figure 3. Social-network proof of concept consuming the local core API.**

## References

- Augustine, J., Chatterjee, S., and Pandurangan, G. (2022). A fully-distributed scalable peer-to-peer protocol for byzantine-resilient distributed hash tables. In *Proceedings of SPAA*, pages 87–98. ACM.
- Gueron, S., Langley, A., and Lindell, Y. (2019). Aes-gcm-siv: Nonce misuse-resistant authenticated encryption. Technical Report RFC 8452, RFC Editor.
- Keranen, A., Holmberg, C., and Rosenberg, J. (2018). Interactive connectivity establishment (ice). Technical Report RFC 8445, RFC Editor.
- Li, G., Wang, C., and Wang, H. (2022). Unreachable peers communication scheme in decentralized networks based on peer-to-peer overlay approaches. *Future Internet*, 14(10):290.
- Maymounkov, P. and Mazieres, D. (2002). Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of IPTPS*, pages 53–65. Springer.
- Mehr-Un-Nisa, Ashraf, F., Naseer, A., and Iqbal, S. (2019). Comparative analysis of unstructured p2p file sharing networks. In *Proceedings of ICISDM*, pages 148–153. ACM.
- Mirzaei, R., Farahani, R., and Yazdani, N. (2018). Public-key based routing in internet for anonymous data communication. In *9th International Symposium on Telecommunica-*

- tions, pages 58–64. IEEE.
- National Institute of Standards and Technology (2024a). Fips 203: Module-lattice-based key-encapsulation mechanism standard. Technical report, NIST, Gaithersburg.
- National Institute of Standards and Technology (2024b). Fips 204: Module-lattice-based digital signature standard. Technical report, NIST, Gaithersburg.
- Petit-Huguenin, M., Salgueiro, G., Rosenberg, J., Wing, D., Mahy, R., and Matthews, P. (2020). Session traversal utilities for nat (stun). Technical Report RFC 8489, RFC Editor.
- Qian, C., Shi, J., Yu, Z., Yu, Y., and Zhong, S. (2016). Garlic cast: Lightweight and decentralized anonymous content sharing. In *22nd International Conference on Parallel and Distributed Systems*, pages 216–223. IEEE.
- Reddy, T., Johnston, A., Matthews, P., and Rosenberg, J. (2020). Traversal using relays around nat (turn). Technical Report RFC 8656, RFC Editor.
- Rowstron, A. and Druschel, P. (2001). Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of Middleware*, pages 329–350. Springer.
- Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup protocol for internet applications. In *Proceedings of ACM SIGCOMM*, pages 149–160. ACM.
- Tosic, A. and Vicić, J. (2022). Spatial path selection and network topology optimisation in p2p anonymous routing protocols. *Journal of Web Engineering*, 21:97–118.
- Zhao, B. Y., Kubiawicz, J., and Joseph, A. D. (2001). Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California, Berkeley.