DEPARTMENT OF
COMPUTER SCIENCE

JOÃO GONÇALO FREITAS PEREIRA
BSc in Science and Computer Engineering

# A DECENTRALIZED MEMBERSHIP SERVICE FOR ENVIRONMENTS WITH DYNAMIC LINKS

Dissertation Plan
MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
February, 2025

DEPARTMENT OF
COMPUTER SCIENCE

# A DECENTRALIZED MEMBERSHIP SERVICE FOR ENVIRONMENTS WITH DYNAMIC LINKS

**JOÃO GONÇALO FREITAS PEREIRA**

BSc in Science and Computer Engineering

**Adviser:** João Carlos Antunes Leitão
*Associate Professor, NOVA University Lisbon*

Dissertation Plan
MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
February, 2025

# Abstract

Swarm computing is becoming increasingly relevant in the context of distributed systems by supporting the coordination of a large number of autonomous devices in highly dynamic environments. This model is of particular interest for use cases such as satellite swarms and mobile ad hoc networks (MANETs), where intermittent connectivity and rapidly changing topologies present significant challenges. One of the fundamental components required for the efficient operation of such systems is a reliable membership service that ensures nodes are aware of the active participants in the system. However, current membership services are primarily designed for static or minimally dynamic environments and often fail to adapt to the unpredictable conditions found in highly dynamic systems like satellite swarms or MANETs. Moreover, existing tools for emulating these conditions are insufficient for accurately modeling selective communication patterns and intermittent connectivity.

This work presents a novel approach to address these challenges by proposing a decentralized membership service tailored for dynamic environments. The solution involves the design and implementation of a prototype tool to emulate dynamic communication patterns and a membership protocol that proactively adapts to environmental constraints. The membership service will leverage knowledge of communication dynamics to optimize membership management with minimal overhead. A prototype emulator integrated into the Babel framework will simulate dynamic network behaviors, providing a controlled environment to test and refine the membership protocol. Finally, the proposed solutions will be evaluated through comparative tests with existing protocols, focusing on communication overhead, membership accuracy, and resilience to network dynamics.

**Keywords:** Swarm Computing, Decentralized Systems, Peer-to-Peer Architectures, Overlay Networks, Network Emulation, Network Simulation

# Resumo

A computação em enxame (do inglês *swarm*) está a tornar-se cada vez mais relevante no contexto dos sistemas distribuídos, suportando a coordenação de um grande número de dispositivos autónomos em ambientes altamente dinâmicos. Este modelo é de particular interesse para casos de uso como enxames de satélites e redes *ad hoc* móveis, onde a conectividade intermitente e topologias em rápida mudança representam desafios significativos. Um dos componentes fundamentais necessários para o funcionamento eficiente de tais sistemas é um serviço de filiação fiável que garanta que os nós têm conhecimento dos participantes ativos no sistema. No entanto, os atuais serviços de filiação são concebidos principalmente para ambientes estáticos ou com dinamismo limitado e muitas vezes não se adaptam às condições imprevisíveis encontradas em sistemas altamente dinâmicos como enxames de satélites ou redes *ad hoc* móveis. Além disso, as ferramentas existentes para emular estas condições são insuficientes para modelar com precisão padrões de comunicação seletiva e conectividade intermitente.

Este trabalho apresenta uma nova abordagem para enfrentar estes desafios, propondo um serviço de filiação descentralizado adaptado a ambientes dinâmicos. A solução envolve a conceção e implementação de uma ferramenta protótipo para emular padrões de comunicação dinâmicos e um protocolo de filiação que se adapta proativamente às restrições ambientais. O serviço de filiação irá explorar o conhecimento da dinâmica das comunicações para otimizar a gestão da filiação com sobrecarga mínima. Um protótipo deste emulador será integrado na plataforma Babel de forma a simular comportamentos de rede dinâmicos, proporcionando um ambiente controlado para testar e aperfeiçoar o protocolo de filiação. Finalmente, as soluções propostas serão avaliadas através de testes comparativos com protocolos existentes, com foco na sobrecarga de comunicação, precisão da filiação e resiliência ao dinamismo da rede.

**Palavras-chave:** Computação em Enxame, Sistemas Descentralizados, Arquiteturas Ponto-a-Ponto, Redes Sobrepostas, Emulação de Rede, Simulação de Rede

# Contents

# List of Figures

# 1 Introduction

## 1.1 Context

Swarm computing represents a paradigm shift in distributed systems, where a large number of autonomous and potentially heterogeneous devices coordinate their execution to accomplish complex tasks with minimal to no human intervention [57]. This approach is inspired by biological systems, such as ant colonies or bee hives, which exhibit remarkable behavior when working as a group despite their limited individual capabilities [29]. Similarly, in distributed systems, swarms enable decentralized decision-making and robust task execution. This computing model is believed to be the next step for Internet of Things (IoT) [26], which comprises a global infrastructure of billions of interconnected devices that cooperate amongst themselves to achieve a common objective [50, 123].

Swarm systems offer high scalability and computational power, robustness, and adaptability [209], making them well-suited for a wide range of applications. These include routing protocols for wireless sensor networks (WSNs) [223, 81, 60]; node tracking and localization [223, 17, 16]; disaster response coordination in mobile ad hoc networks (MANETs) [1, 2]; environmental monitoring through sensor networks [173, 261, 143, 177]; industrial automation with robotic swarms for logistics and manufacturing [204, 89]; and also unmanned aerial vehicles (UAVs) [223, 209, 260, 14]. All of these applications share common characteristics, namely dynamic topologies, intermittent connectivity, and a need for autonomous operation, which present unique challenges for maintaining efficient and reliable cooperation among devices.

A critical component for the operation of swarm systems is the membership service, a foundational abstraction that provides each device with knowledge of the set of active participants in the system [27]. Membership services underpin various distributed algorithms [21], from resource allocation [244, 150, 151] to consensus [130, 174], by enabling nodes to coordinate their actions [148, 97]. In static or well-connected networks, membership management is relatively straightforward [222]. However, in environments with dynamic links, where connectivity is intermittent due to physical distance [12], orientation of radio transceivers [110, 127], energy-saving modes [112], or environmental interference [257], the design of robust and adaptive membership services becomes significantly more complex [19, 222].

Despite the ever-increasing prevalence of highly dynamic environments, existing approaches often fail to address the challenges posed by such settings. In fact, the majority of the solutions focus on static or minimally dynamic networks [19, 18], with limited capabilities to adapt to highly variable conditions. Furthermore, there is a lack of tools to model and emulate these dynamic conditions in laboratory settings, hindering the development and evaluation of solutions that aim to provide those guarantees [72, 159].

## 1.2 Problem Statement

As discussed previously, membership abstractions are a fundamental aspect of the operation of swarm systems. However, the challenges posed by dynamic topologies are exacerbated in such contexts, which leads to poor and inefficient solutions. In these systems, network configurations are constantly evolving due to node mobility, physical distance, energy constraints, and external factors such as environmental interference. This results in intermittent connectivity, making it particularly difficult to maintain consistent and up-to-date membership views [115, 114, 160].

On the other hand, the greater part of the existing membership services assumes static or barely dynamic network configurations, making them incompatible with these volatile environments [19, 18]. Moreover, traditional fault-tolerance mechanisms focus primarily on handling failures but lack the adaptive capabilities needed to proactively respond to predictable dynamics [20] or take advantage of a priori knowledge about the node's communication patterns. Additionally, maintaining consistent membership views in these environments can incur high communication overhead [136, 120], which is especially problematic in bandwidth-limited or energy-constrained settings [206].

Another pressing issue is the lack of tools and frameworks to model and emulate these dynamic network conditions in laboratory environments [72, 159]. Although existing network emulation tools can, to an extent, emulate certain traditional dynamic behaviors such as latency, bandwidth fluctuations, packet loss, jitter, and other network effects, they fall short when it comes to accurately modeling more complex scenarios. These include selective communication patterns and intermittent connectivity, which are critical characteristics of highly dynamic and decentralized environments such as satellite swarms or MANETs.

Finally, the limitations of membership services in dynamic environments have broader implications for the reliability and efficiency of swarm systems, since they are the building blocks of the majority of distributed protocols [21].

To conclude, these challenges underscore the need for robust solutions that address the unique demands of these dynamic environments. Such solutions must not only tolerate faults but also proactively adapt to the constraints and variability inherent in these systems.

## 1.3 Research Context

This work is partially motivated by the European TaRDIS [227] project (Trustworthy and Resilient Decentralised Intelligence for Edge Systems), which aims to simplify the development of correct and efficient heterogeneous swarms. TaRDIS seeks to achieve this by providing a novel event-driven programming model, decentralized machine learning primitives, and a toolbox supporting the design and execution of distributed applications.

## 1.4 Use Cases

This dissertation focuses on two primary use cases that exemplify the challenges of designing and testing robust membership services for dynamic environments:

1. Satellite swarms, a prominent use case within the TaRDIS project, involve highly dynamic topologies driven by orbital dynamics, limited communication windows, and non-fixed radio transceiver orientation.

2. Mobile ad hoc networks (MANETs) characterized by frequent topology changes due to node mobility, unpredictable environmental conditions, and resource-constrained settings.

## 1.5 Expected Outcomes

This work aims to deliver three key contributions:

1. The design and implementation of a decentralized membership service for highly dynamic environments (as those mentioned in the previous section).

2. A prototype tool to emulate dynamic, selective communication patterns.

3. An experimental evaluation comparing the proposed membership service with existing abstractions and a detailed assessment of the prototype tool's capabilities.

## 1.6 Document Structure

The remainder of this document is organized as follows: Chapter 2 presents the related work, covering peer-to-peer systems, frameworks for distributed protocols, network simulation and emulation tools, and membership abstractions. Finally, Chapter 3 elaborates on the problem statement, outlines the proposed solution and contributions, and provides a detailed task plan for the dissertation.

# 2 Related Work

In this chapter, we discuss the state-of-the-art related to decentralized communication and its challenges in environments with dynamic links and restrictive communication conditions. In particular, we begin by reviewing the peer-to-peer architecture with a special focus on overlay networks (Section 2.1). Then, we study different frameworks that enable the design of distributed protocols (Section 2.2). Next, we discuss network simulation and emulation tools (Section 2.3). After that, we review several membership abstractions (Section 2.4). Finally, we conclude with a summary of the main ideas presented in the previous sections and explain why existing solutions do not fully address the challenges posed by highly dynamic environments (Section 2.5).

## 2.1 Peer-to-Peer Systems

The peer-to-peer (P2P) [218] architecture has gained recognition over the years as a compelling alternative to the traditional client-server model, offering distinct advantages such as scalability, fault tolerance, and reduced operational costs. Unlike centralized systems, P2P systems rely on participants to contribute resources, thereby eliminating the need for dedicated infrastructure. Applications like BitTorrent [106], Skype [22], and PPLive [88] exemplify the practical relevance of P2P systems in domains such as file sharing, communication, and Internet Protocol Television (IPTV) [258].

Early P2P systems, designed for small-scale deployments, assumed complete membership information for all participants. This approach proved inefficient in large-scale and dynamic environments, where high churn rates (frequent node joins, departures, and concurrent failures) [221] make global views prohibitively expensive to maintain [140]. Consequently, modern P2P systems employ membership protocols that provide local/partial views [134], enabling the construction of overlay networks. These logical networks, built and maintained at the application level, are broadly categorized into structured and unstructured (random) overlays [145].

### 2.1.1 Structured Overlay Networks

Structured overlay networks [13, 39, 145] use global coordination to enforce predictable topologies, enabling efficient routing and services like resource location [52] and distributed storage [149]. Distributed Hash Tables (DHTs) [37] exemplify this approach, with notable implementations including Chord [219], Pastry [203], and Kademlia [154]. While structured overlays excel in routing efficiency, their rigid topologies and reliance on intricate routing infrastructures make them less robust to churn and failures.

### 2.1.2 Unstructured Overlay Networks

Unstructured overlays [74, 122, 145], by contrast, employ random and flexible topologies with lower maintenance costs, making them well-suited for dynamic environments. These overlays leverage gossip protocols and support services such as application-level multicast [116, 147], resource discovery [44, 157], and data aggregation [104, 225]. However, their flexibility often comes at the cost of inefficient routing and redundancy. Examples of unstructured overlays include Scamp [75] and Cyclon [246], which maintain neighbor sets through reactive or periodic updates.

### 2.1.3 Hybrid Overlay Networks

Recent research explores hybrid approaches [33, 234] that combine features of both structured and unstructured overlays to enhance performance and robustness. For instance, hybrid overlays can use structured components for efficient routing while leveraging unstructured elements for adaptability and resilience.

### 2.1.4 Overlay Topology Management

Overlay topology management plays a crucial role in optimizing P2P systems. Two primary strategies for managing unstructured overlays are control and bias approaches [134, 138, 51]. Control strategies impose constraints on neighbor selection, such as latency thresholds, to improve performance metrics. However, strict controls may disrupt connectivity under dynamic conditions. On the contrary, bias strategies iteratively optimize random topologies, favoring links with desirable properties (e.g., low latency). While more resilient to churn, biasing can introduce overhead.

Control-based systems implement structured strategies for overlay neighbor selection, emphasizing performance metrics such as latency and resilience. For instance, Araneola [156] selects neighbors based on proximity in the underlying topology to optimize communication efficiency. HiScamp [73] constructs hierarchical overlays aligned with autonomous system (AS) topologies, reducing inter-AS link usage in gossip dissemination. HyParView [136] ensures symmetric links for balanced in-degree distribution, enhancing robustness against node failures.

By contrast, bias-based systems iteratively refine neighbor selection to optimize overlay properties. T-Man [105] employs gossip-driven optimization guided by utility functions to improve overlay metrics such as latency. Gia [43] biases connections toward high-capacity nodes, enhancing resource discovery and query routing. X-BOT [139] dynamically optimizes active neighbors based on link costs, ensuring improved latency and cost efficiency while maintaining critical overlay properties, such as connectivity and randomness.

On the other hand, at the P2P service layer, techniques such as embedding and enrichment have been proposed to further optimize performance [134, 51]. Embed approaches create secondary topologies over existing overlays to improve protocol efficiency without altering the base structure. For example, Plumtree [135] and Thicket [65] embed robust, low-latency spanning trees, enhancing multicast performance. Enrich approaches add independent links at the service layer to support specific protocol needs [137, 117]. Systems like FASE [67] leverage this strategy to enhance resource location and robustness under churn.

Despite these advancements, challenges remain in balancing scalability, efficiency, and adaptability in P2P systems. For instance, overlay topology adaptations must maintain connectivity and node balance while addressing application-specific requirements like latency or bandwidth. Similarly, mechanisms for handling Network Address Translation (NAT) boxes and firewalls [137, 117, 58, 180, 172] must mitigate biased peer interactions without compromising performance.

## 2.2 Frameworks for Distributed Protocols

The correct design and implementation of distributed protocols is essential for ensuring fault-tolerant and dependable systems, particularly in domains like swarm computing, satellite networks, and large-scale sensor systems. However, building such algorithms is often a challenging task due to the inherent complexities of handling communication, concurrency, and fault recovery in distributed environments [70]. Researchers and developers must address several low-level concerns, such as managing communication channels, scheduling events, and implementing timeouts, which can overshadow the core logic of the protocols and lead to error-prone or inefficient implementations [118].

Frameworks specifically tailored for distributed protocols address these challenges by providing structured environments that simplify the development and evaluation of distributed algorithms. They allow developers to focus on high-level protocol logic by abstracting away low-level implementation details like concurrency management and networking primitives. This abstraction not only accelerates prototyping but also promotes correctness and reproducibility in experimental settings.

The importance of these tools extends beyond facilitating development. They enable systematic experimentation and realistic performance evaluation, which are crucial for comparing alternative designs and validating novel algorithms. By leveraging these frameworks, researchers and practitioners can build dependable systems with greater efficiency and reliability, laying the groundwork for advancements in distributed computing.

Even though network simulators, like PeerSim [166] and NS2/NS3 [101, 197], have been used for prototyping and testing distributed systems, they are not the most suitable since they fail to replicate realistic execution environments. Ultimately, this leads to discrepancies between simulated performance and real-world behavior.

In the following subsections, we discuss some of the most relevant frameworks in this context. Particularly, Cactus (Section 2.2.1), Appia (Section 2.2.2), Yggdrasil (Section 2.2.3), and Babel (Section 2.2.4). Finally, we briefly cover actor-based frameworks for building distributed applications, such as Erlang-OTP (Section 2.2.5) and Akka (Section 2.2.6).

### 2.2.1 Cactus

The Cactus framework [77] is a modular and extensible architecture for large-scale scientific and engineering simulations, used in fields like astrophysics and numerical relativity. It features a core "flesh" and interchangeable "thorns" that integrate functionalities such as parallel computing, I/O handling, and data management. Designed for portability, it supports platforms from laptops to supercomputers and abstracts complex computational technologies. Being open-source, it fosters collaboration, enabling researchers to share and improve components, enhancing quality and robustness. While flexible and powerful, its complexity and reliance on proper integration can pose challenges for new users or unconventional use cases.

### 2.2.2 Appia

Appia [162] is a protocol kernel designed to simplify the management of multiple communication channels with different quality-of-service (QoS) requirements in distributed applications. It enables explicit coordination among channels through a flexible framework that integrates inter-channel constraints. Appia allows dynamic composition and reconfiguration of communication stacks, separating static QoS definitions from dynamic implementation.

In Appia, QoS is modeled as a stack of layers, with shared sessions across channels to maintain consistency, such as in failure detection and synchronization. The system uses an open event model, allowing dynamic event definition and efficient routing, minimizing runtime overhead. Fine-grained control over session binding, both explicit and automatic, offers customizable configurations.

R-Appia [199] is an enhanced version of Appia, designed to support the dynamic reconfiguration of communication protocols. Unlike Appia, which focuses on static composition, R-Appia introduces features like context monitoring, dynamic event routing, and state transfer mechanisms, enabling it to handle adaptive communication needs more efficiently.

### 2.2.3 Yggdrasil

The Yggdrasil framework [48, 47] is a novel solution designed to facilitate the development, execution, and validation of distributed protocols and applications in wireless ad hoc networks. It addresses challenges in implementing distributed systems for commodity devices, a critical component of future edge systems. Unlike existing tools, Yggdrasil is tailored to the specific demands of wireless ad hoc settings, offering a robust foundation for creating modular, correct, and reusable protocol implementations.

The framework's design embraces a state machine model, supporting event-driven execution through messages, timers, requests/replies, and notifications. Recognizing the multi-core capabilities of modern devices, Yggdrasil promotes parallel execution of protocols while abstracting concurrency complexities. It also supports dynamic protocol management, allowing runtime activation, deactivation, or switching to adapt to varying network conditions. Moreover, Yggdrasil includes debugging and validation tools, simplifying experimental setups, and fostering protocol testing across devices without relying on a centralized control infrastructure.

Architecturally, Yggdrasil comprises a Runtime core and a Low-Level library. The Runtime manages high-level abstractions, Application Programming Interfaces (APIs), and execution logic, while the Low-Level library handles device communication via a channel abstraction. This separation ensures modularity and facilitates adaptation to diverse environments.

Yggdrasil includes implementations of core protocols essential for its operation. The Dispatcher protocol handles message serialization, network transmission and reception, supporting features like ignore lists for selective communication.

The experimental evaluation conducted by the authors highlights the use of Yggdrasil to develop and test distributed protocols in wireless ad hoc networks. Of particular relevance is the Topology Control protocol, which enables dynamic configuration of logical multi-hop topologies in dense deployments. By leveraging Yggdrasil's abstractions, the protocol reads configuration files to establish Media Access Control (MAC) address mappings and defines communication links among nodes. Through interaction with the Dispatcher protocol, it filters messages from non-target nodes to enforce a static topology. Furthermore, it supports dynamic topology adjustments by simulating link failures and recoveries via real-time requests.

### 2.2.4 Babel

Babel [70] is a Java-based [45] framework designed to simplify the development and execution of distributed protocols and applications. Babel employs an event-driven programming model that allows developers to focus on the core logic of distributed protocols while abstracting away many complex low-level details. The framework ensures efficient prototyping and production-ready implementations, offering competitive performance without significant overhead. By enabling developers to define networking components independently of underlying communication mechanisms, Babel supports a wide range of use cases, such as peer-to-peer networks [36, 124, 210] and consensus protocols [130, 174]. Notably, the framework is built to support real-world deployment, making it a practical alternative to simulators, which often fail to accurately replicate real execution environments and require substantial implementation effort (sometimes the need for two completely different codebases). On the other hand, Babel can also be used as an educational tool, tailored for advanced distributed algorithms courses (analogous to Netkit [184]).

Babel's architecture consists in three key components: protocols, the core, and networking channels. Protocols are implemented as state machines that process events through dedicated queues, enabling independent and concurrent execution. Developers define callbacks for handling events like timers, channel notifications, and network messages, allowing them to focus on the business logic. The Babel core orchestrates all interactions within a process, mediating both intra-process communication and message delivery across processes. This central coordination ensures that timers and protocol interactions are managed transparently and efficiently — with minimal intervention from the developer.

Babel's API is categorized into three functional areas: networking, timers, and inter-protocol communication. Networking in Babel is abstracted through channels (built on top of the Netty framework [153]), which encapsulate complexities such as connection management and message delivery. It includes several predefined channels, such as Transmission Control Protocol (TCP)-based channels and failure detection mechanisms, and allows developers to create custom channels to match their specific requirements. These abstractions provide flexibility and reusability, enabling protocols to operate seamlessly across diverse network configurations.

Timers in Babel support periodic or one-time actions, crucial for distributed protocols. Developers can define their own timers by simply extending a generic class (`ProtoTimer`) and then defining all necessary fields and logic. Callbacks are registered to handle timer expirations, and methods like `setupTimer` and `setupPeriodicTimer` are used to activate timers. Timers can also be canceled, providing flexibility in managing protocol timing requirements. Inter-protocol communication allows concurrent protocols within the same process to interact effectively. This includes mechanisms for one-to-one requests and replies or one-to-many notifications. Developers define communication primitives by extending generic classes like `ProtoRequest` or `ProtoNotification` and registering callbacks to process these interactions.

The utility of the Babel framework was demonstrated by the authors through two case studies: a peer-to-peer (P2P) application and a state machine replication (SMR) application [25]. The P2P application integrates HyParView [136] for overlay management and Flood Dissemination for broadcasting, showcasing Babel's ability to reduce development complexity through features like event handlers and inter-protocol notifications. The SMR application implements MultiPaxos [239, 59] with both classic and distinguished learner variants, highlighting Babel's abstraction of execution support (e.g., threading, messaging, timers) to minimize boilerplate code. Compared to other Java implementations, such as the simple but inefficient MyPaxos [255] and the robust yet complex WPaxos [252], Babel delivers equivalent functionality with significantly less code. Performance evaluations confirm Babel's reliability and fault tolerance in the P2P application under network failures and its competitive performance in SMR scenarios.

Overall, Babel emerges as a powerful tool for developing distributed protocols, offering modularity, ease of use, and performance optimization through its comprehensive API and abstractions.

### 2.2.5 Erlang-OTP

Erlang-OTP (Open Telecom Platform) [15] is a programming platform and runtime system designed specifically for building highly concurrent, fault-tolerant, and distributed applications. Initially developed by Ericsson [62] for telecommunication systems, it has gained widespread use in domains requiring high availability and scalability. The platform provides a simple yet powerful set of tools and libraries for handling concurrency through lightweight processes, message passing for inter-process communication, and robust error handling through supervision trees. These features enable developers to create systems capable of achieving near-zero downtime and handling millions of concurrent connections. However, Erlang's unique syntax and functional programming paradigm can be challenging for developers accustomed to imperative or object-oriented programming languages. Additionally, Erlang was designed for soft real-time systems and thus it is not adequate for mission-critical systems.

### 2.2.6 Akka

Akka [84] is a popular actor-based concurrency framework that provides a powerful toolkit for building scalable and resilient distributed systems in the Java Virtual Machine (JVM) [175]. It is designed around the Actor Model [3], where actors are the fundamental units of computation, handling tasks asynchronously and communicating through message passing.

One of Akka's key strengths is its ability to simplify the development of concurrent and parallel applications by abstracting low-level thread management and enabling high scalability through features like actor supervision, fault tolerance, and location transparency. However, its steep learning curve, coupled with the potential for subtle concurrency issues, can pose challenges. Additionally, while Akka's abstraction layers help with scalability, they may introduce performance overheads, particularly in highly performance-sensitive applications.

## 2.3 Network Simulation and Emulation

Network simulation and emulation are critical tools for studying, testing, and evaluating distributed systems, particularly in scenarios where real-world experimentation is impractical or unfeasible. While both methodologies aim to replicate the behavior of networks, they differ significantly in their approaches and applications. Network simulation models the behavior of a network using software abstractions, enabling the study of theoretical scenarios under controlled conditions. It allows researchers to test network protocols, configurations, and topologies at scale without the need for physical hardware. However, simulations often rely on idealized assumptions and abstracted models, which can lead to discrepancies between simulated results and real-world behavior, particularly in the presence of complex or unpredictable conditions.

In contrast, network emulation seeks to provide a closer approximation to real-world behavior by integrating live network traffic with virtualized or physical network components. Emulation tools interact with real systems and applications, making them more suitable for evaluating performance in realistic environments. This capability is particularly advantageous for validating distributed systems and dynamic network protocols, as it captures the nuanced effects of real-world constraints, such as latency, packet loss, jitter, and network failures. Despite these strengths, emulation tools typically require more computational resources and hardware support, which can limit their scalability and flexibility compared to simulation.

The following sections are organized as follows. First, we address network simulation tools (Section 2.3.1) and analyze some of the most relevant tools in this context, in particular NS-2 (Section 2.3.2), OMNeT++ (Section 2.3.3), NS-3 (Section 2.3.4), PeerSim (Section 2.3.5), and CloudSim Plus (Section 2.3.6). In Section 2.3.7, we discuss the main takeaways from the network simulation tools studied and explain why they fail to address the challenges posed by highly dynamic network topologies. Next, we shift our focus to network emulation tools (Section 2.3.8), where we explore different solutions, namely NIST Net (Section 2.3.9), NetKit (Section 2.3.10), NetEm (Section 2.3.11), CORE (Section 2.3.12), DOCKEMU (Section 2.3.13), Kathará (Section 2.3.14), and Kollaps (Section 2.3.15).

### 2.3.1 Network Simulation Tools

While tools such as PeerSim [166] and NS-2/NS-3 [101, 197] are useful for initial explorations and large-scale theoretical analysis, they are not adequate for scenarios requiring high fidelity and real-world accuracy. Their reliance on simplified models can fail to capture the intricacies of real-world network dynamics, making them less effective for testing systems designed to operate under variable or unpredictable conditions.

This limitation becomes particularly evident when transitioning from theoretical contexts to practical deployments, where accuracy and realism are paramount. Network simulators often abstract critical aspects of system behavior, such as communication overhead and node interaction constraints. Moreover, the process of implementing protocols within these tools can be both labor-intensive and error-prone, with some frameworks requiring nearly as much effort as a full-scale deployment. This inability to faithfully replicate real-world dynamics underscores the need for alternative approaches, such as network emulation tools, which offer higher fidelity and practical relevance. That being said, in the following, we explore some of these tools.

### 2.3.2 NS-2

Network Simulator 2 (NS-2) [232, 101] is a widely recognized open-source discrete event network simulator primarily utilized for research and educational purposes in the domain of computer networking. Its popularity stems from its ability to model complex network protocols, architectures, and topologies with a decent degree of accuracy and flexibility.

NS-2 is a discrete event simulator that processes events sequentially to model network behavior over time. Written in C++ [220] for performance with an OTcl [233] scripting interface for dynamic configuration, it employs an object-oriented architecture to modularize network entities like nodes, links, protocols, and agents. This design enables users to build custom network models by combining and configuring components. NS-2's link model simulates delays, bandwidth, and packet management using queues and tracing mechanisms, while visualization tools like the Network Animator (NAM) [231] and trace files support analysis and performance evaluation.

The tool's widespread adoption in academia and research stems from key advantages: its modular design supports rapid prototyping of custom agents and protocols; reproducibility is ensured through extensive protocol support (e.g., Transmission Control Protocol (TCP) [41], User Datagram Protocol (UDP) [187], Dynamic Source Routing (DSR) [107], and Ad Hoc On-Demand Distance Vector (AODV) [183]) and standards like IEEE (Institute of Electrical and Electronics Engineers) 802.11 [92]; and a large user community with rich documentation facilitates accessibility. However, NS-2 has limitations: its discrete event model incurs performance overhead in large-scale simulations; a steep learning curve arises from the dual use of C++ and OTcl; the framework has lacked updates since 2011, limiting support for modern technologies; and NAM's basic visualization lacks advanced features found in newer tools.

In conclusion, NS-2 remains a valuable tool for researchers due to its extensibility and comprehensive feature set. However, its limitations in performance, usability, and support for modern technologies highlight the need for careful consideration when selecting it for research purposes. Contemporary simulators, such as NS-3 [197] and OMNeT++ [240], address many of these issues while building on the foundational principles established by NS-2.

### 2.3.3 OMNeT++

OMNeT++ [240] is a prominent open-source, modular, and fully programmable discrete event simulation system designed specifically for computer networks and distributed systems. Developed in 1997, OMNeT++ has gained widespread adoption in both academic and research communities due to its flexibility and usability.

The tool was designed with key principles that emphasize hierarchical and reusable components, robust debugging and traceability, modularity, customizability, and open data interfaces. Its modular design enables the development of large-scale simulations, while its debugging and traceability features make it particularly suitable for educational environments. Furthermore, the system's flexibility allows it to be embedded into larger applications and network planning tools, making it a versatile solution for both research and practical applications.

OMNeT++'s core architecture consists of simple modules, written in C++, that perform specific functions, and compound modules, that combine multiple modules into hierarchical structures for complex simulations. Modules communicate through message passing via gates and connections with properties like propagation delay or data rate, supporting dynamic topology changes. Parameters configure modules with random values, expressions, or user input. The Network Description (NED) language defines module structures and interconnections declaratively, supporting hierarchical, reusable models and topologies like ring, grid, or star. NED integrates with Extensible Markup Language (XML) [251] for external system compatibility and data import/export. External model libraries provide reusable protocols like Transmission Control Protocol (TCP), Fiber Distributed Data Interface (FDDI) [202], Ethernet [217], and Global System for Mobile Communications (GSM) [167].

The tool offers two programming models for simple modules: a coroutine-based model, where modules have their own threads and use loops for message handling, and an event-driven model, which is memory-efficient and ideal for large-scale simulations. It supports multi-stage initialization and finalization for complex setups and results preservation. The platform includes an object library to track ownership and prevent memory errors, along with statistical tools, random number generation, and user-defined distributions. OMNeT++ supports both Graphical User Interface (GUI) and command-line modes, with Tkenv offering visual debugging and automatic animation, and the command-line mode enabling scalable batch simulations. Its clear separation of topology design and simulation logic, along with integrated debugging and animation features, makes OMNeT++ more user-friendly than tools like OPNET [42] and NS-2/NS-3.

In summary, OMNeT++ combines modularity, ease of use, and robust debugging tools, making it an invaluable tool for simulating computer networks, distributed systems, and also parallel systems. Its ability to support dynamic modeling, hierarchical structures, and diverse scenarios underscores its suitability for advanced research and educational applications.

### 2.3.4 NS-3

Network Simulator 3 (NS-3) [197] is a highly versatile and scalable tool designed to address the increasing complexity of network research. As the successor to NS-2, it offers substantial advancements in realism, usability, and maintainability, making it a preferred choice for simulating networking protocols and communication methods in academic and practical applications. The tool's ability to provide reproducible and adaptable simulations positions it as an essential complement to physical testbeds, especially for evaluating large-scale or highly dynamic network environments.

NS-3 was developed to address NS-2's limitations, focusing on realism and ease of use. Written entirely in C++ with an optional Python [69] API, it avoids NS-2's hybrid OTcl/C++ complexity and enables simulations that closely mirror real-world implementations. NS-3 integrates features from frameworks like GTNetS [196] and YANS [128], supports interoperability with external tools and simplifies development through rigorous maintenance standards. Its architecture abstracts components like nodes, devices, channels, and protocols, offering both low-level APIs for customization and high-level APIs for streamlined simulations. With tools for trace analysis and random variable generation, NS-3 enables detailed performance evaluations, including metrics like packet loss and queue sizes.

The tool offers advanced features like object aggregation and smart memory management for flexibility and efficiency. Nodes can dynamically add components, such as location data or IPv6 [96] support, as needed, while smart pointers prevent memory leaks through reference counting. Distributed simulation supports large-scale networks by partitioning workloads and using ghost nodes to maintain topology awareness. NS-3 bridges simulation and real-world systems with emulation capabilities, integration with testbeds like ORBIT [193], and hybrid models combining real devices with simulated networks. Its tracing framework enables detailed analysis using standardized formats like pcap [213] for tools like Wireshark [23], relational databases, and statistical libraries. Visualization tools provide animation trace files for intuitive insights into network behavior.

In summary, NS-3 marks a clear advancement over its predecessor, NS-2, by addressing many of its limitations. With its emphasis on flexibility, scalability, and some emulation features, NS-3 provides a robust framework for simulating a wide range of networking scenarios, including dynamic and large-scale environments.

### 2.3.5 PeerSim

PeerSim [166] is a highly scalable and versatile simulation framework specifically designed for research in peer-to-peer (P2P) systems. Its primary goal is to address the challenges of scalability and dynamism inherent to P2P protocols while providing a platform for their evaluation in simulated environments. Real-world testing of P2P protocols is often unfeasible due to its high costs and the difficulty of reproducing realistic conditions.

PeerSim offers a simulation environment that can model dynamic scenarios, such as churn and network failures, which are critical for studying P2P systems.

PeerSim's modular architecture models the network as a list of nodes, each with a list of protocols. Initializers set up the network before simulation, while controls modify or monitor components during runtime, enabling tasks like adding nodes or tracking metrics. PeerSim offers two engines: a scalable cycle-based engine for abstract simulations with millions of nodes, and an event-based engine for detailed transport-layer simulations using datasets like King [82]. Both support churn simulation for studying P2P robustness, though cycle-based protocols are compatible with the event-based engine, not vice versa. Optimized for memory efficiency, PeerSim handles simulations from small to large scales, with customizable components via configuration files.

PeerSim's graph abstraction capabilities are particularly noteworthy for overlay network research. The simulator treats overlay networks as graphs, providing initializers for generating common topologies such as random graphs and small-world networks. Observers can analyze graph properties such as diameter, clustering, and connectivity. The simulation results can also be exported in popular graph formats for further analysis or visualization. For researchers requiring additional mathematical analysis, the vector abstraction module offers tools for manipulating protocol instances as vectors, enabling operations like vector initialization and angle calculations.

The tool's configuration system is user-friendly, employing plain text files similar to Java property files. These files define components, parameters, and simulation engines, ensuring reproducibility and simplifying the process of sharing experiments. The Java-based implementation leverages the runtime configuration and dynamic class loading, allowing seamless integration of custom components.

In summary, PeerSim provides a scalable, modular, and configurable simulation environment that addresses the unique challenges of P2P systems. Its ability to simulate dynamic scenarios, combined with its support for graph and vector abstractions, makes it an invaluable tool for advancing research in this domain.

### 2.3.6 CloudSim Plus

CloudSim Plus [214] is a state-of-the-art simulation framework specifically designed to address the challenges inherent in cloud computing research. Cloud computing, characterized by on-demand resource provisioning, requires several services such as load balancing, energy efficiency optimization, and Service Level Agreement (SLA) management. However, conducting real-world experiments in cloud environments is often prohibitively costly and complex. This has led researchers to adopt simulation tools like CloudSim [34] to model and analyze cloud computing systems. Despite its popularity, CloudSim suffers from several limitations, including inadequate documentation, code duplication, and poor adherence to software engineering principles. To address these shortcomings, CloudSim Plus was developed as a fork of CloudSim, with an emphasis on modularity, extensibility, and correctness.

CloudSim Plus addresses CloudSim's limitations, such as poor documentation, limited testing, and lack of adherence to modern design principles, by enhancing reusability, modularity, and extensibility. It improves class hierarchy, introduces better documentation and integration tests, and ensures a clear separation of concerns. Built for simulating cloud layers from Infrastructure-as-a-Service (IaaS) to Software-as-a-Service (SaaS), CloudSim Plus models physical resources (datacenters, hosts, networks) and logical resources (Storage Area Networks (SANs), Virtual Machines (VMs), topologies). Key advancements include modular package structures, power and network-aware components, and advanced schedulers and allocation policies, such as Simulated Annealing [238] for resource mapping, enabling efficient testing and validation of cloud architectures.

CloudSim Plus supports dynamic VM and cloudlet arrivals, allowing brokers to manage submissions for more realistic simulations. Enhanced brokers offer custom allocation policies and heuristics, while event listeners track simulation events. Quality assurance is prioritized with continuous integration, code analysis tools, and automated testing, achieving higher code coverage (35%) compared to CloudSim (10%-18%). CloudSim Plus simplifies simulations by allowing developers to define and configure components like datacenters, brokers, VMs, and cloudlets, then execute simulations and generate detailed results on resource usage and execution timing, aiding algorithm and model evaluation.

In conclusion, CloudSim Plus represents a significant improvement over its predecessor by adhering to best practices in software engineering, avoiding code duplication, and providing an extensible and well-documented framework. It is a valuable tool for researchers aiming to develop and evaluate algorithms that address various challenges in cloud computing.

### 2.3.7 Discussion

The presented network simulation tools are invaluable platforms for modeling and analyzing a wide range of networking scenarios, however they still fall short in addressing specific challenges presented by highly dynamic environments with intermittent connectivity, unpredictable communication patterns, and selective communication requirements. These environments, such as satellite swarms and mobile ad hoc networks (MANETs), which are central to the work planned in this project, demand higher fidelity and adaptability than most network simulators can provide.

Simulation tools, like the ones previously studied, offer modularity, scalability, and rich feature sets for a variety of use cases, including traditional networking protocols, peer-to-peer (P2P) systems, and cloud computing architectures. However, their design paradigms inherently rely on abstracted models that simplify network behavior to achieve computational efficiency and scalability. While effective for theoretical explorations and controlled experimental setups, these abstractions often fail to accurately capture the real-world complexities encountered in dynamic and volatile environments. For example, event-driven models in NS-3 [197] and OMNeT++ [240] and the cycle-based engine of PeerSim [166] are well-suited for static or semi-dynamic conditions, but they struggle to reflect transient network failures and irregular communication patterns.

Moreover, the static nature of many simulation frameworks limits their ability to accommodate selective communication patterns, where nodes engage in communication based on contextual and application-specific criteria. Such scenarios require dynamic reconfiguration capabilities, adaptive protocol behaviors, and nuanced representation of communication overheads and failures. These aspects are either omitted or simplified in most simulation tools, making their applicability to systems operating under unpredictable conditions inadequate. For instance, while NS-3 offers some support for emulation and hybrid setups, the complexity of integration with real-world systems introduces practical constraints that may render such approaches unfeasible in rapidly changing environments.

Another limitation lies in the tool's handling of intermittent connectivity. While mechanisms such as churn simulation in PeerSim or dynamic topology changes in OMNeT++ provide partial support for modeling disconnections, they do not address the intricate challenges posed by highly unstable links and irregular availability. Effective solutions in such contexts demand frameworks capable of real-time adaptation to shifting connectivity patterns and selective engagement with network participants. These capabilities are beyond the scope of the surveyed tools, as they are designed to operate within well-defined and relatively predictable simulation models.

In light of these limitations, network simulation tools cannot fully address the challenges posed by environments with extreme dynamism and selective communication requirements. These gaps emphasize the need for alternative approaches, such as decentralized and adaptive systems, to model and manage dynamic membership and communication patterns effectively. Ultimately, while network simulation provides a valuable foundation for understanding theoretical behaviors, it must be complemented by novel approaches to address the demands of highly dynamic and unpredictable networking environments.

### 2.3.8 Network Emulation Tools

Due to the limitations discussed in the previous section, network emulators are often considered a more suitable option for testing distributed applications. Unlike simulators, emulators provide a closer approximation of real-world network conditions by allowing the actual protocols and application code to run on an emulated network infrastructure. This higher fidelity enables more accurate evaluations of system behavior under dynamic and unpredictable network conditions. In the following, we explore some of the most relevant network emulation tools, highlighting their features and main use cases.

### 2.3.9 NIST Net

NIST Net [38] is a Linux-based [119] network emulation tool designed to address the challenges of testing network protocols and distributed applications in increasingly complex and dynamic environments. Developed as a commodity router, it operates on live Internet Protocol (IP) packets and simulates a wide range of network conditions, including packet delay and jitter, bandwidth limitations, congestion, packet loss, and duplication. The tool's ability to model diverse scenarios, such as satellite delays or asymmetric links, makes it particularly suitable for testing adaptive protocols under reproducible conditions.

The primary strength of NIST Net lies in its hybrid semi-synthetic approach, combining the reproducibility of simulation with the realism of live testing. This design allows real code to run in a real network environment while introducing synthetic delays and faults. Its dynamic performance modeling makes it ideal for evaluating adaptive mechanisms that static methods cannot address.

NIST Net emulates network effects with precision, including configurable packet delays (fixed or random) with adjustable statistical parameters (mean, standard deviation, correlation). Packet loss and duplication probabilities are fine-tuned, and congestion-related loss is modeled using Derivative Random Drop (DRD) [141], supporting mechanisms like Explicit Congestion Notification (ECN) [66]. Bandwidth limits are applied through theoretical transmission delays, with additional controls like queue length limits for cumulative delay modeling.

NIST Net's modular architecture includes a kernel module and user interfaces. The loadable kernel module allows runtime updates without disrupting connections, integrating hooks into the IP packet handler to intercept and process packets based on emulator entries. A high-resolution timer enables fine-grained delay scheduling, supporting real-time updates and concurrent control by multiple processes.

Emulator entries are managed with a two-level hash table, enabling sub-microsecond lookup times even for thousands of active entries. Statistical routines introduce impairments like delays, using pre-generated inverse CDF (Cumulative Distribution Function) tables for efficient pattern approximation.

Basic delay generation captures short-term correlations, while advanced features like the Multifractal Wavelet Model (MWM) [195] and long-term analyzers provide an accurate representation of medium and long-term delay patterns, essential for high-fidelity emulation of extended network dynamics.

NIST Net has been rigorously tested, showing reliable performance at high data rates, including gigabit Ethernet speeds. Its optimized lookup operations and delay scheduling introduce minimal overhead. However, at higher traffic rates (e.g., 100 Mbit/s to 200 Mbit/s), timer queue management impacts performance, increasing latency and bandwidth demands.

Compared to other emulators like Dummynet [198] and the Ohio Network Emulator [8], NIST Net stands out with advanced statistical modeling, high-speed operation, and a user-friendly design. While Dummynet excels in queuing models, it lacks NIST Net's precision in delay modeling. Its modular architecture and ability to handle high data rates make NIST Net ideal for academic research, teaching, and adaptive protocol development.

In summary, NIST Net provides a robust and flexible framework for emulating diverse network conditions due to its hybrid approach that combines simulation with emulation.

### 2.3.10 NetKit

Netkit [184] is a network emulation tool designed for education, providing a flexible platform for replicating real-world network environments. It leverages User-Mode Linux (UML) [176] for lightweight virtualization, allowing users to emulate complex networks with minimal resources. The tool consists of two layers: the emulation engine for networking features like Virtual Local Area Networks (VLANs) [76] and Multiprotocol Label Switching (MPLS) [200], and the core layer that simplifies user interaction. Netkit's design enables the creation and sharing of reusable "labs" or pre-configured scenarios, making it ideal for hands-on learning and academic integration.

The framework has had a significant educational impact, being adopted by over 15 institutions for networking courses, exams, and group projects. Its pre-configured labs cover topics from basic protocols (Address Resolution Protocol (ARP) [185], Routing Information Protocol (RIP) [87]) to advanced concepts (Open Shortest Path First (OSPF) [168], Border Gateway Protocol (BGP) [144], MPLS). Students gain practical experience in network configuration and troubleshooting using tools like tcpdump [80], Quagga [188], and Apache Server [68]. Netkit is praised for its ease of use and performance on standard hardware. Unlike GUI-heavy tools like Graphical NS-3 (GNS3) [229], Netkit's text-based configuration aligns with real-world practices, teaching students to engage with networking syntax. Its modular design ensures compatibility across Linux distributions, offering a lightweight, extendable platform for networking education.

Netkit balances usability with sophistication, unlike tools like MiniNet [131] and Virtual Networks over Linux (VNX) [230], which have verbose configurations and dependency management issues, or NS-3, which lacks interactivity. Its lightweight, interactive design and support for various networking technologies make it ideal for academic and research settings.

Developed at Roma Tre University [237] and evolved into an open-source project, Netkit's community-driven development ensures sustainability. However, its text-based configuration can be challenging for non-Linux users, and it lacks native performance measurement and hardware-specific features. Proposed enhancements include Software-Defined Networking (SDN) [253] support, topology generators, and expanded labs to meet evolving needs.

In conclusion, Netkit represents a robust and effective tool for network emulation, particularly within educational contexts. Its lightweight design, emphasis on usability, and alignment with real-world practices make it an invaluable resource for teaching networking concepts and facilitating hands-on experimentation. While primarily developed with educational purposes in mind, the tool's features extend beyond this domain, proving equally relevant for a variety of other use cases, including research and practical network testing.

### 2.3.11 NetEm

NetEm [90] aims at enhancing Linux's traffic control [9], enabling the emulation of network conditions like delay and packet loss. Built on Linux's QoS [191] and Differentiated Services (DiffServ) infrastructure [10], NetEm offers a software-based alternative for network emulation, making it valuable for testing protocols and applications in real-world conditions. By emulating Wide Area Network (WAN) conditions on Local Area Networks (LANs), it bridges the gap between controlled environments and the complexities of real-world networking.

NetEm was initially used to evaluate new TCP enhancements in the Linux 2.6 kernel [216], focusing on improving TCP performance over high-speed networks. It has since been employed to test various TCP congestion control algorithms, such as TCP Vegas [31], TCP Westwood+ [53], and Binary Increase Congestion (BIC) TCP [254], addressing issues like bandwidth estimation and congestion window adjustments. Early experiments with tools like iperf [95] provided insights into these algorithms performance and limitations, highlighting NetEm's value in protocol testing.

A key feature of NetEm is its role as a queuing discipline in Linux [212]. It operates with two queues: a private holding queue for packets awaiting transmission and a nested FIFO (First In, First Out) queue for transmission. Packets are timestamped and managed based on send times, enabling precise network behavior emulation. Configuration of network parameters is managed via the `tc` command, enabling the simulation of various network conditions:

- **Packet Delay:** Supports constant/variable delays with configurable averages, standard deviations, and distribution types (e.g., normal, Pareto).
- **Packet Loss:** Randomly drops packets based on a user-defined percentage, with correlation parameters for realistic loss patterns.
- **Packet Duplication:** Simulates redundant routes by duplicating packets at user-defined percentages.
- **Packet Reordering:** Introduces a delay to reorder packets, though its application is limited to ungapped packets.
- **Rate Control:** Enables bandwidth limitation, though timer granularity in Linux limits precise control on high-speed links.

NetEm, integrated within the Linux kernel and iproute2 package [100], is easily accessible with a web-based GUI. However, it has limitations, particularly in scenarios requiring high precision or complex interactions. The kernel timer granularity (1 ms) limits the accuracy of high-speed link emulations. Additionally, the use of pseudo-random number generators (PRNGs) can introduce patterns, affecting simulation fidelity. A more advanced generator, like Tausworthe [228], could improve randomness.

NetEm extends beyond basic testing by integrating with tools like iperf for performance assessment. Compared to Dummynet [198] and NIST Net [38], NetEm's deep integration with Linux's queuing discipline makes it a preferred choice. However, it cannot replicate the full complexity of the Internet [179], and limitations in network driver capabilities may impact performance under high load.

In summary, NetEm is a powerful tool for network emulation, offering extensive configuration and seamless integration within the Linux ecosystem. The main bottleneck stems from the precision of the Linux kernel timers (usually bounded at 1 ms).

### 2.3.12 CORE

The Common Open Research Emulator (CORE) [5] is an open-source network emulation tool leveraging lightweight virtualization technologies like FreeBSD jails [111, 189], Linux OpenVZ containers [71], and namespaces [201]. It enables protocol and application execution without modification, functioning as a "black box" emulator where users and systems remain unaware of its virtualized nature. CORE's live emulation supports integration with real-world networks and hardware-in-the-loop setups, making it ideal for testbeds.

CORE's architecture includes a GUI for designing network topologies, a services layer for instantiating virtual nodes, and a custom API for communication between components. Optimized for emulating tens of nodes on a single machine, it focuses on Open System Interconnection (OSI) [126] layers 3 and above, supporting basic link characteristics like bandwidth, delay, and error rate. For more control, CORE can integrate with tools like EMANE [6] and NS-3. The framework is built on the IMUNES emulator [259], using its Tcl/Tk-based [249] GUI for the interface.

As discussed previously, simulation tools abstract operating systems and protocols into a model for statistical analysis, while network emulators interact with real systems and may need specialized hardware. CORE combines both approaches, emulating network stacks and simulating link behavior, allowing live applications to run under realistic conditions while maintaining scalability and affordability.

CORE employs para-virtualization techniques, isolating processes in jails or namespaces and associating them with virtual network stacks, sharing hardware resources for scalability. It achieves high-fidelity network layer emulation, using simplified models for the data-link and physical layers, relying on Netgraph [171] in FreeBSD and NetEm [90] in Linux. For advanced modeling, CORE integrates with tools like EMANE for MAC

(Media Access Control) and PHY (Physical) models. CORE has two operational modes: editing, for designing network topologies via the GUI, and execution, where the virtual network is instantiated and nodes are accessed through an interactive shell. Configuration and mobility scripts enable dynamic deployment and node movement. CORE's design supports installing real network interfaces into virtual nodes. It integrates seamlessly with EMANE or NS-3, using EMANE's pluggable architecture for OSI layers 2 and below and facilitating distributed emulations across multiple servers with TUN/TAP interfaces [236], enhancing scalability.

Performance evaluation carried out previously [7, 5] shows CORE's efficiency, especially with FreeBSD jails, which outperform Linux platforms in throughput, CPU utilization, and latency. The Netgraph subsystem's resource handling boosts FreeBSD's performance. Among Linux platforms, namespaces offer better CPU (Central Processing Unit) efficiency than OpenVZ containers, highlighting CORE's ability to handle large-scale emulations effectively.

In summary, CORE's robust virtualization-based architecture, combined with its support for real-time emulation and integration with advanced tools like EMANE, positions it as a versatile and powerful network emulation framework. Its lightweight design, scalability, and adaptability make it a valuable tool for simulating complex network scenarios and evaluating protocol performance in dynamic environments.

### 2.3.13 DOCKEMU

DOCKEMU [235] is a network emulation tool that simplifies setting up and managing network experiments. By leveraging modern virtualization technologies, it addresses challenges in configuring scenarios while ensuring accuracy and realism.

DOCKEMU's architecture integrates Docker-based [158] Linux containers, Linux network bridges, and the NS-3 network simulator to emulate both wired and wireless networks, including MANETs and wireless ad hoc networks (WANETs). The use of Docker containers allows the tool to support real-world operating systems and applications, eliminating the need for separate implementations in simulated and real environments. This reduces development time and improves experimental accuracy by enabling direct deployment of applications and prototypes.

The tool's design is based on three key principles of network emulation: realism, reproducibility, and representativeness. Realism is achieved by closely mimicking real-world environments to minimize discrepancies with actual network behavior. Reproducibility ensures consistent replication of scenarios for comparative studies, while representativeness guarantees the inclusion of critical factors affecting network performance.

DOCKEMU relies on Linux containers (LXC) [103] for lightweight virtualization, partitioning the host Operating System (OS) into isolated containers and enabling scalability to hundreds or thousands of nodes. Compared to traditional virtualization platforms like VirtualBox [40] and VMware [247], LXC is more resource-efficient, virtualizing only the necessary components. Docker enhances LXC with features like portability, versioning, and shared libraries, streamlining the experimental workflow. Linux network bridges act as virtual switches, connecting physical and virtual interfaces, while NS-3 simulates layers 1 and 2 of the OSI model, enabling seamless interaction with virtual environments. DOCKEMU supports IPv4 [186] and IPv6, ensuring future relevance, and simplifies installation and configuration, addressing inefficiencies in traditional research workflows by enabling real-world experiments without redundant transitions.

Users configure scenarios via a text file (dockemu.conf), specifying variables like node count, network configurations, and routing protocols. DOCKEMU automatically sets up nodes and networking layers, including Linux bridges [241], tap interfaces, and NS-3 components. Compared to similar tools, DOCKEMU excels in flexibility and scalability. Unlike NEMAN [190], which relies on NS-2 and supports only Ethernet networks, DOCKEMU uses NS-3 to support both wired and wireless communication. The integration of NS-3's real-time simulation with Docker-based virtualization ensures high-fidelity emulation for large-scale experiments.

Overall, DOCKEMU addresses critical limitations in traditional network simulation and emulation tools, such as reproducibility issues in MANET simulations and scalability constraints. By combining the strengths of Linux containers, Docker, and NS-3, DOCKEMU offers a comprehensive solution for network research, enabling researchers to achieve realistic and representative results with minimal setup effort.

### 2.3.14 Kathará

Kathará [30] takes advantage of Network Function Virtualization (NFV) [161] and Software-Defined Networking (SDN) by using container technology for high performance, scalability, and minimal resource usage. Unlike traditional NFV solutions based on virtual machines (VMs), Kathará uses containers, which offer lower overhead and faster provisioning. It can support up to 300 network nodes, far exceeding the 40 nodes managed by VM-based systems like User Mode Linux (UML). Kathará's efficiency in memory and CPU usage makes it ideal for large-scale, resource-constrained environments.

Kathará stands out with its support for programmable data-plane capabilities through P4-compliant [32] switches and OpenFlow [155]. P4 enables advanced packet-forwarding rule programmability, allowing network devices to perform complex tasks with performance akin to hardware. The tool's architecture includes three modules: Scripts, Operations, and the Container Platform Adapter. These modules enable users to define network configurations and manage resource allocations via a simple interface, promoting rapid deployment and scaling. Kathará also supports Service Function Chains (SFCs) [85] for grouping VNFs to optimize data-plane performance.

Kathará enhances security by using a Finite State Machine (FSM) security wrapper to filter and validate Docker commands, preventing unauthorized actions like file access or mounting, and protecting against privilege escalation. This ensures safe experimentation for non-root users in both academic and production environments. It also boasts strong performance, with low response times and linear scalability in multi-hop network scenarios. Latency increases of only 0.3 milliseconds per hop make Kathará ideal for testing and deploying VNFs and SDN applications in edge and cloud environments.

In summary, Kathará provides a lightweight, efficient, and versatile solution for network emulation. Its integration of container technology, support for programmable data planes, and robust security measures distinguish it from traditional VM-based frameworks. Future enhancements, such as orchestration features and cloud deployment support, could further solidify Kathará's position as a critical tool for advancing NFV and SDN research and deployment.

### 2.3.15 Kollaps

Kollaps [78] is a decentralized network emulation tool for large-scale distributed systems in dynamic environments. It emphasizes end-to-end network properties, such as latency, bandwidth, jitter, and packet loss, while dynamically adapting to topology changes. Unlike MiniNet [131] and MaxiNet [250], which scale poorly for resource-intensive applications, or EmuLab [91], which lacks dynamic topology support, Kollaps provides a scalable, fully distributed solution. Tools like CrystalNet [142], focusing only on control-plane emulation, are also unsuitable for data-plane scenarios, further highlighting Kollaps's unique capabilities.

Kollaps overcomes these limitations by leveraging three key design principles:

1. **End-to-End Focus:** Kollaps emulates observable network properties without modeling router and switch states. Simplified virtual end-to-end links preserve critical properties while reducing computational overhead.

2. **Decentralized Architecture:** Kollaps's decentralized control ensures scalability, avoiding centralized bottlenecks and supporting thousands of processes in distributed deployments.

3. **Dynamic Adaptability:** Kollaps allows real-time adjustments to link properties, failures, additions, and traffic changes, ensuring accurate and responsive network modeling.

Kollaps integrates with container orchestration platforms like Docker Swarm [64] and Kubernetes [146], enabling the deployment of unmodified containerized systems. Its deployment generator translates topology descriptions into plans using formats like XML and Yet Another Markup Language (YAML) [24]. The Emulation Manager (EM) and Emulation Core (EC) enforce topology constraints and manage bandwidth, while the Traffic Control Abstraction Layer (TCAL) manipulates network properties via Linux's Traffic Control system, ensuring efficient traffic shaping and bandwidth adjustments.

Kollaps supports large-scale networks with thousands of nodes by collapsing topologies into virtual links, reducing computational demands. Pre-computed topology changes enable rapid adjustments during execution, such as adding or removing links and nodes. Its application-agnostic design supports unmodified applications across programming languages and transport protocols. Additionally, it facilitates cost-effective modeling of hypothetical scenarios, like changing latencies or deployment configurations without requiring physical setups.

Kollaps stands out through scalability, decentralization, and dynamic modeling. MiniNet and MaxiNet struggle with scalability and co-located deployments, while Kollaps supports distributed deployments without centralized control. Unlike Dummynet and ModelNet [243], which require dedicated infrastructure, Kollaps operates on shared clusters. SplayNet [207] relies on a domain-specific language, whereas Kollaps supports unmodified applications and containerized environments.

Kollaps shows low error rates in bandwidth and jitter emulation, accurately replicating real-world deployments like Amazon EC2 [11]. It successfully models complex distributed systems such as Cassandra [129] and BFT-SMaRt [25] with minimal deviations. Its decentralized design ensures consistent throughput across distributed setups, even with increased container and host counts.

However, Kollaps has limitations, namely, it lacks support for interactive testing, multipath or multicast routing, and is constrained by the physical capacity of the hosting cluster. Fixed-interval bandwidth updates reduce effectiveness for short-lived flows, making it more suited for wide area networks than datacenter scenarios.

In short, Kollaps represents a significant advancement in network emulation, addressing the limitations of existing tools with a decentralized and scalable architecture.

## 2.4 Membership Abstractions

Membership abstractions are a fundamental concept in distributed systems, providing a mechanism for nodes to maintain an updated view of the system's active participants [27]. This abstraction serves as a building block for higher-level distributed algorithms, such as consensus, replication, and fault tolerance. A membership service typically manages the dynamic list of nodes in the system, supporting operations such as node addition, removal, and failure detection.

The primary goal of membership protocols is to ensure consistency [224, 211, 194, 163], scalability [194, 35, 74, 56, 248], and fault tolerance [98, 262, 242, 121] while operating in dynamic and often unreliable environments. These protocols enable nodes to achieve a coherent view of system membership despite challenges such as network partitions, high churn rates, and concurrent failures. Membership can be classified into static and dynamic approaches, with the latter being more relevant for highly dynamic systems where nodes frequently join and leave the network.

Traditional approaches to membership management often assume stable communication links and rely on centralized or hierarchical structures for decision-making [19, 18]. However, these methods face significant challenges in highly dynamic and decentralized settings, such as ad hoc networks, satellite swarms, or large-scale peer-to-peer systems. Consequently, more recent protocols increasingly emphasize decentralized designs [170], leveraging techniques like gossip-based dissemination to achieve robustness and adaptability.

In the following subsections, we explore several pertinent membership protocols that are particularly adequate for our use cases — satellite swarms and mobile ad hoc networks (MANETs). First, we cover the Hybrid Partial View (HyParView) protocol (Section 2.4.1). Next, we delve into the X-BOT abstraction (Section 2.4.2). Finally, we conclude this section with the Thicket protocol (Section 2.4.3).

### 2.4.1 HyParView

HyParView [136] is a hybrid membership protocol designed to enhance the reliability of gossip-based broadcast systems, particularly under high failure rates. Unlike traditional protocols that rely on maintaining a single partial view [55, 63, 181], HyParView introduces a dual-view architecture comprising an active view and a passive view, each serving distinct purposes. The active view, a small symmetric set of nodes, is used for gossip-based message dissemination and ensures reliable communication through the use of TCP connections. Meanwhile, the passive view, a larger set of backup nodes, supports rapid failure recovery and view management through periodic updates and shuffling.

A key feature of HyParView is its ability to maintain robust message delivery and fast recovery from node failures. By leveraging TCP for both communication and failure detection, the protocol minimizes the overhead typically associated with larger fanouts while ensuring high reliability. For instance, HyParView demonstrates recovery from failure rates as high as 90% within just four membership rounds, a significant improvement over comparable protocols such as Cyclon [246] and Scamp [75]. The dual-view approach also enables efficient failure management: failed nodes in the active view are quickly replaced by candidates from the passive view, maintaining the integrity of the system.

Experimental results in [136, 133] highlight HyParView's superior performance in large-scale and dynamic environments. Compared to other membership protocols, it exhibits lower clustering coefficients, shorter average path lengths, and a more balanced in-degree distribution, all of which contribute to its resilience under heavy failure conditions. Moreover, the protocol's design is well-suited for real-world applications, such as disaster recovery and dynamic network scenarios, where node failures are inevitable.

### 2.4.2 X-BOT

X-BOT [139, 182] is a decentralized protocol designed to optimize the efficiency of unstructured gossip overlay networks while preserving their critical properties, such as connectivity and degree stability. Unlike previous approaches, which often rely on global information or hierarchical structures [28, 54], X-BOT employs a fully decentralized optimization process based on local information. This design enables it to improve overlay performance in metrics such as latency or cost without compromising resilience, scalability, or the inherent randomness of unstructured overlays.

The protocol builds upon the previously discussed HyParView protocol, which consists of a small active view for reliable dissemination and a larger passive view for fault tolerance. X-BOT dynamically enhances this model by periodically replacing nodes in the active view with better candidates from the passive view, guided by an oracle that assesses link costs (e.g., latency or Internet Service Provider (ISP) proximity). To maintain the essential properties of random overlays, such as low clustering and connectivity, X-BOT ensures a mix of biased (optimized) and unbiased (random) neighbors in each node's active view, controlled by a protocol parameter ($\mu$). The optimization process is carried out through a lightweight, coordinated interaction between four nodes: the initiator, a candidate for replacement, and their respective neighbors. This process involves a minimal overhead of seven message exchanges per optimization round, making X-BOT both efficient and scalable.

Experimental evaluations reported in [139, 182] against protocols like GoCast [226], Araneola [156], and T-Man [105] have demonstrated X-BOT's superiority in reducing overlay costs while achieving balanced clustering coefficients and maintaining shorter path lengths without sacrificing connectivity or fault tolerance.

### 2.4.3 Thicket

Thicket [65] is a decentralized protocol for constructing and maintaining multiple spanning trees in P2P overlay networks. It addresses critical challenges in traditional tree-based dissemination methods, including imbalanced load distribution and vulnerability to single-node failures. Unlike single-tree approaches, where interior nodes bear disproportionate loads and tree failures can disrupt communication, Thicket utilizes multiple trees to enhance load balancing and fault tolerance. Each node typically serves as an interior node in only one or a few trees, ensuring fair resource distribution and providing redundancy for reliability.

The protocol uses a reactive peer-sampling service to construct divergent spanning trees and maintains their integrity through dynamic repair and reconfiguration mechanisms. These features ensure minimal tree overlap, low latency, and scalability, even in large-scale systems with dynamic link changes.

Simulations with 10,000 nodes reported in [65] show that Thicket outperforms comparable protocols, such as Plumtree [135], in metrics like load distribution, reliability, and message latency. Its adaptive tree repair mechanism enables rapid recovery from node failures, maintaining connectivity and minimizing disruptions. Furthermore, Thicket achieves superior performance in both sequential and simultaneous node failure scenarios, demonstrating high broadcast reliability and effective reconfiguration under catastrophic conditions.

## 2.5 Discussion

Network simulation, as discussed previously, provides robust platforms for modeling and analyzing a variety of networking scenarios, offering modularity, scalability, and rich feature sets. However, its inherent reliance on abstracted models limits its ability to address the specific challenges posed by highly dynamic environments with intermittent connectivity, unpredictable communication patterns, and selective communication requirements. While effective for controlled experimental setups, these tools struggle to capture real-world complexities such as transient network failures, irregular communication patterns, and adaptive protocol behaviors.

Existing network emulators, while powerful, are not fully equipped to handle the unique challenges of highly dynamic environments with intermittent connectivity, unpredictable communication patterns, and selective communication.

Most emulators, such as NetEm and DOCKEMU, are designed for relatively stable networks with consistent connectivity and predictable communication patterns. They struggle to model scenarios where network links frequently change and nodes must adapt dynamically. Similarly, tools like CORE and GNS3 target static or deployment-oriented scenarios and lack the flexibility to emulate nodes frequently joining or leaving the network, as seen in satellite swarms or mobile ad hoc networks (MANETs).

Additionally, existing tools do not natively support selective communication, where nodes interact based on dynamic criteria like proximity or application-specific criteria. Besides, the high unpredictability of communication patterns in dynamic environments renders existing tools incapable of accurately and precisely modeling these network dynamics.

On the other hand, existing membership abstractions, such as HyParView, X-BOT, and Thicket, are designed for dynamic environments but still struggle with the unique challenges of highly dynamic systems with intermittent connectivity and unpredictable communication patterns. HyParView, for instance, relies on the deterministic selection of nodes and symmetric views, which assume that the overlay network remains connected — a condition that may not hold in highly dynamic and unpredictable environments where connectivity is frequently disrupted. X-BOT, on the other hand, depends exclusively on local information and the estimation of link costs, which can be particularly problematic in environments with erratic and unpredictable communication patterns. The difficulty of accurately assessing link costs in such settings undermines the effectiveness of its optimization process. Similarly, while Thicket's use of multiple spanning trees enhances fault tolerance, its tree repair and reconfiguration mechanisms, when executed multiple times, can lead to certain nodes appearing as interior nodes in more than one tree — an undesirable condition that compromises load balancing and resource distribution. This issue becomes even more pronounced in highly dynamic and unpredictable scenarios, where frequent changes in the network topology exacerbate the problem. More important, all these abstractions assume an underlay network (Internet Protocol — IP) that enables any pair of active devices to (in general) communicate, which is not true in satellite swarms or wireless networks.

The main observation is that, while existing network simulation tools, emulators, and membership abstractions provide valuable capabilities for modeling and managing network environments, they fall short when faced with the unique challenges posed by highly dynamic and unpredictable systems such as satellite swarms and MANETs. The limitations in handling intermittent connectivity, fluctuating communication patterns, and selective communication hinder their ability to accurately capture and manage the complexities of real-world scenarios in these environments. Consequently, there is a clear need for exploring alternative and complementary approaches that can better address these specific challenges, ensuring reliable operation in highly dynamic conditions.

# 3 Proposed Work

The structure of this chapter is organized into four sections. Section 3.1 provides a detailed analysis of the challenges and limitations associated with existing solutions for highly dynamic environments. Section 3.2 outlines the intuition and high-level approach for addressing these challenges, including the development of a prototype tool (emulator) and a decentralized membership service. Section 3.3 describes the methodology and metrics for testing and validating the proposed solutions. Finally, Section 3.4 presents the tasks involved in completing the dissertation and their respective timelines.

## 3.1 Problem Description

As discussed in Chapter 1, the design and implementation of robust membership services for dynamic environments remain a significant challenge in distributed systems [208]. These challenges are amplified in swarm systems, such as satellite swarms and mobile ad hoc networks (MANETs), where the highly dynamic nature of the network topology complicates the maintenance of consistent and accurate membership views [19]. The difficulties arise from several factors, including node mobility, physical separation between nodes (distance and altitude), orientation of communication hardware, energy constraints [206], and external interference from environmental conditions. These factors lead to intermittent connectivity [115, 114, 160], selective communication patterns, and unpredictable topology changes, which severely impact the effectiveness of traditional membership services.

Membership abstractions are fundamental to the operation of the vast majority of distributed protocols [21], namely resource allocation, consensus, and data replication. However, the greater part of membership services is designed for static or minimally dynamic networks, where connectivity remains relatively stable over time [19, 18]. Consequently, these services fail to adapt to the unpredictable and transient communication patterns inherent in highly dynamic systems. As a result, they face difficulties in maintaining up-to-date membership views and often incur high communication overhead to achieve a correct operation [136, 120]. Such inefficiencies are particularly problematic in environments where bandwidth is limited, or energy consumption must be minimized [206], as is the case with both satellite swarms and MANETs.

Satellite swarms, for instance, present unique challenges due to their reliance on orbital dynamics, which dictate periods of limited communication between nodes. Unlike traditional IP-based networks, satellite swarms communicate primarily through radio signals [256]. Effective communication depends on the alignment of antennas between satellites [169, 256], which may not always be possible due to their independent orbital trajectories and the relative motion of the nodes. This alignment requirement leads to intermittent connectivity, where communication windows are both brief and highly unpredictable. Furthermore, these systems face significant challenges due to limited power and bandwidth [152, 61], as strict energy budgets restrict data exchange during brief communication windows, often complicated by radio spectrum interference [108]. Latency and rapidly evolving topologies, driven by orbital mechanics and spatial distribution, further complicate real-time network adaptation. Additionally, satellites operate in harsh environments, facing risks like radiation [86] and micrometeoroids [132], which demand fault-tolerant and adaptive protocols. Besides, the decentralized nature of swarms precludes centralized management and the reliance on ground stations [178].

In a similar fashion, MANETs are decentralized and highly dynamic, facing several challenges. Frequent topology changes [125], driven by node mobility [192], disrupt communication paths and require constant route updates [164], increasing latency and packet loss [102, 93]. Environmental factors [205, 83] like obstructions, signal interference, and weather degrade link quality and reliability. Resource constraints [79], including limited battery life, processing power, and bandwidth, complicate consistent communication and protocol execution. Intermittent connectivity [113, 4], as nodes frequently join or leave, makes maintaining accurate membership views difficult. Additionally, the open nature of MANETs heightens security risks, including eavesdropping, spoofing, and denial-of-service attacks [245, 109].

Another critical limitation hindering the design of suitable membership services for these enviroments is the lack of tools and frameworks to model and emulate dynamic network behaviors accurately [19, 18]. While traditional network emulation tools can replicate some network effects, such as latency, bandwidth, jitter, and packet loss and duplication, they fail to capture the more complex and nuanced scenarios inherent to swarm systems [94]. For example, selective communication patterns, where certain nodes may only occasionally communicate with specific subsets of peers, and intermittent connectivity due to external or internal constraints.

Overall, the core issue is that existing membership abstractions and emulators cannot meet the demands of highly dynamic systems like satellite swarms and MANETs, as they lack adaptability to unpredictable conditions and the ability to leverage a priori communication patterns.

## 3.2 Proposed Solution

To address the challenges posed by highly dynamic systems, this dissertation proposes two key contributions: a prototype tool for emulating dynamic communication patterns and a decentralized membership abstraction tailored to such environments.

The prototype tool will be integrated into the Babel [70] framework, a platform for building performant and dependable distributed protocols. It will function as a middleware layer between the distributed protocols developed in Babel and Babel's network abstractions (channels). This layer will emulate dynamic communication restrictions, including selective communication patterns, intermittent connectivity, and other network constraints.

For the membership service, the solution involves designing and implementing a decentralized membership protocol by also leveraging Babel's ecosystem. This protocol will be specifically optimized for highly dynamic network conditions. Unlike traditional fault-tolerant approaches, this abstraction will proactively exploit a priori knowledge about communication dynamics to optimize membership strategies. By dynamically adjusting its behavior to account for communication restrictions and patterns, the protocol will maintain accurate and consistent membership views with minimal overhead, even under adverse conditions. The design will draw inspiration from existing approaches, such as the HyParView [136] protocol while incorporating novel adaptations to meet the unique demands of scenarios such as satellite swarms and MANETs.

The design of the prototype emulator and the decentralized membership service represents a symbiotic relationship, where each component complements and enhances the other. The emulator provides a highly realistic and controlled environment to test and refine the membership protocol under dynamic and challenging network conditions. In turn, the membership service drives the emulator's development by offering a highly relevant practical use case that highlights its capabilities. This interplay not only strengthens the validity of the proposed solutions but also contributes to the advancement of tools and abstractions for research in highly dynamic systems.

## 3.3 Evaluation

The evaluation of the proposed solution will focus on assessing both the decentralized membership protocol and the prototype tool for emulating dynamic communication patterns.

For the membership protocol, the primary goal is to evaluate its performance and adaptability under dynamic network conditions. Comparative tests will be conducted against well-established membership abstractions, including HyParView [136], X-BOT [139], and Thicket [65], to measure properties such as communication overhead, accuracy of membership views, and resilience to network dynamics. Additionally, the membership protocol will be tested using the prototype tool, which will emulate various dynamic scenarios, including selective communication patterns and intermittent connectivity, to further assess its robustness and efficiency in realistic conditions.

Evaluating the prototype tool presents unique challenges due to the lack of comparable systems. The evaluation will, therefore, focus on several key metrics and properties:

- **Resource Efficiency:** Measuring CPU (Central Processing Unit) and memory consumption to ensure the tool operates within acceptable performance bounds.
- **Precision:** Evaluating the accuracy of the emulation in reproducing intended communication restrictions and patterns.
- **Time:** Analyzing the time required for the tool to stabilize and enforce the desired network dynamics.
- **Consistency:** Ensuring that metrics observed when running a distributed protocol on top of the tool closely match those obtained when the protocol is executed without the tool. To ascertain correctness, tests will focus on application-dependent metrics relevant to our use cases. For example, in MANETs, data aggregation from sensors will leverage distributed algorithms such as MIRAge [46, 49] to assess the tool's accuracy and efficiency under dynamic conditions. In satellite swarms, tests will focus on calculating orbital and positional parameters [215, 165], and other geophysical parameters based on distributed measurements.

## 3.4 Work Plan

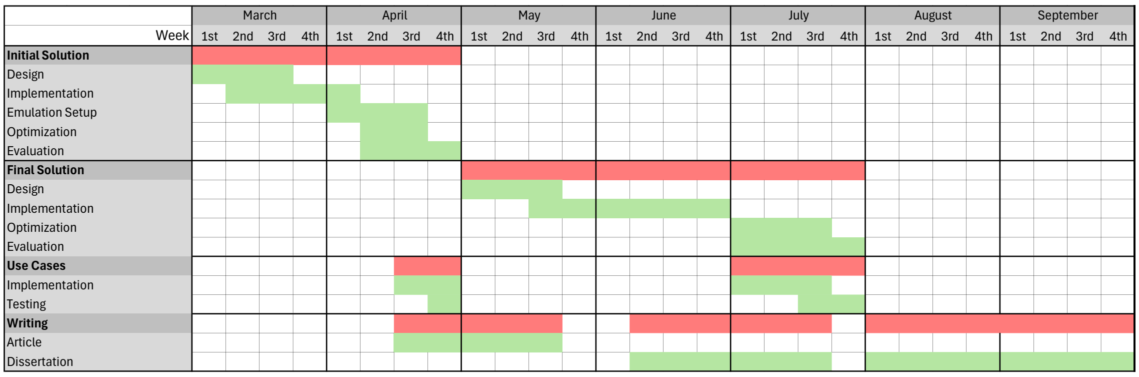

Figure 3.1: Gantt chart of the proposed work schedule

Figure 3.1 presents the schedule for the proposed work, outlining the key tasks and their dependencies. The work plan includes four major tasks, each encompassing several subtasks, as detailed below:

- **Initial Solution:** This task is dedicated to the design and development of the prototype tool (emulator).

  - **Design:** Formulate the architecture and features of the prototype.
  - **Implementation:** Develop the tool within the Babel framework [70].
  - **Emulation Setup:** Configure and prepare the tool for the evaluation stage.
  - **Optimization:** Refine the prototype for resource efficiency and precision.
  - **Evaluation:** Assess the tool's performance based on metrics such as CPU/memory consumption and emulation accuracy.

  This task will last approximately 8 weeks and is a prerequisite for subsequent tasks, particularly the evaluation of the membership service.

- **Final Solution:** This task focuses on the design and implementation of the decentralized membership service.

  - **Design:** Define the protocol's structure and adaptive strategies, leveraging Babel's ecosystem.
  - **Implementation:** Develop the protocol, incorporating dynamic adaptation capabilities.
  - **Optimization:** Enhance performance and reduce communication overhead.
  - **Evaluation:** Compare the protocol against HyParView [136], X-BOT [139], and Thicket [65], and assess it using the prototype tool.

  This task will comprise 12 weeks of work and depends on the completion of the *Initial Solution* for its evaluation.

- **Use Cases:** This task involves demonstrating the solution's applicability in real-world scenarios.

  - **Implementation:** Implement satellite swarm and MANET-specific scenarios, and also demonstrate other applications of the prototype tool.
  - **Testing:** Evaluate the solution's performance and correctness in these scenarios.

  This task will require 6 weeks and depends on the *Initial Solution* for use cases involving the prototype tool/emulator and on the *Final Solution* for use cases involving the membership service.

- **Writing:** This task focuses on the elaboration of the dissertation and the dissemination of the work.

  - **Article:** Prepare and submit a paper for publication at INForum 2025 [99].
  - **Dissertation:** Write and finalize the dissertation document.

  This task is a continuous process spanning from March to September 2025 and running in parallel with the later stages of other tasks.

# Bibliography

[1] M. Abolhasan, T. Wysocki, and E. Dutkiewicz. "A review of routing protocols for mobile ad hoc networks". In: *Ad Hoc Networks* 2.1 (2004), pp. 1–22. ISSN: 1570-8705. DOI: https://doi.org/10.1016/S1570-8705(03)00043-X. URL: https://www.sciencedirect.com/science/article/pii/S157087050300043X (cit. on p. 1).

[2] L. Abraham et al. "Swarm robotics in disaster management". In: *2019 International Conference on Innovative Sustainable Computational Technologies (CISCT)*. IEEE. 2019, pp. 1–5 (cit. on p. 1).

[3] G. Agha. *Actors: a model of concurrent computation in distributed systems*. MIT press, 1986 (cit. on p. 10).

[4] D. E. M. Ahmed and O. O. Khalifa. "An overview of MANETs: applications, characteristics, challenges and recent issues". In: (2017) (cit. on p. 31).

[5] J. Ahrenholz. "Comparison of CORE network emulation platforms". In: *2010-Milcom 2010 Military Communications Conference*. IEEE. 2010, pp. 166–171 (cit. on pp. 21, 22).

[6] J. Ahrenholz, T. Goff, and B. Adamson. "Integration of the CORE and EMANE Network Emulators". In: *2011-MILCOM 2011 Military Communications Conference*. IEEE. 2011, pp. 1870–1875 (cit. on p. 21).

[7] J. Ahrenholz et al. "CORE: A real-time network emulator". In: *2008 IEEE Military Communications Conference (2008-MILCOM)*. IEEE. 2008, pp. 1–7 (cit. on p. 22).

[8] M. Allman, A. Caldwell, and S. Ostermann. *ONE: The ohio network emulator*. Tech. rep. Technical Report TR-19972, Ohio University, 1997 (cit. on p. 18).

[9] W. Almesberger. "Linux Network Traffic Control - Implementation Overview". In: (2001-07), p. 13 (cit. on p. 20).

[10] W. Almesberger, J. H. Salim, and A. Kuznetsov. “Differentiated services on linux”. In: *Seamless Interconnection for Universal Services. Global Telecommunications Conference. GLOBECOM’99.(Cat. No. 99CH37042)*. Vol. 1. IEEE. 1999, pp. 831–836 (cit. on p. 20).

[11] Amazon Web Services. *Amazon EC2 Concepts - User Guide*. 2025. URL: `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html` (visited on 2025-01-23) (cit. on p. 25).

[12] M. Anathi and K. Vijayakumar. “An intelligent approach for dynamic network traffic restriction using MAC address verification”. In: *Computer Communications* 154 (2020), pp. 559–564 (cit. on p. 1).

[13] S. El-Ansary and S. Haridi. “An overview of structured overlay networks”. In: (2005) (cit. on p. 5).

[14] M. Y. Arafat and S. Moh. “Localization and clustering based on swarm intelligence in UAV networks for emergency communications”. In: *IEEE Internet of Things Journal* 6.5 (2019), pp. 8958–8976 (cit. on p. 1).

[15] J. Armstrong. “Making reliable distributed systems in the presence of software errors”. PhD thesis. 2003 (cit. on p. 10).

[16] S. Arora and R. Kaur. “Nature inspired range based wireless sensor node localization algorithms”. In: (2017) (cit. on p. 1).

[17] S. Arora and S. Singh. “Node localization in wireless sensor networks using butterfly optimization algorithm”. In: *Arabian Journal for Science and Engineering* 42 (2017), pp. 3325–3335 (cit. on p. 1).

[18] J. Augustine, G. Pandurangan, and P. Robinson. “Distributed algorithmic foundations of dynamic networks”. In: *ACM SIGACT News* 47.1 (2016), pp. 69–98 (cit. on pp. 2, 26, 30, 31).

[19] J. Augustine et al. “Towards robust and efficient computation in dynamic peer-to-peer networks”. In: *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2012, pp. 551–569 (cit. on pp. 2, 26, 30, 31).

[20] A. Ballesteros et al. “An infrastructure for enabling dynamic fault tolerance in highly-reliable adaptive distributed embedded systems based on switched ethernet”. In: *Sensors* 22.18 (2022), p. 7099 (cit. on p. 2).

[21] R. Barbosa, A. Ferreira, and J. Karlsson. “Implementation of a flexible membership protocol on a real-time ethernet prototype”. In: *13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007)*. IEEE. 2007, pp. 342–347 (cit. on pp. 1, 2, 30).

[22] S. A. Baset and H. Schulzrinne. “An analysis of the skype peer-to-peer internet telephony protocol”. In: *arXiv preprint cs/0412017* (2004) (cit. on p. 4).

[23] J. Beale, A. Orebaugh, and G. Ramirez. *Wireshark & Ethereal network protocol analyzer toolkit*. Elsevier, 2006 (cit. on p. 14).

[24] O. Ben-Kiki, C. Evans, and I. döt Net. *YAML Ain't Markup Language (YAML) Version 1.2*. https://yaml.org/spec/1.2/spec.html. 2009. URL: `https://yaml.org/spec/1.2/spec.html` (visited on 2025-01-22) (cit. on p. 24).

[25] A. Bessani, J. Sousa, and E. E. Alchieri. "State machine replication for the masses with BFT-SMART". In: *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE. 2014, pp. 355–362 (cit. on pp. 9, 25).

[26] L. C. de Biase et al. "Swarm Computing: The Emergence of a Collective Artificial Intelligence at the Edge of the Internet". In: *Edge Computing*. Ed. by S. Goundar. Rijeka: IntechOpen, 2023. Chap. 5. DOI: `10.5772/intechopen.110907`. URL: `https://doi.org/10.5772/intechopen.110907` (cit. on p. 1).

[27] K. P. Birman and T. A. Joseph. "Reliable communication in the presence of failures". In: *ACM Transactions on Computer Systems (TOCS)* 5.1 (1987), pp. 47–76 (cit. on pp. 1, 25).

[28] K. P. Birman et al. "Bimodal multicast". In: *ACM Transactions on Computer Systems (TOCS)* 17.2 (1999), pp. 41–88 (cit. on p. 27).

[29] E. Bonabeau. "Swarm Intelligence: From Natural to Artificial Systems". In: *Oxford University Press* 2 (1999), pp. 25–34 (cit. on p. 1).

[30] G. Bonofiglio et al. "Kathará: A container-based framework for implementing network function virtualization and software defined networks". In: *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE. 2018, pp. 1–9 (cit. on p. 23).

[31] L. S. Brakmo and L. L. Peterson. "TCP Vegas: End to end congestion avoidance on a global Internet". In: *IEEE Journal on Selected Areas in Communications* 13.8 (1995), pp. 1465–1480 (cit. on p. 20).

[32] M. Budiu and C. Dodd. "The p416 programming language". In: *ACM SIGOPS Operating Systems Review* 51.1 (2017), pp. 5–14 (cit. on p. 23).

[33] H. Byun and M. Lee. "Hypo: a peer-to-peer based hybrid overlay structure". In: *2009 11th International Conference on Advanced Communication Technology*. Vol. 1. IEEE. 2009, pp. 840–844 (cit. on p. 5).

[34] R. N. Calheiros et al. "Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services". In: *arXiv preprint arXiv:0903.2525* (2009) (cit. on p. 15).

[35] B. Canakci and R. Van Renesse. "Scaling membership of Byzantine consensus". In: *ACM Transactions on Computer Systems (TOCS)* 38.3-4 (2021), pp. 1–31 (cit. on p. 25).

[36] B. Carlsson and R. Gustavsson. "The rise and fall of napster-an evolutionary approach". In: *International Computer Science Conference on Active Media Technology*. Springer. 2001, pp. 347–354 (cit. on p. 9).

[37] N. Carriero, D. Gelernter, and J. Leichter. "Distributed data structures in Linda". In: *Proceedings of the 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. 1986, pp. 236–242 (cit. on p. 5).

[38] M. Carson and D. Santay. "NIST Net: a Linux-based network emulation tool". In: *ACM SIGCOMM Computer Communication Review* 33.3 (2003), pp. 111–126 (cit. on pp. 18, 21).

[39] M. Castro et al. "Secure routing for structured peer-to-peer overlay networks". In: *ACM SIGOPS Operating Systems Review* 36.SI (2002), pp. 299–314 (cit. on p. 5).

[40] R. Catling. *The VirtualBox Networking Primer: Connecting and Configuring Virtual Machines*. Illustrated. Proactivity Press, 2020, p. 134. ISBN: 1916119484 (cit. on p. 22).

[41] V. Cerf and R. Kahn. "A protocol for packet network intercommunication". In: *IEEE Transactions on Communications* 22.5 (1974), pp. 637–648 (cit. on p. 12).

[42] X. Chang. "Network simulations with OPNET". In: *Proceedings of the 31st Conference on Winter Simulation: Simulation—A Bridge to the Future-Volume 1*. 1999, pp. 307–314 (cit. on p. 13).

[43] Y. Chawathe et al. "Making gnutella-like P2P systems scalable". In: *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '03. Karlsruhe, Germany: Association for Computing Machinery, 2003, pp. 407–418. ISBN: 1581137354. DOI: 10.1145/863955.864000. URL: https://doi.org/10.1145/863955.864000 (cit. on p. 6).

[44] Y. Chawathe et al. "Making gnutella-like p2p systems scalable". In: *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. 2003, pp. 407–418 (cit. on p. 5).

[45] O. Corporation. *Java SE 23 Documentation*. 2024. URL: https://docs.oracle.com/en/java/javase/23/ (visited on 2025-01-22) (cit. on p. 8).

[46] P. Á. Costa and J. Leitão. "Practical continuous aggregation in wireless edge environments". In: *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. IEEE. 2018, pp. 41–50 (cit. on p. 33).

[47] P. Á. Costa, A. Rosa, and J. Leitão. "Enabling wireless ad hoc edge systems with yggdrasil". In: *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. 2020, pp. 2129–2136 (cit. on p. 8).

[48] P. Á. H. F. da Costa. "Practical Aggregation in the Edge". PhD thesis. NOVA University of Lisbon, 2018 (cit. on p. 8).

[49] P. Á. H. F. da Costa. “Practical Aggregation in the Edge”. PhD thesis. NOVA University of Lisbon, 2018 (cit. on p. 33).

[50] L. Da Xu, W. He, and S. Li. “Internet of things in industries: A survey”. In: *IEEE Transactions on Industrial Informatics* 10.4 (2014), pp. 2233–2243 (cit. on p. 1).

[51] J. L. De Sousa. “Affinity Based Overlays for Decentralized Systems”. In: (2023) (cit. on pp. 5, 6).

[52] G. DeCandia et al. “Dynamo: Amazon’s highly available key-value store”. In: *ACM SIGOPS Operating Systems Review* 41.6 (2007), pp. 205–220 (cit. on p. 5).

[53] A. Dell’Aera, L. A. Grieco, and S. Mascolo. “Linux 2.4 Implementation of Westwood+ TCP with rate-halving: A Performance Evaluation over the Internet”. In: *2004 IEEE International Conference on Communications (IEEE Cat. No. 04CH37577)*. Vol. 4. IEEE. 2004, pp. 2092–2096 (cit. on p. 20).

[54] A. Demers et al. “Epidemic algorithms for replicated database maintenance”. In: *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*. 1987, pp. 1–12 (cit. on p. 27).

[55] M. Deshpande et al. “Crew: A gossip-based flash-dissemination system”. In: *26th IEEE International Conference on Distributed Computing Systems (ICDCS’06)*. IEEE. 2006, pp. 45–45 (cit. on p. 26).

[56] L. R. Dondeti, S. Mukherjee, and A. Samal. “Scalable secure one-to-many group communication using dual encryption”. In: *Computer Communications* 23.17 (2000), pp. 1681–1701 (cit. on p. 25).

[57] M. Dorigo et al. “Swarm intelligence”. In: *Scholarpedia* 2.9 (2007), p. 1462 (cit. on p. 1).

[58] J. Dowling and A. H. Payberah. “Shuffling with a croupier: Nat-aware peer-sampling”. In: *2012 IEEE 32nd International Conference on Distributed Computing Systems*. IEEE. 2012, pp. 102–111 (cit. on p. 6).

[59] H. Du and D. J. S. Hilaire. “Multi-Paxos: An implementation and evaluation”. In: *Department of Computer Science and Engineering, University of Washington, Tech. Rep. UW-CSE-09-09-02* (2009), p. 10 (cit. on p. 9).

[60] F. Ducatelle, G. A. Di Caro, and L. M. Gambardella. “Principles and applications of swarm intelligence for adaptive routing in telecommunications networks”. In: *Swarm Intelligence* 4.3 (2010), pp. 173–198 (cit. on p. 1).

[61] S. Engelen, E. K. Gill, and C. J. Verhoeven. “Systems engineering challenges for satellite swarms”. In: *2011 Aerospace Conference*. IEEE. 2011, pp. 1–8 (cit. on p. 31).

[62] Ericsson. *Ericsson Official Website*. 2025. URL: https://www.ericsson.com (visited on 2025-01-22) (cit. on p. 10).

[63] P. T. Eugster et al. "Lightweight probabilistic broadcast". In: *ACM Transactions on Computer Systems (TOCS)* 21.4 (2003), pp. 341–374 (cit. on p. 26).

[64] V. Farcic. *The DevOps 2.1 Toolkit: Docker Swarm*. Packt Publishing Ltd, 2017 (cit. on p. 24).

[65] M. Ferreira, J. Leitão, and L. Rodrigues. "Thicket: A protocol for building and maintaining multiple trees in a p2p overlay". In: *2010 29th IEEE Symposium on Reliable Distributed Systems*. IEEE. 2010, pp. 293–302 (cit. on pp. 6, 27, 32, 34).

[66] S. Floyd. "TCP and explicit congestion notification". In: *ACM SIGCOMM Computer Communication Review* 24.5 (1994), pp. 8–23 (cit. on p. 18).

[67] P. Fonseca and H. Miranda. "FASE: Reaching Scalability in Unstructured P2P Networks Using a Divide and Conquer Strategy". In: (2008) (cit. on p. 6).

[68] A. S. Foundation. *Apache HTTP Server*. 2025. URL: `https://httpd.apache.org/` (visited on 2025-01-23) (cit. on p. 19).

[69] P. S. Foundation. *Python Language Reference, version 3.13.2*. 2025. URL: `https://docs.python.org/3/reference/index.html` (visited on 2025-01-22) (cit. on p. 14).

[70] P. Fouto et al. "Babel: A framework for developing performant and dependable distributed protocols". In: *2022 41st International Symposium on Reliable Distributed Systems (SRDS)*. IEEE. 2022, pp. 146–155 (cit. on pp. 6, 8, 32, 34).

[71] M. Furman. *OpenVZ essentials*. Packt Publishing Ltd, 2014 (cit. on p. 21).

[72] S. Gandhi, R. K. Singh, et al. "Design and development of dynamic satellite link emulator with experimental validation". In: *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. IEEE. 2021, pp. 1–6 (cit. on p. 2).

[73] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. "HiScamp: self-organizing hierarchical membership protocol". In: *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*. 2002, pp. 133–139 (cit. on p. 5).

[74] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. "Peer-to-peer membership management for gossip-based protocols". In: *IEEE Transactions on Computers* 52.2 (2003), pp. 139–149 (cit. on pp. 5, 25).

[75] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. "Scamp: Peer-to-peer lightweight membership service for large-scale group communication". In: *Networked Group Communication: Third International COST264 Workshop, NGC 2001 London, UK, November 7–9, 2001 Proceedings 3*. Springer. 2001, pp. 44–55 (cit. on pp. 5, 26).

[76] P. Garimella et al. "Characterizing VLAN usage in an operational network". In: *Proceedings of the 2007 SIGCOMM Workshop on Internet Network Management*. 2007, pp. 305–306 (cit. on p. 19).

[77] T. Goodale et al. “The cactus framework and toolkit: design and applications: invited talk”. In: *High Performance Computing for Computational Science—VECPAR 2002: 5th International Conference Porto, Portugal, June 26–28, 2002 Selected Papers and Invited Talks 5*. Springer. 2002, pp. 197–227 (cit. on p. 7).

[78] P. Gouveia et al. “Kollaps: decentralized and dynamic topology emulation”. In: *Proceedings of the Fifteenth European Conference on Computer Systems*. 2020, pp. 1–16 (cit. on p. 24).

[79] N. Goyal and A. Gaba. “A review over MANET-Issues and Challenges”. In: *International Journal of Enhanced Research in Management and Computer Applications (IJERMCA)* 2 (2013), pp. 24–36 (cit. on p. 31).

[80] T. T. Group. *tcpdump - Dump Traffic on a Network*. 2025. URL: `https://www.tcpdump.org/manpages/tcpdump.1.html` (visited on 2025-01-23) (cit. on p. 19).

[81] T. Gui et al. “Survey on swarm intelligence based routing protocols for wireless sensor networks: An extensive study”. In: *2016 IEEE International Conference on Industrial Technology (ICIT)*. IEEE. 2016, pp. 1944–1949 (cit. on p. 1).

[82] K. P. Gummadi, S. Saroiu, and S. D. Gribble. “King: Estimating latency between arbitrary internet end hosts”. In: *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*. 2002, pp. 5–18 (cit. on p. 15).

[83] A. Gupta, P. Verma, and R. S. Sambyal. “An overview of MANET: features, challenges and applications”. In: *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* 4.1 (2018), pp. 122–126 (cit. on p. 31).

[84] M. Gupta. *Akka essentials*. Packt Publishing Ltd, 2012 (cit. on p. 10).

[85] J. Halpern and C. Pignataro. *Service function chaining (SFC) architecture*. Tech. rep. 2015 (cit. on p. 23).

[86] A. D. Hands et al. “Radiation effects on satellites during extreme space weather events”. In: *Space Weather* 16.9 (2018), pp. 1216–1226 (cit. on p. 31).

[87] C. Hedrick. *Routing Information Protocol*. RFC 1058. 1988-06. DOI: `10.17487/RFC1058`. URL: `https://www.rfc-editor.org/info/rfc1058` (cit. on p. 19).

[88] X. Hei et al. “A measurement study of a large-scale P2P IPTV system”. In: *IEEE Transactions on Multimedia* 9.8 (2007), pp. 1672–1687 (cit. on p. 4).

[89] M. K. Heinrich et al. “Swarm robotics: Robustness, scalability, and self-X features in industrial applications”. In: *IT-Information Technology* 61.4 (2019), pp. 159–167 (cit. on p. 1).

[90] S. Hemminger et al. “Network emulation with NetEm”. In: *Linux conf au*. Vol. 5. 2005, p. 2005 (cit. on pp. 20, 21).

[91] M. Hibler et al. “Large-scale virtualization in the emulab network testbed”. In: *2008 USENIX Annual Technical Conference (USENIX ATC 08)*. 2008 (cit. on p. 24).

[92] G. R. Hiertz et al. "The IEEE 802.11 universe". In: *IEEE Communications Magazine* 48.1 (2010), pp. 62–70 (cit. on p. 12).

[93] J. Hoebeke et al. "An overview of mobile ad hoc networks: applications and challenges". In: *Journal of Communications and Networks* 3.3 (2004), pp. 60–66 (cit. on p. 31).

[94] D. Hou et al. "A realistic, flexible and extendible network emulation platform for space networks". In: *Electronics* 11.8 (2022), p. 1236 (cit. on p. 31).

[95] C.-H. Hsu and U. Kremer. "IPERF: A framework for automatic construction of performance prediction models". In: *Workshop on Profile and Feedback-Directed Compilation (PFDC), Paris, France*. Citeseer. 1998 (cit. on p. 20).

[96] C. Huitema. *IPv6: the new Internet protocol*. Prentice-Hall, Inc., 1995 (cit. on p. 14).

[97] P. Hunt et al. "{ZooKeeper}: Wait-free coordination for internet-scale systems". In: *2010 USENIX Annual Technical Conference (USENIX ATC 10)*. 2010 (cit. on p. 1).

[98] A. Iamnitchi and I. Foster. "A problem-specific fault-tolerance mechanism for asynchronous, distributed systems". In: *Proceedings 2000 International Conference on Parallel Processing*. IEEE. 2000, pp. 4–13 (cit. on p. 25).

[99] INForum - Simpósio de Informática. *INForum - Simpósio de Informática*. 2025. URL: https://inforum.org.pt/en#div%5C_acerca (visited on 2025-01-23) (cit. on p. 34).

[100] *iproute2 Networking*. Linux Foundation. 2025. URL: https://wiki.linuxfoundation.org/networking/iproute2 (visited on 2025-01-23) (cit. on p. 21).

[101] T. Issariyakul et al. *Introduction to network simulator 2 (NS2)*. Springer, 2009 (cit. on pp. 7, 11, 12).

[102] W. A. Jabbar et al. "Power-efficient routing schemes for MANETs: a survey and open issues". In: *Wireless Networks* 23 (2017), pp. 1917–1952 (cit. on p. 31).

[103] S. M. Jain. *Linux Containers and Virtualization. A Kernel Perspective*. 1st ed. Berkeley, CA: Apress, 2020-10, p. 148. ISBN: 978-1-4842-6283-2. DOI: https://doi.org/10.1007/978-1-4842-6283-2 (cit. on p. 22).

[104] M. Jelasity and A. Montresor. "Epidemic-style proactive aggregation in large overlay networks". In: *24th International Conference on Distributed Computing Systems, 2004. Proceedings.* IEEE. 2004, pp. 102–109 (cit. on p. 5).

[105] M. Jelasity, A. Montresor, and O. Babaoglu. "T-man: Gossip-based fast overlay topology construction". In: *Computer Networks* 53.13 (2009), pp. 2321–2339 (cit. on pp. 6, 27).

[106] J. A. Johnsen, L. E. Karlsen, and S. S. Birkeland. "Peer-to-peer networking with BitTorrent". In: *Department of Telematics, NTNU* (2005) (cit. on p. 4).

[107] D. B. Johnson, D. A. Maltz, J. Broch, et al. "DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks". In: *Ad hoc Networking* 5.1 (2001), pp. 139–172 (cit. on p. 12).

[108] D. Johnston. "Radio frequency interference protection of communications between the Deep Space Network and deep space flight projects". In: *The Telecommunications and Data Acquisition Progress Report 42-62* (1981), p. 125 (cit. on p. 31).

[109] P. Joshi. "Security issues in routing protocols in MANETs at network layer". In: *Procedia Computer Science* 3 (2011), pp. 954–960 (cit. on p. 31).

[110] K. Kaemarungsi and P. Krishnamurthy. "Properties of indoor received signal strength for WLAN location fingerprinting". In: *The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004.* IEEE. 2004, pp. 14–23 (cit. on p. 2).

[111] P.-H. Kamp and R. N. Watson. "Jails: Confining the omnipotent root". In: *Proceedings of the 2nd International SANE Conference*. Vol. 43. 2000, p. 116 (cit. on p. 21).

[112] T. Kang, X. Sun, and T. Zhang. "Base station switching based dynamic energy saving algorithm for cellular networks". In: *2012 3rd IEEE International Conference on Network Infrastructure and Digital Content*. IEEE. 2012, pp. 66–70 (cit. on p. 2).

[113] L. Kant and R. Chadha. "MANET management: Industry challenges & potential solutions". In: *2008 International Symposium on a World of Wireless, Mobile and Multimedia Networks*. IEEE. 2008, pp. 1–8 (cit. on p. 31).

[114] Y. Kantaros and M. M. Zavlanos. "Distributed communication-aware coverage control by mobile sensor networks". In: *Automatica* 63 (2016), pp. 209–220 (cit. on pp. 2, 30).

[115] Y. Kantaros and M. M. Zavlanos. "Intermittent connectivity control in mobile robot networks". In: *2015 49th Asilomar Conference on Signals, Systems and Computers*. IEEE. 2015, pp. 1125–1129 (cit. on pp. 2, 30).

[116] A.-M. Kermarrec, L. Massoulié, and A. J. Ganesh. "Efficient application-level multicast on a network-aware self-organizing overlay". In: *IRISA Internal Publication, 1166-8687* 1657 (2004) (cit. on p. 5).

[117] A.-M. Kermarrec et al. "Nat-resilient gossip peer sampling". In: *2009 29th IEEE International Conference on Distributed Computing Systems*. IEEE. 2009, pp. 360–367 (cit. on p. 6).

[118] Y. Kermarrec and L. Pautet. "Ada reusable software components for teaching distributed systems". In: *Software Engineering Education: 7th SEI CSEE Conference San Antonio, Texas, USA, January 5–7, 1994 Proceedings 7*. Springer. 1994, pp. 77–96 (cit. on p. 6).

[119] M. Kerrisk. *The Linux programming interface: a Linux and UNIX system programming handbook*. No Starch Press, 2010 (cit. on p. 18).

[120] R. I. Khazan. "Group membership: a novel approach and the first single-round algorithm". In: *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing*. 2004, pp. 347–356 (cit. on pp. 2, 31).

[121] H. Kopetz, G. Grünsteidl, and J. Reisinger. "Fault-Tolerant Membership Service in a Synchronous Distributed Real-Time System". In: *Dependable Computing for Critical Applications*. Ed. by A. Avižienis and J.-C. Laprie. Vienna: Springer Vienna, 1991, pp. 411–429. ISBN: 978-3-7091-9123-1. DOI: `10.1007/978-3-7091-9123-1_19`. URL: `https://doi.org/10.1007/978-3-7091-9123-1_19` (cit. on p. 25).

[122] D. Kostic et al. "Using Random Subsets to Build Scalable Network Services." In: *USENIX Symposium on Internet Technologies and Systems*. 2003, p. 19 (cit. on p. 5).

[123] R. van Kranenburg et al. "The Internet of Things". In: *Proceedings of the 1st Berlin Symposium on Internet and Society*. 2011, pp. 25–27 (cit. on p. 1).

[124] Y. Kulbak, D. Bickson, et al. "The eMule protocol specification". In: *eMule Project, http://sourceforge.net* (2005) (cit. on p. 9).

[125] M. Kumar and R. Mishra. "An overview of MANET: History, challenges and applications". In: *Indian Journal of Computer Science and Engineering (IJCSE)* 3.1 (2012), pp. 121–125 (cit. on p. 31).

[126] S. Kumar, S. Dalal, and V. Dixit. "The osi model: overview on the seven layers of computer networks". In: *International Journal of Computer Science and Information Technology Research* 2.3 (2014), pp. 461–466 (cit. on p. 21).

[127] A. Kushki, K. N. Plataniotis, and A. N. Venetsanopoulos. "Intelligent dynamic radio tracking in indoor wireless local area networks". In: *IEEE Transactions on Mobile Computing* 9.3 (2009), pp. 405–419 (cit. on p. 2).

[128] M. Lacage and T. R. Henderson. "Yet another network simulator". In: *Proceedings of the 2006 Workshop on ns-3*. 2006, 12–es (cit. on p. 14).

[129] A. Lakshman and P. Malik. "Cassandra: a decentralized structured storage system". In: *ACM SIGOPS Operating Systems Review* 44.2 (2010), pp. 35–40 (cit. on p. 25).

[130] L. Lamport. "Paxos made simple". In: *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)* (2001), pp. 51–58 (cit. on pp. 1, 9).

[131] B. Lantz, B. Heller, and N. McKeown. "A network in a laptop: rapid prototyping for software-defined networks". In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. 2010, pp. 1–6 (cit. on pp. 19, 24).

[132] C. G. Lee et al. "Micro-meteoroid and orbital debris radar from Goldstone radar observations". In: *Journal of Space Safety Engineering* 7.3 (2020), pp. 242–248 (cit. on p. 31).

[133] J. Leitão. "Gossip-based broadcast protocols". MA thesis. University of Lisbon, 2007 (cit. on p. 26).

[134] J. Leitão. "Topology management for unstructured overlay networks". In: *Technical University of Lisbon* (2012), p. 12 (cit. on pp. 4–6).

[135] J. Leitão, J. Pereira, and L. Rodrigues. "Epidemic broadcast trees". In: *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*. IEEE. 2007, pp. 301–310 (cit. on pp. 6, 27).

[136] J. Leitão, J. Pereira, and L. Rodrigues. "HyParView: A membership protocol for reliable gossip-based broadcast". In: *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. IEEE. 2007, pp. 419–429 (cit. on pp. 2, 5, 9, 26, 31, 32, 34).

[137] J. Leitão, R. van Renesse, and L. Rodrigues. "Balancing gossip exchanges in networks with firewalls." In: *IPTPS*. 2010, p. 7 (cit. on p. 6).

[138] J. Leitão et al. "On adding structure to unstructured overlay networks". In: *Handbook of Peer-to-Peer Networking* (2010), pp. 327–365 (cit. on p. 5).

[139] J. C. A. Leitão et al. "X-bot: A protocol for resilient optimization of unstructured overlays". In: *2009 28th IEEE International Symposium on Reliable Distributed Systems*. IEEE. 2009, pp. 236–245 (cit. on pp. 6, 26, 27, 32, 34).

[140] J. Li et al. "A performance vs. cost framework for evaluating DHT design tradeoffs under churn". In: *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies*. Vol. 1. IEEE. 2005, pp. 225–236 (cit. on p. 4).

[141] D. Lin and R. Morris. "Dynamics of random early detection". In: *Proceedings of the ACM SIGCOMM'97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. 1997, pp. 127–137 (cit. on p. 18).

[142] H. H. Liu et al. "Crystalnet: Faithfully emulating large production networks". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. 2017, pp. 599–613 (cit. on p. 24).

[143] L. Liu and Y. Zhang. "Design of greenhouse environment monitoring system based on Wireless Sensor Network". In: *2017 3rd International Conference on Control, Automation and Robotics (ICCAR)*. IEEE. 2017, pp. 463–466 (cit. on p. 1).

[144] K. Lougheed and Y. Rekhter. *Border Gateway Protocol (BGP)*. RFC 1105. 1989-06. DOI: `10.17487/RFC1105`. URL: `https://www.rfc-editor.org/info/rfc1105` (cit. on p. 19).

[145] E. K. Lua et al. "A survey and comparison of peer-to-peer overlay network schemes". In: *IEEE Communications Surveys & Tutorials* 7.2 (2005), pp. 72–93 (cit. on pp. 4, 5).

[146] M. Luksa. *Kubernetes in action*. Simon and Schuster, 2017 (cit. on p. 24).

[147] J. Luo, P. T. Eugster, and J.-P. Hubaux. “Route driven gossip: Probabilistic reliable multicast in ad hoc networks”. In: *IEEE INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428)*. Vol. 3. IEEE. 2003, pp. 2229–2239 (cit. on p. 5).

[148] R. J. A. Macêdo and F. M. A. Silva. “The mobile groups approach for the coordination of mobile agents”. In: *Journal of Parallel and Distributed Computing* 65.3 (2005), pp. 275–288 (cit. on p. 1).

[149] G. S. Machado et al. “A cloud storage overlay to aggregate heterogeneous cloud services”. In: *38th Annual IEEE Conference on Local Computer Networks*. IEEE. 2013, pp. 597–605 (cit. on p. 5).

[150] S. T. Maguluri and R. Srikant. “Scheduling jobs with unknown duration in clouds”. In: *IEEE/ACM Transactions On Networking* 22.6 (2013), pp. 1938–1951 (cit. on p. 1).

[151] S. T. Maguluri, R. Srikant, and L. Ying. “Heavy traffic optimal resource allocation algorithms for cloud computing clusters”. In: *Performance Evaluation* 81 (2014), pp. 20–39 (cit. on p. 1).

[152] M. Marszalek, M. Rummelhagen, and F. Schramm. “Potentials and limitations of IEEE 802.11 for satellite swarms”. In: *2014 IEEE Aerospace Conference*. IEEE. 2014, pp. 1–9 (cit. on p. 31).

[153] N. Maurer and M. Wolfthal. *Netty in action*. Simon and Schuster, 2015 (cit. on p. 9).

[154] P. Maymounkov and D. Mazieres. “Kademlia: A peer-to-peer information system based on the xor metric”. In: *International Workshop on Peer-to-Peer Systems*. Springer. 2002, pp. 53–65 (cit. on p. 5).

[155] N. McKeown et al. “OpenFlow: enabling innovation in campus networks”. In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), pp. 69–74 (cit. on p. 23).

[156] R. Melamed and I. Keidar. “Araneola: A scalable reliable multicast system for dynamic environments”. In: *Third IEEE International Symposium on Network Computing and Applications, 2004.(NCA 2004). Proceedings*. IEEE. 2004, pp. 5–14 (cit. on pp. 5, 27).

[157] D. A. Menascé and L. Kanchanapalli. “Probabilistic scalable P2P resource location services”. In: *ACM SIGMETRICS Performance Evaluation Review* 30.2 (2002), pp. 48–58 (cit. on p. 5).

[158] D. Merkel et al. “Docker: lightweight linux containers for consistent development and deployment”. In: *Linux J.* 239.2 (2014), p. 2 (cit. on p. 22).

[159] N. N. Mhala and N. Choudhari. “An Envision of Low Cost Mobile Adhoc Network Test Bed in a Laboratory Environment Emulating an Actual MANET”. In: *arXiv preprint arXiv:1005.1787* (2010) (cit. on p. 2).

[160] O. Michail and P. G. Spirakis. "Elements of the theory of dynamic networks". In: *Communications of the ACM* 61.2 (2018), pp. 72–72 (cit. on pp. 2, 30).

[161] R. Mijumbi et al. "Network function virtualization: State-of-the-art and research challenges". In: *IEEE Communications Surveys & Tutorials* 18.1 (2015), pp. 236–262 (cit. on p. 23).

[162] H. Miranda, A. Pinto, and L. Rodrigues. "Appia, a flexible protocol kernel supporting multiple coordinated channels". In: *Proceedings 21st International Conference on Distributed Computing Systems*. IEEE. 2001, pp. 707–710 (cit. on p. 7).

[163] S. Mishra, L. L. Peterson, and R. D. Schlichting. "Consul: A communication substrate for fault-tolerant distributed programs". In: *Distributed Systems Engineering* 1.2 (1993), p. 87 (cit. on p. 25).

[164] A. H. Mohsin. "Optimize routing protocol overheads in MANETs: challenges and solutions: a review paper". In: *Wireless Personal Communications* 126.4 (2022), pp. 2871–2910 (cit. on p. 31).

[165] O. Montenbruck and P. Steigenberger. "GNSS orbit determination and time synchronization". In: *Position, Navigation, and Timing Technologies in the 21st Century: Integrated Satellite Navigation, Sensor Systems, and Civil Applications* 1 (2020), pp. 233–258 (cit. on p. 33).

[166] A. Montresor and M. Jelasity. "PeerSim: A scalable P2P simulator". In: *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*. IEEE. 2009, pp. 99–100 (cit. on pp. 7, 11, 14, 17).

[167] M. Mouly and M.-B. Pautet. *The GSM system for mobile communications*. Telecom Publishing, 1992 (cit. on p. 13).

[168] J. Moy. *OSPF Version 2*. RFC 1583. 1994-03. DOI: `10.17487/RFC1583`. URL: `https://www.rfc-editor.org/info/rfc1583` (cit. on p. 19).

[169] P. Muri, O. Challa, and J. McNair. "Enhancing small satellite communication through effective antenna system design". In: *2010 Military Communications Conference (2010-MILCOM)*. IEEE. 2010, pp. 347–352 (cit. on p. 31).

[170] D. S. Neely. "A scalable decentralized group membership service for an asynchronous environment". PhD thesis. Monterey, California. Naval Postgraduate School, 1994 (cit. on p. 26).

[171] *Netgraph*. FreeBSD. 2025. URL: `https://man.freebsd.org/cgi/man.cgi?netgraph(4)` (visited on 2025-01-23) (cit. on p. 21).

[172] S. Niazi and J. Dowling. "Usurp: Distributed nat traversal for overlay networks". In: *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer. 2011, pp. 29–42 (cit. on p. 6).

[173] L. M. Oliveira and J. J. Rodrigues. "Wireless Sensor Networks: A Survey on Environmental Monitoring." In: *J. Commun.* 6.2 (2011), pp. 143–151 (cit. on p. 1).

[174] D. Ongaro and J. Ousterhout. "In search of an understandable consensus algorithm". In: *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. 2014, pp. 305–319 (cit. on pp. 1, 9).

[175] Oracle Corporation. *The Java Virtual Machine Specification, Java SE 8 Edition*. 2014. URL: `https://docs.oracle.com/javase/specs/jvms/se8/html/` (visited on 2025-01-22) (cit. on p. 10).

[176] L. K. Organization. *User-mode Linux (UML) Documentation*. 2025. URL: `https://www.kernel.org/doc/html/v5.9/virt/uml/user_mode_linux.html` (visited on 2025-01-23) (cit. on p. 19).

[177] J. Oyekan and H. Huosheng. "Toward bacterial swarm for environmental monitoring". In: *2009 IEEE International Conference on Automation and Logistics*. IEEE. 2009, pp. 399–404 (cit. on p. 1).

[178] S. Parjan and S. A. Chien. "Decentralized observation allocation for a large-scale constellation". In: *Journal of Aerospace Information Systems* 20.8 (2023), pp. 447–461 (cit. on p. 31).

[179] V. Paxson and S. Floyd. "Why we don't know how to simulate the Internet". In: *Proceedings of the 29th Conference on Winter Simulation*. 1997, pp. 1037–1044 (cit. on p. 21).

[180] A. H. Payberah, J. Dowling, and S. Haridi. "Gozar: Nat-friendly peer sampling with one-hop distributed nat traversal". In: *Distributed Applications and Interoperable Systems: 11th IFIP WG 6.1 International Conference, DAIS 2011, Reykjavik, Iceland, June 6-9, 2011. Proceedings 11*. Springer. 2011, pp. 1–14 (cit. on p. 6).

[181] J. Pereira et al. "Low latency probabilistic broadcast in wide area networks". In: *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 2004.* IEEE. 2004, pp. 299–308 (cit. on p. 26).

[182] J. Pereira et al. "X-BOT: A Protocol for Resilient Optimization of Unstructured Overlay Networks". In: *IEEE Transactions on Parallel & Distributed Systems* 23.11 (2012-11), pp. 2175–2188. ISSN: 1558-2183. DOI: `10.1109/TPDS.2012.29`. URL: `https://doi.ieeecomputersociety.org/10.1109/TPDS.2012.29` (cit. on pp. 26, 27).

[183] C. Perkins, E. Belding-Royer, and S. Das. *Ad hoc on-demand distance vector (AODV) routing*. Tech. rep. 2003 (cit. on p. 12).

[184] M. Pizzonia and M. Rimondini. "Netkit: network emulation for education". In: *Software: Practice and Experience* 46.2 (2016), pp. 133–165 (cit. on pp. 9, 19).

[185] D. C. Plummer. *An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware*. RFC 826. 1982-11. DOI: `10.17487/RFC0826`. URL: `https://www.rfc-editor.org/info/rfc826` (cit. on p. 19).

[186] J. Postel. *Internet Protocol*. RFC 791. RFC Editor, 1981-09. URL: https://www.rfc-editor.org/rfc/rfc791.txt (cit. on p. 23).

[187] J. Postel. *User datagram protocol*. Tech. rep. 1980 (cit. on p. 12).

[188] Q. Project. *Quagga Routing Suite*. 2025. URL: https://www.nongnu.org/quagga/ (visited on 2025-01-23) (cit. on p. 19).

[189] T. F. Project. *The FreeBSD Handbook*. 2025. URL: https://docs.freebsd.org/en/books/handbook/ (visited on 2025-01-22) (cit. on p. 21).

[190] M. Puzar and T. Plagemann. "NEMAN: a network emulator for mobile ad-hoc networks". In: *Proceedings of the 8th International Conference on Telecommunications, 2005. ConTEL 2005*. Vol. 1. 2005, pp. 155–161. DOI: 10.1109/CONTEL.2005.185841 (cit. on p. 23).

[191] *QoS in Linux: tc and Filters*. Linux.com. 2025. URL: https://www.linux.com/training-tutorials/qos-linux-tc-and-filters/ (visited on 2025-01-23) (cit. on p. 20).

[192] L. Raja, S. S. Baboo, et al. "An overview of MANET: Applications, attacks and challenges". In: *International Journal of Computer Science and Mobile Computing* 3.1 (2014), pp. 408–417 (cit. on p. 31).

[193] D. Raychaudhuri et al. "Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols". In: *IEEE Wireless Communications and Networking Conference, 2005*. Vol. 3. IEEE. 2005, pp. 1664–1669 (cit. on p. 14).

[194] M. K. Reiter. "A secure group membership protocol". In: *IEEE Transactions on Software Engineering* 22.1 (1996), pp. 31–42 (cit. on p. 25).

[195] R. H. Riedi et al. "A multifractal wavelet model with application to network traffic". In: *IEEE Transactions on Information Theory* 45.3 (1999), pp. 992–1018 (cit. on p. 18).

[196] G. F. Riley. "The georgia tech network simulator". In: *Proceedings of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research*. 2003, pp. 5–12 (cit. on p. 14).

[197] G. F. Riley and T. R. Henderson. "The ns-3 network simulator". In: *Modeling and Tools for Network Simulation*. Springer, 2010, pp. 15–34 (cit. on pp. 7, 11, 12, 14, 17).

[198] L. Rizzo. "Dummynet: a simple approach to the evaluation of network protocols". In: *ACM SIGCOMM Computer Communication Review* 27.1 (1997), pp. 31–41 (cit. on pp. 18, 21).

[199] L. Rosa, L. Rodrigues, and A. Lopes. "Appia to R-appia: refactoring a protocol composition framework for dynamic reconfiguration". In: (2007) (cit. on p. 7).

[200] E. Rosen, A. Viswanathan, and R. Callon. *Multiprotocol label switching architecture*. Tech. rep. 2001 (cit. on p. 19).

[201] R. Rosen. “Resource management: Linux kernel namespaces and cgroups”. In: *Haifux, May* 186 (2013), p. 70 (cit. on p. 21).

[202] F. E. Ross. “An overview of FDDI: The fiber distributed data interface”. In: *IEEE Journal on Selected Areas in Communications* 7.7 (1989), pp. 1043–1051 (cit. on p. 13).

[203] A. Rowstron and P. Druschel. “Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems”. In: *Middleware 2001: IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12–16, 2001 Proceedings 2*. Springer. 2001, pp. 329–350 (cit. on p. 5).

[204] M. Rubenstein, A. Cornejo, and R. Nagpal. “Programmable self-assembly in a thousand-robot swarm”. In: *Science* 345.6198 (2014), pp. 795–799 (cit. on p. 1).

[205] F. Safari et al. “The diverse technology of MANETs: a survey of applications and challenges”. In: *International Journal of Future Computer and Communication (IJFCC)* 12.2 (2023) (cit. on p. 31).

[206] G. Samara et al. “Energy efficiency Wireless Sensor Networks Protocols: a Survey”. In: *2022 International Conference on Emerging Trends in Computing and Engineering Applications (ETCEA)*. IEEE. 2022, pp. 1–6 (cit. on pp. 2, 30, 31).

[207] V. Schiavoni, E. Rivière, and P. Felber. “SplayNet: Distributed User-Space Topology Emulation”. In: *Middleware 2013: ACM/IFIP/USENIX 14th International Middleware Conference, Beijing, China, December 9-13, 2013, Proceedings 14*. Springer. 2013, pp. 62–81 (cit. on p. 25).

[208] A. Schiper. “Dynamic group communication”. In: *Distributed Computing* 18 (2006), pp. 359–374 (cit. on p. 30).

[209] M. Schranz et al. “Swarm robotic behaviors and current applications”. In: *Frontiers in Robotics and AI* 7 (2020), p. 36 (cit. on p. 1).

[210] SETI@home Team. *SETI@home*. 2025. URL: `https://setiathome.berkeley.edu/` (visited on 2025-01-22) (cit. on p. 9).

[211] A. Shraer et al. “Dynamic {Reconfiguration} of {Primary/Backup} Clusters”. In: *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. 2012, pp. 425–437 (cit. on p. 25).

[212] D. Siemon. “Queueing in the Linux network stack”. In: *Linux Journal* 2013.231 (2013), p. 2 (cit. on p. 20).

[213] L. F. Sikos. “Packet analysis for network forensics: A comprehensive survey”. In: *Forensic Science International: Digital Investigation* 32 (2020), p. 200892 (cit. on p. 14).

[214] M. C. Silva Filho et al. “CloudSim plus: a cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness”. In: *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE. 2017, pp. 400–406 (cit. on p. 15).

[215] G. Sirbu and M. Leonardi. “Fully Autonomous Orbit Determination and Synchronization for Satellite Navigation and Communication Systems in Halo Orbits”. In: *Remote Sensing* 15.5 (2023), p. 1173 (cit. on p. 33).

[216] J. Šoltỳs. “Linux Kernel 2. 6 Documentation”. In: *Comenius University Faculty of Mathematics, Physics and Informatics. Bratislava* (2006) (cit. on p. 20).

[217] C. E. Spurgeon. *Ethernet: the definitive guide*. O’Reilly Media, Inc., 2000 (cit. on p. 13).

[218] R. Steinmetz and K. Wehrle. *Peer-to-peer systems and applications*. Vol. 3485. Springer, 2005 (cit. on p. 4).

[219] I. Stoica et al. “Chord: A scalable peer-to-peer lookup service for internet applications”. In: *ACM SIGCOMM Computer Communication Review* 31.4 (2001), pp. 149–160 (cit. on p. 5).

[220] B. Stroustrup. *The C++ Programming Language*. 4th. Addison-Wesley Professional, 2013. ISBN: 0321563840 (cit. on p. 12).

[221] D. Stutzbach and R. Rejaie. “Understanding churn in peer-to-peer networks”. In: *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*. 2006, pp. 189–202 (cit. on p. 4).

[222] D. Stutzbach et al. “On unbiased sampling for unstructured peer-to-peer networks”. In: *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*. 2006, pp. 27–40 (cit. on pp. 1, 2).

[223] W. Sun et al. “A survey of using swarm intelligence algorithms in IoT”. In: *Sensors* 20.5 (2020), p. 1420 (cit. on p. 1).

[224] L. Suresh et al. “Stable and consistent membership at scale with rapid”. In: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 2018, pp. 387–400 (cit. on p. 25).

[225] A. Takeda, T. Oide, and A. Takahashi. “A structured overlay network for aggregating sensor data”. In: *2012 Seventh International Conference on Broadband, Wireless Computing, Communication and Applications*. IEEE. 2012, pp. 684–689 (cit. on p. 5).

[226] C. Tang, R. N. Chang, and C. Ward. “GoCast: Gossip-enhanced overlay multicast for fast and dependable group communication”. In: *2005 International Conference on Dependable Systems and Networks (DSN’05)*. IEEE. 2005, pp. 140–149 (cit. on p. 27).

[227] *TaRDIS - Trustworthy and Resilient Decentralised Intelligence for Edge Systems*. 2025. URL: https://project-tardis.eu/ (visited on 2025-01-21) (cit. on p. 3).

[228] R. C. Tausworthe. “Random numbers generated by linear recurrence modulo two”. In: *Mathematics of Computation* 19.90 (1965), pp. 201–209 (cit. on p. 21).

[229] G. D. Team. *GNS3 - Graphical Network Simulator*. 2025. URL: https://gns3.com/ (visited on 2025-01-23) (cit. on p. 19).

[230] V. P. Team. *Virtual Networks over Linux (VNX)*. 2025. URL: https://web.dit.upm.es/vnxwiki/index.php/Main_Page (visited on 2025-01-23) (cit. on p. 19).

[231] The NAM Team. *Network Animator (NAM)*. 2025. URL: https://www.isi.edu/websites/nsnam/nam/ (visited on 2025-01-22) (cit. on p. 12).

[232] The Network Simulator (NS-2) Team. *Network Simulator 2 (NS-2)*. 2025. URL: https://www.isi.edu/websites/nsnam/ns/ (visited on 2025-01-22) (cit. on p. 12).

[233] The OTcl Team. *OTcl (Object-oriented Tcl)*. 2025. URL: https://otcl-tclcl.sourceforge.net/otcl/ (visited on 2025-01-22) (cit. on p. 12).

[234] R. Tian et al. "Hybrid overlay structure based on random walks". In: *Peer-to-Peer Systems IV: 4th International Workshop, IPTPS 2005, Ithaca, NY, USA, February 24-25, 2005. Revised Selected Papers 4*. Springer. 2005, pp. 152–162 (cit. on p. 5).

[235] M. A. To, M. Cano, and P. Biba. "DOCKEMU–A Network Emulation Tool". In: *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*. IEEE. 2015, pp. 593–598 (cit. on p. 22).

[236] *Universal TUN/TAP device driver*. Linux Kernel. 2025. URL: https://www.kernel.org/doc/Documentation/networking/tuntap.txt (visited on 2025-01-23) (cit. on p. 22).

[237] R. T. University. *Roma Tre University Official Website*. 2025. URL: https://www.uniroma3.it/en/ (visited on 2025-01-22) (cit. on p. 19).

[238] P. J. Van Laarhoven et al. *Simulated annealing*. Springer, 1987 (cit. on p. 16).

[239] R. Van Renesse and D. Altinbuken. "Paxos made moderately complex". In: *ACM Computing Surveys (CSUR)* 47.3 (2015), pp. 1–36 (cit. on p. 9).

[240] A. Varga and R. Hornig. "An overview of the OMNeT++ simulation environment". In: *1st International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems*. 2010 (cit. on pp. 12, 13, 17).

[241] N. Varis. "Anatomy of a Linux bridge". In: *Proceedings of Seminar on Network Protocols in Operating Systems*. Vol. 58. 2012 (cit. on p. 23).

[242] J. S. Varma et al. "Scalable, Fault-Tolerant Membership for Group Communication on HPC Systems". In: (2006) (cit. on p. 25).

[243] K. V. Vishwanath et al. "Modelnet: Towards a datacenter emulation environment". In: *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*. IEEE. 2009, pp. 81–82 (cit. on p. 25).

[244] E. Vlahakis, N. Athanasopoulos, and S. McLoone. “Aimd scheduling and resource allocation in distributed computing systems”. In: *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE. 2021, pp. 4642–4647 (cit. on p. 1).

[245] J. Von Mulert, I. Welch, and W. K. Seah. “Security threats and solutions in MANETs: A case study using AODV and SAODV”. In: *Journal of Network and Computer Applications* 35.4 (2012), pp. 1249–1259 (cit. on p. 31).

[246] S. Voulgaris, D. Gavidia, and M. Van Steen. “Cyclon: Inexpensive membership management for unstructured p2p overlays”. In: *Journal of Network and systems Management* 13 (2005), pp. 197–217 (cit. on pp. 5, 26).

[247] C. A. Waldspurger. “Memory resource management in VMware ESX server”. In: *ACM SIGOPS Operating Systems Review* 36.SI (2002), pp. 181–194 (cit. on p. 22).

[248] G. Wang, J. Cao, and K. C. Chan. “RGB: A scalable and reliable group membership protocol in mobile Internet”. In: *International Conference on Parallel Processing, 2004. ICPP 2004.* IEEE. 2004, pp. 326–333 (cit. on p. 25).

[249] B. B. Welch, K. Jones, and J. Hobbs. *Practical programming in Tcl and Tk*. Prentice Hall Professional, 2003 (cit. on p. 21).

[250] P. Wette et al. “Maxinet: Distributed emulation of software-defined networks”. In: *2014 IFIP Networking Conference*. IEEE. 2014, pp. 1–9 (cit. on p. 24).

[251] World Wide Web Consortium (W3C). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C Recommendation. 2008. URL: `https://www.w3.org/TR/xml/` (visited on 2025-01-22) (cit. on p. 13).

[252] Wuba. *Wpaxos*. A Production-Grade Java Implementation of Paxos. 2020. URL: `https://github.com/wuba/WPaxos` (visited on 2025-01-22) (cit. on p. 10).

[253] W. Xia et al. “A survey on software-defined networking”. In: *IEEE Communications Surveys & Tutorials* 17.1 (2014), pp. 27–51 (cit. on p. 19).

[254] L. Xu, K. Harfoush, and I. Rhee. “Binary increase congestion control (BIC) for fast long-distance networks”. In: *IEEE INFOCOM 2004*. Vol. 4. IEEE. 2004, pp. 2514–2524 (cit. on p. 20).

[255] Yixin Luo. *Mypaxos*. A Paxos Implementation in Java. 2017. URL: `https://github.com/luohaha/MyPaxos` (visited on 2025-01-22) (cit. on p. 10).

[256] B. Yost and S. Weston. *State-of-the-art small spacecraft technology*. Tech. rep. 2024 (cit. on p. 31).

[257] D. Yu et al. “Dynamic adaptation in wireless networks under comprehensive interference via carrier sense”. In: *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2017, pp. 337–346 (cit. on p. 2).

[258] S. Zeadally, H. Moustafa, and F. Siddiqui. “Internet protocol television (IPTV): architecture, trends, and challenges”. In: *IEEE Systems Journal* 5.4 (2011), pp. 518–527 (cit. on p. 4).

[259] M. Zec and M. Mikuc. "Operating system support for integrated network emulation in imunes". In: *1st Workshop on Operating System and Architectural Support for the On Demand IT Infrastructure (OASIS)*. 2004, pp. 3–12 (cit. on p. 21).

[260] Y. Zeng, R. Zhang, and T. J. Lim. "Wireless communications with unmanned aerial vehicles: Opportunities and challenges". In: *IEEE Communications magazine* 54.5 (2016), pp. 36–42 (cit. on p. 1).

[261] Y. Zhang et al. "Coverage enhancing of 3D underwater sensor networks based on improved fruit fly optimization algorithm". In: *Soft Computing* 21 (2017), pp. 6019–6029 (cit. on p. 1).

[262] W. Zhao, P. Melliar-Smith, and L. E. Moser. "Fault tolerance middleware for cloud computing". In: *2010 IEEE 3rd International Conference on Cloud Computing*. IEEE. 2010, pp. 67–74 (cit. on p. 25).