Journal of Software Engineering Research and Development, 2025, 13:4, doi: 10.5753/jserd.2025.4917

Software testing for peer-to-peer systems: Challenges and the state-of-the-art

Bruno E. R. Garcia [**ICMC/USP** | ***bruno.erg@usp.br***]
Simone R. S. Souza [**ICMC/USP** | ***srocio@icmc.usp.br***]

Abstract

Peer-to-peer (P2P) systems, characterized by their decentralized architecture, have become crucial to various applications such as file sharing, content distribution, and blockchain technologies. Due to their dynamic and heterogeneous nature, ensuring their reliability and robustness presents challenges. Such an issue has motivated researchers to develop testing techniques for these systems. Despite advances in research about software testing for distributed systems, there is a lack of data about their applicability and which features from P2P systems are highlighted by the proposed testing approaches. This paper provides a comprehensive overview of the state-of-the-art software testing for P2P systems. We systematically examine the inherent challenges and situations. Our findings show that some aspects of P2P systems can make the software testing activity difficult, such as validation of global properties and consensus, peer churn, a large volume of data, malformed data, reproducibility, and non-determinism. Several testing techniques have been used in this context, such as model-based testing, fault injection, fuzzing, stress testing, differential testing, conformance testing, and concurrency testing. We highlight fuzzing for testing malformed data, global properties, parallel events, and model-based testing as a powerful technique to abstract the complexities and varieties of P2P applications during the testing activities. Moreover, blockchain applications are the most addressed P2P application explored by the studies, where most papers were published in 2023, highlighting the theme's relevance.

Keywords: *peer-to-peer, software testing, distributed systems, systematic mapping*

1 Introduction

Peer-to-peer (P2P) systems are part of a distributed network where each participant (node) has equal privileges and responsibilities. Unlike traditional client-server models, in P2P systems, each node can act both as a client and a server, sharing resources directly with other nodes without the need for a central coordinator or server. Also, Schollmeier (2001) points out that a P2P network is a distributed network where the peers share their own computing resources, such as processing and storage.

P2P systems have emerged as a fundamental architecture for distributed computing, enabling decentralized communication and resource sharing among interconnected systems (Fox, 2001). These systems represent a paradigm shift from traditional client-server models, empowering users to directly interact and collaborate without relying on centralized infrastructure. The pervasive adoption of P2P technology has fueled its application across a broad spectrum of domains, including file sharing, content delivery networks, communication platforms, and blockchain applications, among others (Steinmetz and Wehrle, 2005a).

The significance of P2P systems lies in their ability to distribute tasks and data across a network of interconnected nodes, enhancing scalability, fault tolerance, and efficiency (Steinmetz and Wehrle, 2005b). However, this decentralized nature also introduces unique challenges, particularly in software testing. Due to their distinctive characteristics, traditional testing methodologies designed for centralized architectures often must catch up when applied to P2P systems.

One of the primary challenges in testing P2P systems is the network's dynamic and heterogeneous nature. Nodes in a P2P network can join or leave dynamically, leading to fluctuations in network topology and connectivity (Imtiaz et al., 2019). This dynamism introduces uncertainties and makes replicating real-world scenarios in testing environments challenging. Moreover, the diversity of nodes, including hardware, software, and configuration variations, further complicates the testing process.

Another critical aspect that adds complexity to P2P system testing is the inherent lack of central control (Steinmetz and Wehrle, 2005b). Unlike centralized systems, where a single entity manages resources and access, P2P networks operate distributedly, with each node making autonomous decisions. This decentralization raises coordination, synchronization, and consensus issues, which must be thoroughly tested to ensure system reliability and performance.

In this paper, we investigate the existing contributions of software testing for P2P systems, highlighting the challenges faced by studies, and examining the state-of-the-art testing techniques and approaches developed to address the features of P2P systems. This study presents evidence of which characteristics of these systems are explored by proposed testing techniques, mapping the defined testing approaches to observe gaps to be investigated in further works.

The remainder of the paper is organized as follows. Section 2 provides an overview of the related work. Section 3 presents the methodology we used for this study. Section 4 presents the challenges and nuances of testing P2P systems. Section 5 analyzes the testing techniques addressed by the studies. Section 6 analyzes the applications covered by the articles. Section 7 explores the open issues and future research directions. Section 8 presents the threats to validity. Finally, Section 9 presents the conclusion and future work.

2 Related work

In our investigation of software testing methodologies within peer-to-peer (P2P) systems, we identified related works in the literature. One study, referenced as (Cvitic et al., 2023), examines the software testing practices implemented across 18 open-source projects in distributed blockchain/ledger environments. The authors observed that unit and integration tests are present in most projects. However, they do not follow a holistic system testing approach. Another interesting finding is that most of the projects rely on a small group of contributors.

Further exploration was conducted by Arsat et al. (2022) and Reddivari et al. (2023), who each performed literature reviews focusing on systems testing specifically within blockchain technologies. The work by Reddivari et al. (2023) is a preliminary literature review and introduces a framework for conducting an empirical study on available data about it from *GitHub*. They found 17 studies that include an empirical study on a tool developed for blockchain testing. In contrast, the study by Arsat et al. (2022) concentrates on three critical aspects: performance, privacy, and smart contracts. Their methodology included developing targeted search strings for each aspect, leading to the analysis of approximately 36 pertinent articles. Notably, most of these articles primarily addressed issues related to smart contracts.

Additionally, Ahmad et al. (2017) delve into testing methodologies involving cloud computing, which includes discussions on P2P systems and how they leverage cloud computing for testing. Lastly, the study by Pankov (2020) examines verification, validation, and testing practices within distributed ledger systems. This research highlights the wide range of tools available for these purposes but also points out the lack of standardized specifications and approaches in the field.

All works presented in this section are not specific to P2P systems; they only cover or study a specific application of this system type and are not focused on specific characteristics of P2P systems. Our study focuses on the specific features of P2P systems and the existing software testing approaches defined for them.

3 Methodology

This section presents the methodology for conducting a systematic mapping of software testing for Peer-to-Peer (P2P) systems.

We first defined the research questions that will be used to guide our study and then the inclusion and exclusion criteria. Second, we defined the search string. To do it, we created a control group consisting of diverse and relevant studies for our mapping that is used to validate our search string. The search string was defined based on common terminologies of this area and the ones the control group approached. We expected to find all the studies from our control group using our search string. With the search string, we searched for the articles in the databases and filtered the relevant papers by reading the title and abstract and then the full text. With all selected studies, we applied snowballing to find new, interesting ones. We used the snowballing process backward and forwardly for every paper we characterized as attractive from the full-text reading. We repeated it until we could not find any new interesting study.

The methodology outlined here encompasses the planning phase, including defining research questions, inclusion criteria, and the execution phase, involving literature search, screening, data extraction, and synthesis. By adhering to rigorous methodological standards, as described by Petersen et al. (2008), we aim to provide a comprehensive overview of existing knowledge in software testing for P2P systems, facilitating informed decision-making and advancing research in this domain.

The structured approach adopted in this methodology ensures transparency, reproducibility, and reliability in the mapping review process. It enables researchers and practitioners to navigate the vast landscape of literature related to P2P system testing, identify key themes, trends, and gaps, and gain insights into the diverse methodologies, tools, and strategies researchers employ in this area.

Through the systematic mapping of existing literature, this methodology seeks to contribute to a deeper understanding of the challenges, advancements, and best practices in software testing for P2P systems, ultimately fostering the development of more effective and robust testing methodologies tailored to the characteristics of peer-to-peer architectures.

3.1 Research questions

This systematic mapping aims to answer the following research questions:

- (**RQ1**) *What are the primary challenges and nuances of P2P systems that are being addressed or studied in the context of software testing?*
- (**RQ2**) *Which software testing techniques, tools, or approaches have been applied in the context of P2P systems?*
- (**RQ3**) *Which applications of P2P systems are most covered in the literature?*

3.2 Inclusion and exclusion criteria

The studies must meet one of the inclusion criteria and none of the exclusion criteria.

Inclusion criteria:

- (**IC1**) Proposes a new approach or testing technique for P2P systems.
- (**IC2**) Evaluates or compares existing testing techniques and approaches for P2P systems or evaluates how systems testing is approached in P2P systems.

Exclusion criteria:

- (**EC1**) Not written in English or Portuguese.
- (**EC2**) Not peer-reviewed.
- (**EC3**) Published more than a decade ago.
- (**EC4**) Proposes a testing approach or technique, but it is not for a primary aspect of P2P system, such as scalability, volatility, global properties, etc.

- (**EC5**) Proposes only a simulator/testbed and does not present a new testing approach or technique.

About the **EC4**, it is important to highlight that there are a lot of studies that use P2P systems in their experiments. However, their proposal or focus does not involve aspects of P2P systems. We want to analyze studies in which findings and proposals can be replicated in any P2P system or studies which analyzes the state of testing of P2P systems, that is why Jain et al. (2023) was considered in this study. For example, there are many studies that explore mutation testing for the Solidity language. Solidity is a language designed for creating smart contracts on the Ethereum blockchain. Even though it is something under a P2P protocol, the applicability of the study does not depend on the characteristics of P2P systems themselves and may not be useful for other applications.

3.3 Search sources

Table 1 presents the search sources used in this systematic mapping, chosen based on their relevance to the area of computing.

Table 1. Database to search for articles and their respective source

Database	Source
IEEE Xplore	`https://ieeexplore.ieee.org/`
ACM	`https://dl.acm.org/`
Science Direct	`https://www.sciencedirect.com/`
Wiley	`https://onlinelibrary.wiley.com/`

3.4 Search string

A control group consisting of four papers was used to create the search string, as shown in Table 2. These papers were chosen based on the diversity of *keywords* that cover the peer-to-peer topic. For example, two papers address the topic of blockchain; one addresses a peer-to-peer mobile system, and, finally, one paper addresses peer-to-peer networks using terms such as *"Network applications"*. All papers were chosen based on their publication date; therefore, they are recent studies.

Based on the control group and the review's objective, the search string was created considering terms directly or indirectly referring to P2P systems:

(**"peer-to-peer system"** OR **"peer-to-peer application"** OR **"peer-to-peer app"** OR **"peer-to-peer software"** OR **"peer-to-peer network"** OR **"peer-to-peer architecture"** OR **"P2P architecture"** OR **"P2P system"** OR **"P2P application"** OR **"P2P app"** OR **"P2P software"** OR **"P2P network"** OR **"distributed ledger"** OR **"blockchain"** OR **"DHT"**) AND (**"test*"**)

3.5 Selection and snowballing steps

In total, 2507 papers were returned by search string applied in all digital libraries, considering title and abstract. Table 3 shows the number of articles found per library.

Table 3. Database to search for articles and their respective source

Database	Number of articles
IEEE Xplore	1664
ACM	92
Science Direct	679
Wiley	72

Figure 1 illustrates the sequential steps involved in article selection. Following the initial screening, we identified 2507 articles filtered out based on duplication and alignment with our inclusion and exclusion criteria. This process involved an initial review of article titles and abstracts, followed by a detailed examination of the full text where necessary. Finally, 21 articles were deemed suitable for inclusion in our study.

We employed a snowballing technique with the 21 selected primary studies. This method involves expanding the pool of articles by scrutinizing the references and citations of each primary study. If we encountered a relevant article meeting our research questions and criteria during this analysis, we repeated the process of examining its references and citations. This iterative process continued until no further studies meeting our criteria were discovered. Consequently, we identified and included 5 additional studies, bringing the total to 26 articles in our study. Besides that, we applied snowballing in the papers mentioned in Section 2, however, no new papers were found considering the ones we had already selected or discarded.

It is important to highlight that only two researchers (both authors) participated in this study. One researcher (first author) was responsible for executing each stage of the protocol, including searching, evaluating, and selecting studies. The other researcher (the last author) performed a thorough review of the selected studies and classifications to ensure accuracy and consistency. The primary researcher reviewed and classified the studies according to the protocol criteria. The secondary researcher reviewed the classified studies, providing a validation step to confirm the initial classifications. Since only two people were involved, we did not have problems with consensus about the review process.

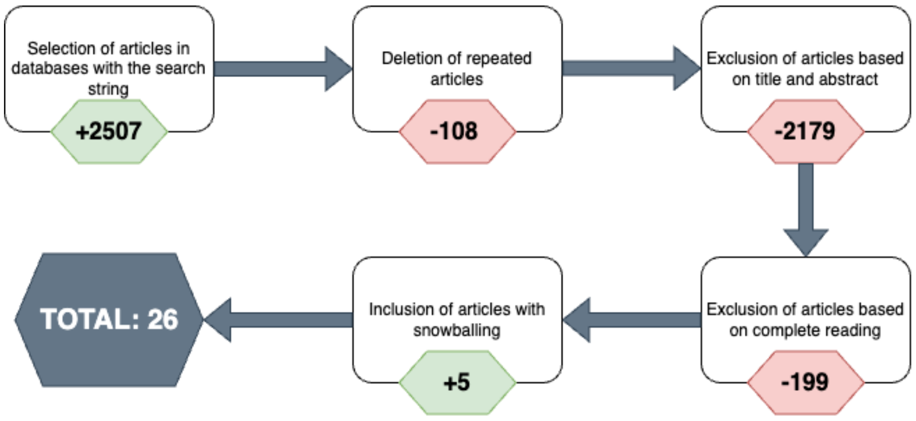
4 RQ1 - What are the challenges and nuances of P2P systems that are being addressed or studied in the context of software testing?

To delve into the challenges of testing P2P systems and the proposed testing solutions, we mapped articles that specifically address or suggest testing methodologies tailored to the characteristics and nuances of P2P systems. We identified these challenges by reading the motivation of the proposal and grouped them according to their characteristics. Table 4 provides an overview of these challenges alongside the corresponding articles. Subsequently, we will elaborate on each challenge in detail.

Table 2. Studies from the control group

Title	Published at	Year
Evolutionary Approach for Concurrency Testing of Ripple Blockchain Consensus Algorithm (van Meerten et al., 2023)	IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)	2023
LOKI: State-Aware Fuzzing Framework for the Implementation of Blockchain Consensus Protocols (Ma et al., 2023a)	Network and Distributed System Security Symposium	2023
ANDROFLEET: Testing WiFi peer-to-peer mobile apps in the large (Meftah et al., 2017)	32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)	2017
NetLoiter: A Tool for Automated Testing of Network Applications using Fault-injection (Rozsíval and Smrčka, 2023)	53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)	2023

Figure 1. Steps to select the articles


Selection of articles in databases with the search string
+2507
Deletion of repeated articles
-108
Exclusion of articles based on title and abstract
-2179
Exclusion of articles based on complete reading
-199
Inclusion of articles with snowballing
+5
TOTAL: 26

4.1 Validation of global properties and Consensus

Global properties in P2P systems are attributes or characteristics applicable to all nodes within the network. Similarly, consensus in a P2P network refers to the process by which nodes in a decentralized network agree on the current state of the network. The literature indicates difficulties in validating these global properties during testing activities. According to Sunyé et al. (2014), validating global properties necessitates collecting data from distributed nodes and aggregating it for analysis to obtain an overview of the properties. However, this task is challenging due to the dynamic nature of the systems and local properties that can impact global properties. Therefore, it is essential to update this ”overview frequently”.

Another challenge in testing global properties is performing the tests without disturbing the system (Machado et al., 2020). According to Machado et al. (2020), such disturbance can interfere with the system under test and the results. Ma et al. (2023a) indicate that Blockchain consensus protocols are responsible for coordinating nodes to achieve consensus on transactions occurring within the network. In other words, the consensus among network nodes defines the network's global properties. However, according to the authors, existing fuzz testing tools cannot handle the consensus state among network nodes. The generated data is mostly useless and hinders the testing's effectiveness in precisely targeting the consensus protocols' logic.

Also, there is a lack of both precise detectors and efficient test case generation for consensus bugs in blockchain applications (Chen et al., 2023). Most tools identify bugs by monitoring whether the target program exits normally. However, they can not detect these bugs that violate consensus properties of blockchain systems, for example, by erroneously processing blocks without crashing the nodes (Chen et al., 2023).

4.2 Volatility

In a P2P network, nodes can join or leave at any time. This churn can happen for various reasons, such as network instability, node failures, voluntary disconnections, or new nodes joining the network. The literature contains studies analyzing the impact of volatility on P2P networks. For example, in the context of the Bitcoin P2P network, several studies measure node volatility and assess its impact. These articles and their respective contributions are summarized in Table 5.

The studies present data measuring node volatility in the network, evaluating the percentage of nodes that join and leave the network daily, and analyzing how this volatility can affect system performance. The data presented by the articles are similar. The daily rate of nodes entering and leaving the network ranges from approximately 6% to 9%, and most nodes must deal with volatility in their connections. Additionally, two studies indicate that this volatility negatively impacts the network by delaying block propagation times and affecting network synchronization (Saad et al., 2021) (Motlagh et al., 2020).

Concerning software testing, studies indicate that consid-

Table 4. Challenges and nuances of P2P systems addressed by each article

Challenges	Studies
Validation of global properties and consensus	Sunyé et al. (2014) Machado et al. (2020) Ma et al. (2023a) Chen et al. (2023)
Volatility	Shala et al. (2017) Meftah et al. (2017) Milka and Rzadca (2018)
Large volume of data	Aziz et al. (2020) Jabbar et al. (2020) Adamsky et al. (2021)
Malformed data	Groce et al. (2022) Winter et al. (2023) Veldkamp et al. (2023) Wang et al. (2024)
Reproducibility and non-determinism	Amaral et al. (2020) Kim and Hwang (2023) Jabbar et al. (2020)
Network/environment errors or limitations	Dinesh et al. (2019) Rozsíval and Smrčka (2023) Ma et al. (2023b) Boumaiza and Sanfilippo (2024a) Chen (2024) Boumaiza and Sanfilippo (2024b) Hassanzadeh-Nazarabadi et al. (2023)
Multiple implementations	Chepurnoy and Rathee (2018) Kim and Hwang (2023) Chen et al. (2023) Yang et al. (2021)
Concurrency and parallelism	Machado et al. (2020) Adamsky et al. (2021) van Meerten et al. (2023)

Table 5. Studies on the effect of volatility in the Bitcoin P2P Network

Article	Results
Imtiaz et al. (2019)	About 97% of nodes have to deal with volatility.
Motlagh et al. (2020)	Network volatility increases block propagation time by an average of 135%.
Eisenbarth et al. (2021a)	The daily rate of nodes joining and leaving the network is about 6%.
Saad et al. (2021)	The daily rate of nodes entering and leaving the network is about 8%, impacting network synchronization.
Li et al. (2023)	The daily rate of nodes entering and leaving the network can reach 9%.

ering this characteristic in a testing approach is important since volatility can impact system behavior (Shala et al., 2017) (Milka and Rzadca, 2018). Despite the volatility present in the network, Blockchain applications must be resilient (Ma et al., 2023b). Due to errors and vulnerabilities, many Blockchain applications fail to recover from certain situations, such as node failures, leading to service interruptions. The authors classify these vulnerabilities as ”resilience issues”. Their approach to testing these errors involves identifying the most common types of resilience problems and injecting them during testing to evaluate the nodes' behavior in such situations.

4.3 Large Volume of Data

A P2P system must be capable of processing and often storing large data. This data can include message exchanges between nodes, data storage such as keys (in the case of DHT protocols), or transactions and blocks (in the case of Blockchain systems). Aziz et al. (2020) mention that with the increase in the number of transactions in a blockchain network, many nodes may not be able to handle this large volume of data and may crash during peak situations. For these reasons, considering data volume in the testing activity of P2P systems is important. Besides that, the large volume of data combined with the dynamism of the network can hinder testing activity, so it is important to combine functional and load aspects in the testing activity (Jabbar et al., 2020).

Also, some articles propose the use of stress testing. A systematic review of blockchain system testing indicated that about 27.78% of studies in this area focus on performance testing. Excluding studies on smart contracts, which are programs that automatically execute the terms of a digital contract based on blockchain when predetermined conditions are met, this number can increase Arsat et al. (2022).

4.4 Malformed data

In a P2P network, a node cannot predict the messages it will receive from other nodes, either in the sequence of the messages or their content. Additionally, it must be considered that a node may receive malformed messages from other nodes, meaning messages that deviate from the P2P network protocol standards. These messages may be generated due to system defects or by malicious agents attempting, for example, a denial-of-service (DoS) attack. DoS refers to an attack that aims to make a computer, network, or service unavailable to its intended users by overwhelming it with a flood of illegitimate requests (Long and Thomas, 2001). For this reason, it is important to test the behavior of a P2P system with data and messages outside the protocol standard (Groce et al., 2022) (Veldkamp et al., 2023) (Winter et al., 2023).

According to Wang et al. (2024), testing the security of BFT protocols often requires manual operations and there is a lack of automated testing tools for security purposes that allow to test scenarios with malformed, delayed and duplicated messages.

In summary, considering unpredictable message sequences and contents and malformed messages in testing activities is crucial for P2P systems to ensure robustness and security against potential faults and attacks.

4.5 Reproducibility and non-determinism

During the testing phase, when faults are revealed manually or through automated techniques, it is crucial to reproduce the fault to investigate the failure. According to Amaral et al. (2020), distributed systems are inherently complex, with various components interacting, running in heterogeneous and varied environments, and therefore subject to various failures. The authors also point out that orchestration platforms, containers, and virtual machines do not solve the reproducibility problem as they do not capture events related to the system's dynamic properties, such as node volatility. For this reason, the authors propose a fault injection testing tool that allows for writing tests using a specific language and considering the system characteristics, environment, workload, and fault pattern. In other words, with a single language, it is possible to control components that are not part of the system under test but can impact it. Without control over the environment and workload, tests risk suffering from a lack of determinism.

Besides that, the study by Kim and Hwang (2023) indicates that protocol specifications are not always clear. For the case study, there was a lack of clarification regarding handling non-deterministic events. Differential testing is the method found by the authors to deal with these events and test such behaviors. Similarly, Machado et al. (2020) present the idea that non-deterministic events hinder the testing activity because they increase the effort required to verify potential errors.

4.6 Network/environment errors or limitations

P2P networks are subject to environmental limitations and errors. In the "real world", when two or more systems communicate, there can be packet loss, delays, corruption, etc (Rozsíval and Smrčka, 2023). These failures may not be caused by the system but can still impact it. Additionally, Boumaiza and Sanfilippo (2024a) highlight the importance of testing a P2P system considering the practical topology. In other words, nodes in a P2P network can be distributed across different locations, affecting their performance.

Furthermore, network limitations must also be considered. For example, when a P2P network is structured under a 5G network, it is necessary to test the systems within the limitations imposed by the network (Dinesh et al., 2019). In this context, Dinesh et al. (2019) present a conforming testing approach. Conformance testing verifies whether the system conforms to specific standards and specifications. They reflect on a conformance testing approach for blockchain applications for IoT on 5G to test if the communication between nodes complies with the protocol standards of this network. They point out that adopting blockchain in IoT is not straightforward and requires many issues to be addressed, such as the overhead traffic created by blockchain protocols.

Chen (2024) point out that delays are inevitable in distributed networks and can affect system performance. However, improper handling of timeouts can lead to interruptions or system crashes. Therefore, the authors present a fuzz testing tool capable of simulating and injecting delays into the application under test.

Boumaiza and Sanfilippo (2024b) explore blockchain-based energy trade microgrids applications. The authors points out that it is fundamental to test the behavior of this kind of system considering the geographic location of every peer.

4.7 Multiple implementations

In a P2P network, nodes do not always run the same system or version. That is, for a given protocol, there can be multiple implementations. Given this characteristic, testing these different implementations together can be important. Differential testing has been explored in this context to achieve this. Differential testing involves testing two or more different applications or versions of the same application with the same inputs and checking for discrepancies between the outputs Gulzar et al. (2019). Kim and Hwang (2023) applied differential testing to different implementations of Ethereum. The authors' approach consists of generating a non-deterministic chain with concurrent transactions and propagation delays, then applying "property-based generation and type-preserving mutation" to generate semantically valid and invalid (but executable) test cases. The tool they created executes the test cases on the different nodes and evaluates the returned values. Using this technique, they discovered 11 defects in different implementations.

The Bitcoin protocol, for example, has more than one implementation, such as Bitcoin Core in C++, btcd in Golang, bitcoinj in Java, and bcoin in JavaScript (Eisenbarth et al., 2021b). However, according to the study by Groce et al. (2022), Bitcoin Core is a reference for other implementations. In other words, the protocol does not have a well-defined specification; therefore, one implementation is used as the specification itself. Because of this, the authors suggest that applying differential fuzzing between these different implementations can be promising.

4.8 Concurrency and Parallelism

As previously mentioned, predicting the volume of messages and requests a node from a P2P network may receive is impractical. A node may receive requests in parallel and concurrently, and these characteristics should be considered in testing. van Meerten et al. (2023) point out that blockchain systems are prone to concurrency errors due to the lack of determinism in the order of message delivery between nodes.

In 2023, a P2P network-related error was found in Bitcoin Core (the reference implementation of the Bitcoin protocol) [1]. When two nodes had a significant amount of messages to send to each other in parallel, a deadlock could occur because

[1]`https://github.com/bitcoin/bitcoin/pull/27981`

both nodes would "stop" receiving any traffic in the socket buffer until the message was fully sent. In this case, receiving would only resume when the message was sent, but this condition would never occur because both nodes were stuck waiting for the complete message to be sent. After the bug fix, a functional test was added to the project to test the sending of large messages between two nodes in parallel [2].

The article by Adamsky et al. (2021) is focused on security aspects and points out that there is a gap in DHT testing because most tests are conducted with few nodes. They highlight the importance of testing concurrent requests in a scenario with many nodes to reflect the reality.

Also, Jabbar et al. (2020) point out that the dynamism of the network makes the test activity costly. It is hard to validate properties when there is a great number of interacting and concurrent components.

5 RQ2 - Which software testing techniques, tools, or approaches have been applied for the context of P2P systems?

Figure 2 shows the relationship between the number of articles by testing approach/technique and their corresponding application with the challenges related to P2P system testing. As seen in the image, various techniques/approaches are used in this context, which we will delve into further below.

5.1 Fuzzing

Fuzz testing, or fuzzing, emerged as a technique in the late 1980s when Professor Barton Miller and his colleagues at the University of Wisconsin-Madison experimented to test the robustness of UNIX utilities by providing random input Miller et al. (1990). The concept is simple yet powerful: fuzz testing automatically generates many random, invalid, or unexpected inputs to a software program to uncover vulnerabilities, crashes, or unexpected behaviors. The fuzzing process typically includes a fuzzer tool that creates inputs and feeds them to the program, monitors the program's behavior for signs of failure, and logs any anomalies for further analysis. Modern fuzzing tools often employ sophisticated algorithms to generate inputs that are more likely to reveal vulnerabilities, including mutation (modifying existing inputs) and feedback-driven fuzzing (using runtime information to guide input generation).

One of the most used techniques for fuzzing is coverage-guided fuzzing. It uses code coverage information to enhance the efficiency and effectiveness of the fuzzing process. In this method, the fuzzer generates inputs and monitors which parts of the code are executed with each input. The goal is to maximize code coverage, ensuring that as much of the code as possible is tested, including edge cases and rarely executed paths. By increasing code coverage, this approach improves the likelihood of discovering hidden bugs and vulnerabilities.

For P2P systems, this testing method has been used to test, among other characteristics, whether any malformed/unexpected messages can reveal a bug in the program. The tool called *Byzzfuzz*, proposed by Winter et al. (2023), is a randomized testing tool that introduces mutations that alter the content of protocol messages by making small changes to the original message, either in value or timing (e.g., repeating a previous message). The mutation approach applied in the tool was inspired by fuzzing techniques. With this tool, the authors could discover several bugs in a Practical Byzantine Fault Tolerance (PBFT) implementation. PBFT is an algorithm used in distributed systems to achieve consensus despite the presence of faulty or malicious nodes. It has been used in blockchain applications as well as IoT ones Li et al. (2021).

Fuzzing, in the context of P2P systems, is mostly used to test the behavior of P2P systems against a large and diverse number of messages, including malformed data that can crash a P2P system. However, we verified that fuzzing is one of the testing techniques that was used for diverse purposes. In addition to test malformed messages, fuzzing is used to test consensus algorithms and non-determinism issues.

In order to measure the efficiency of this technique to discover bugs and vulnerabilities, we analyzed the main contributions of the studies that explore fuzzing for P2P systems. Table 6 presents the articles and their main contributions and we can see that most studies related to blockchain applications have made strong contributions by finding several new bugs and critical vulnerabilities. One of the studies had 9 CVEs assigned (Ma et al., 2023a). CVE stands for *Common Vulnerabilities and Exposures*. It is a standardized identifier system used to catalog and reference publicly known information-security vulnerabilities and exposures [3]. Ma et al. (2023a) developed a fuzz testing tool called *LOKI*. This tool focuses on blockchain consensus protocols and can detect logical consensus errors. Briefly, *LOKI* can detect consensus states in real time by disguising itself as a network node. From this mapping, *LOKI* adaptively generates the targets, types, and contents of inputs according to the state model.

5.1.1 Analysis of the discovered vulnerabilities

By analyzing the studies from the Table 6, we could classify the vulnerabilities into three groups:

Integrity Vulnerabilities related to data integrity. Those that make nodes stop syncing with other nodes, cause unnecessary disconnections, node isolation, etc.

Liveness Vulnerabilities that impact the node liveness. Those that make nodes crash or get stuck.

Consensus Any vulnerability directly related to the network consensus. Those that make nodes violate the consensus rules.

Chen et al. (2023) discovered 20 vulnerabilities in different applications. Among them, we can highlight CVE-2022-26298 (Asynchronous sync procedures cause some propos-

[2]`https://github.com/bitcoin/bitcoin/pull/28287`

[3]`https://www.cve.org/`

Figure 2. Number of articles by software testing approach/technique and their application with challenges related to testing P2P systems

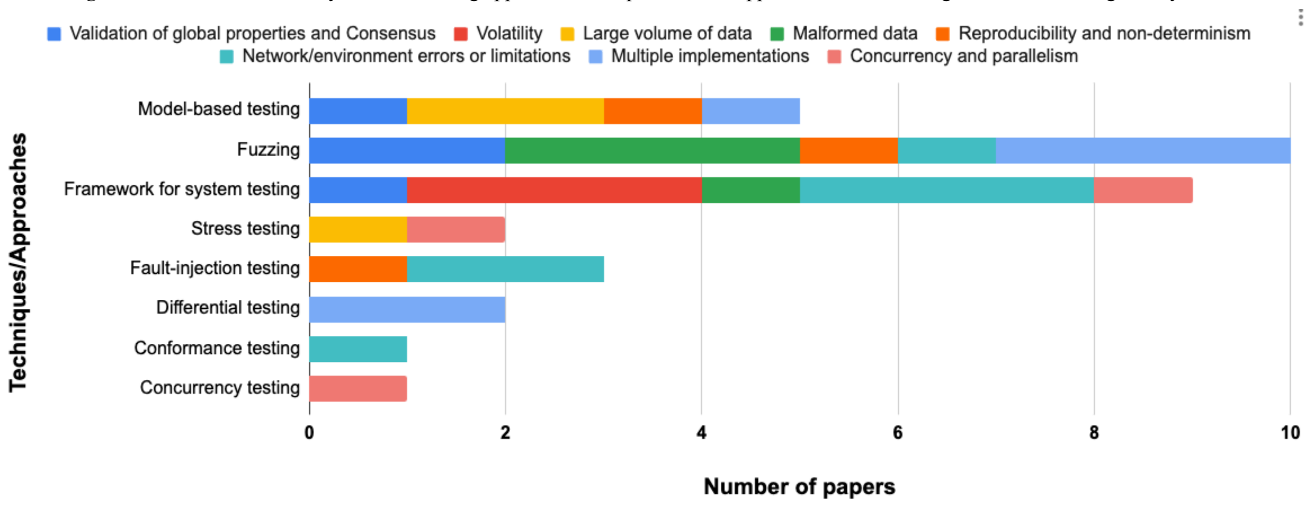


Table 6. Studies proposing or analyzing fuzzing for P2P systems

Article	System Under Test	Contributions/Results
(Yang et al., 2021)	Blockchain (Ethereum)	Two consensus bug found; the proposed tool improved fuzzing throughput by 510x and code coverage by 2.7x
(Groce et al., 2022)	Blockchain (Bitcoin)	Identified areas for improvement in Bitcoin Core fuzz testing.
(Veldkamp et al., 2023)	Blockchain (XRP)	The proposed approach significantly outperformed grammar-based fuzzing, covering 240 more branches.
(Chen et al., 2023)	Blockchain	Found 20 critical vulnerabilities.
(Winter et al., 2023)	Byzantine Fault Tolerant	Found several errors in the PBFT implementation, a potential activity violation in *Tendermint*, and discovered two vulnerabilities in Ripple's XRP Ledger consensus algorithm.
(Ma et al., 2023a)	Blockchain	Discovered 20 vulnerabilities with 9 CVEs assigned. Fourteen were memory-related errors, and six were consensus logic errors.
(Chen, 2024)	Blockchain	The proposed tool detected 27 timeout errors in the applications under test.

als to be double processed), which is related to the violation of the consensus rules; CVE-2022-26297 (Missing Deletion of in-flight when syncing past the in-flight sequence), which is related to the node integrity; and CVE-2022-26300 (The producer node crashes when generating a test account through the txn test gen plugin) which is a liveness vulnerability. The two vulnerabilities found by Yang et al. (2021) and the two ones found by Winter et al. (2023) are related to consensus violation.

5.1.2 Drawbacks

Even though fuzzing is a powerful technique to find bugs and vulnerabilities, we found two main drawbacks. First, fuzzing is good at exploring code paths with loose branch conditions, however, it struggles to drive the target program into paths with tight branch conditions (Yang et al., 2021). Also, when there are different implementations of the same protocol, differential fuzzing can be used to find bugs by analyzing discrepancies in these implementations. However, if all implementations contain the same bug, differential fuzzing will not be able to catch it. This scenario is common when applying differential fuzzing in two implementations and this risk can be minimized when applying it in three or more applications.

5.2 Model-based testing

Model-based testing is a software testing technique to generate test cases from models. The idea is to save time writing a few models that will achieve broad coverage of the input domain instead of writing a lot of test cases (Dalal et al., 1999). In the context of P2P systems, model-based testing is helpful to test global properties and consensus algorithms since there are a lot of possibilities of iteration between peers, which should not affect global properties or their consensus; however, it is impractical to cover them writing test case by test case.

Sunyé et al. (2014) propose a model-based testing approach to define a dynamic oracle that can verify global properties by performing a live analysis of the output of each node. This approach allows for the abstraction of relevant aspects of the system into models, which are then updated in real-time by monitoring the systems.

Jabbar et al. (2020) propose a model-based testing approach that combine both functional and load aspects. The authors say that the SUT may have distinct behaviors corresponding to different load levels.

5.2.1 Drawbacks

We could identify as drawbacks two points. First, both studies that propose a model-based testing approach are domain-specific and may have limited flexibility. Finally, no model can perfectly represent the entire behavior of a complex P2P system. If the model is incomplete or incorrect, the generated tests will be limited and may miss critical defects.

5.3 Framework for system testing

Two studies propose a framework for functional system testing to address the volatility of P2P networks. In Shala et al. (2017), the authors highlight the importance of considering volatility in test scenarios and propose a framework that allows the tester to control the volatility of individual nodes in test cases. The framework proposed by Milka and Rzadca (2018) also allows individual node control in test scenarios, emphasizing the importance of including volatility in testing activities. Both articles propose functional system testing frameworks that allow individual node control in test scenarios. Having this control is essential to write test cases where we can reproduce the iteration between the nodes including nodes joining and leaving the network during it.

Machado et al. (2020) propose a framework called *Minha*, which virtualizes multiple instances of Java Virtual Machine (JVM) within a single JVM, thus simulating a distributed environment where each host operates on a separate machine with dedicated network and CPU resources. This framework comprises several modules, one of which is responsible for monitoring and collecting data for offline use. Thus, global properties are validated through logs collected in real-time and analyzed in parallel.

Boumaiza and Sanfilippo (2024b) propose a framework for system testing that integrates a Geographic Information System (GIS) environment with an Agent-Based Modeling (ABM) simulation platform. The idea is to test energy trading frameworks in a realistic scenario considering that the peers are located in different countries.

Hassanzadeh-Nazarabadi et al. (2023) propose a framework for testing a BFT application that allows us to create a production environment to test and simulate some common attacks.

Finally, Wang et al. (2024) propose a framework for system testing for DHT protocols focused on security aspects. With their framework, it is possible to create test scenarios considering malicious behavior, for example: Delayed transmission, duplicate transmission, and feign death (when a node fails to send a message when the conditions for sending the message and triggering malicious behavior have been met but then sending the message when the conditions for triggering malicious behavior are no longer present Wang et al. (2024)).

5.3.1 Drawbacks

The first point we can notice is that the proposals have limited applicability beyond specific domains. For example, some frameworks are highly specialized, such as the GIS-ABM framework for energy trading or the DHT. Also, frameworks that rely on real-time monitoring and data collection, by analyzing large logs to validate global properties or understand node volatility can be time-consuming and may require specialized analysis tools.

5.4 Stress testing

Stress testing is a type of performance testing that evaluates how a system behaves under extreme conditions (Sommerville, 2011). Since a node in a P2P network is conditioned to have many connections and receive a large volume of messages, testing the performance of these systems under such conditions is crucial to ensure system reliability. Table 7 presents the articles and their respective types of systems under test, along with a brief description of the evaluation performed. Blockchain-related applications address stress testing in scenarios with different network configurations.

The article by Adamsky et al. (2021) propose a framework for testing DHT protocols under high concurrency volumes. According to the authors, there is a gap in DHT testing because most tests are conducted with few nodes. The proposed framework can conduct system test experiments on a single machine with up to 4000 nodes.

5.4.1 Drawbacks

Even if a system fails under stress, it is not always clear what should be improved. The studies do not explore exactly how the systems fail to deal with events under stress. Besides that, it is common for stress testing a significant resource usage.

5.5 Fault-injection testing

Fault-injection testing is a software testing technique used to assess the robustness and reliability of a system by deliberately introducing faults or errors into the system. The primary goal is to observe how the system behaves under adverse conditions and to ensure that it can handle unexpected situations gracefully (Hsueh et al., 1997). In the context of P2P systems, three studies approach this technique. Rozsíval and Smrčka (2023) propose a testing tool to test the application's behavior in unexpected network situations, such as data corruption and disconnections.

Amaral et al. (2020) propose a testing toolkit that allows one to create test scenarios controlling the system configuration, environment, and simulating faults.

Chen (2024) propose a technique that combines fault-injection with fuzzing to test timeout errors. The idea is to inject some known faults during the fuzzing campaign.

In addition, Ma et al. (2023b) propose a tool to detect and locate resilience issues in blockchain applications through context-sensitive chaos. The idea is injecting sensitive strategies into the proper positions in the source code of the blockchain system and check whether the system has a bug.

Table 7. Studies addressing the large volume of data in testing activities

Article	System Under Test	Evaluation
Aziz et al. (2020)	Blockchain/DLT (Stellar)	Behavior and performance of nodes under different network configurations (e.g., block size and transaction speed).
Yutia and Fathiyana (2022)	Blockchain/DLT (Hyperledger)	Performance evaluation of the REST API and the website interacting with a Hyperledger instance.
Adamsky et al. (2021)	DHT	Stress testing of DHT protocols under high concurrency volume.

5.5.1 Mutation testing

The testing suite of comparable blockchain applications was evaluated with mutation testing (Jain et al., 2023). In that study, the authors propose a new framework for reasoning about the extent, limits, and nature of a given testing effort by comparing the source code coverage and the mutation score. Even though it is not a study focused on aspects of P2P systems, it was included in the mapping because they analyzed well-tested code across comparable blockchain projects and showed that their metric can be used to evaluate testing efforts and how using only code coverage as a reference might lead to false impressions.

5.5.2 Drawbacks

Fault injection can lead to unpredictable failures, which can be difficult to control and analyze. This is especially true for systems with interconnected components, such as P2P systems, making root cause analysis challenging. Also, fault-injection testing often requires significant computational resources and network configurations to simulate realistic failures (e.g., network disconnections, data corruption). This can make the testing process costly, particularly in large-scale systems as P2P ones.

5.6 Differential testing

Differential testing involves testing two or more different applications or versions of the same application with the same inputs and checking for discrepancies between the outputs (Gulzar et al., 2019). Kim and Hwang (2023) applied differential testing to different implementations of Ethereum. The authors' approach consists of generating a non-deterministic chain with concurrent transactions and propagation delays, then applying ”property-based generation and type-preserving mutation” to generate semantically valid and invalid (but executable) test cases. The tool they created executes the test cases on the different nodes and evaluates the returned values. Using this technique, they discovered 11 bugs in different implementations. This technique is also applied in the tool proposed by Yang et al. (2021). In their study, the authors used differential fuzzing to find bugs related to the consensus algorithm in different implementations of Ethereum.

Another study also addresses differential testing, but critically (Chen et al., 2023). According to the authors, differential testing tools detect defects by comparing the results of different applications or versions of the same application. However, if both implementations have the same error, differential testing will not be able to detect it. Moreover, many protocols have only one implementation.

5.6.1 Drawbacks

As mentioned above, differential testing will not be able to detect bugs if both tested implementations have it. Besides that, slight differences in implementations, especially those arising from legitimate differences in design, can trigger false positives. These false positives require additional analysis to confirm or rule out genuine issues. For systems like P2P applications, non-determinism (such as transaction order or network timing) can affect test results.

5.7 Conformance testing

Conformance testing verifies whether the system conforms to specific standards and specifications. In this context, Dinesh et al. (2019) present a conforming testing approach for blockchain applications for IoT on 5G to test if the communication between nodes complies with the protocol standards of this network. They point out that adopting blockchain in IoT is not straightforward and requires many issues to be addressed, such as the overhead traffic created by blockchain protocols.

5.7.1 Drawbacks

There is only one study about conformance testing and we could identify one drawback about their proposal. Blockchain and IoT each have complex, evolving standards, especially in the context of 5G networks. Conformance testing requires detailed knowledge of these specifications, and adapting tests to new versions or standards can be time-consuming and complex.

5.8 Concurrency testing

Concurrency testing involves testing an application under concurrent requests and actions. van Meerten et al. (2023) proposed an evolutionary approach to testing the concurrency of a consensus algorithm in a blockchain by exploring different sequences of message delivery in the network. The fitness function for the evolutionary algorithm, responsible for evaluating each solution that performed the best, analyzed the execution time of each test case. In other words, concurrency errors are expected to cause longer-than-normal wait times. Ultimately, the authors revealed that they found a previously unknown error in the production implementation of Ripple (a blockchain application).

5.8.1 Drawbacks

In the proposal of van Meerten et al. (2023), we identified as a drawback the focus on execution time as the primary metric. It may miss other important factors affected by concurrency, such as memory usage or system state consistency. This narrow focus may lead to partial insights that do not fully assess concurrency-related robustness.

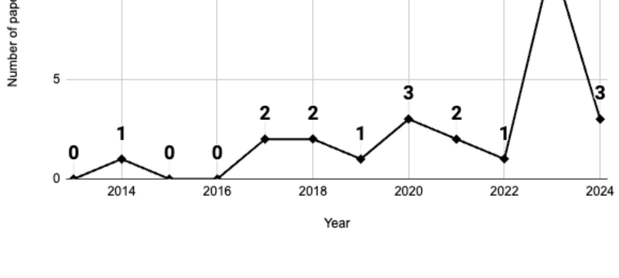
5.9 Testing at scale beyond the number of nodes

The test of P2P systems at scale is implicitly approached in some studies that we showed in previous sections. Although it is common to think that testing P2P systems at scale means only simulating a large number of nodes, it is required to consider real-world scenarios and conditions. Section 5.3 presents frameworks for system testing that besides allowing to create scenarios with a large number of nodes, allow to simulate node's churn. Section 5.5 presents fault-injection testing for P2P systems and some studies address the scalability in the tests. That is, the topic of "testing at scale" is implicit in many studies presented in previous sections. Besides allowing the creation of test scenarios with a large number of nodes, frameworks, and tools for testing P2P systems should consider creating scenarios that can replicate real-world conditions such as failures and churn.

6 RQ3 - Which applications of P2P systems are most covered in the literature?

Figure 3 shows the number of articles published per year related to P2P system testing. It is visible that the year 2023 had a significant increase in the number of published works, indicating a current academic interest in this topic. Most of the articles published in 2023 are related to Blockchain applications. This type of application is recent and has been developing in recent years.

Figure 3. Number of articles published per year



To evaluate academic interest in P2P systems, the applications addressed in the articles were mapped. Figure 4 shows that the vast majority, about 70%, address the topic of blockchain. 20% of the studies are about Byzantine Fault Tolerant (BFT) and Distributed Hash Table (DHT).

DHT is a distributed system that provides a lookup service similar to a hash table. Key-value pairs are stored in the DHT, and any participating node can efficiently retrieve the value associated with a given key. DHTs are an essential component of many peer-to-peer (P2P) networks and efficiently locate data without relying on a centralized directory (Sit and Morris, 2002). BFT is a class of permissionless systems that can identify and reject faulty or dishonest peers, and there are many consensus algorithms to identify the malicious behavior of peers and secure the consensus of the peers (Zhong et al., 2023). Both DHT and BFT lay the foundation for P2P systems by enabling distributed data storage and fault-tolerant consensus, and are even widely used by blockchain applications.

The other studies (12%) approach Internet of System (IOT)/Machine-To-Machine(M2M) applications (Shala et al. (2017)), P2P mobile applications (Meftah et al. (2017)), and peer-sampling services (Machado et al. (2020)).

7 Open issues and future research directions

Although many studies explore software testing for P2P systems, we have identified that most of them conducted experiments on only one P2P system or on just one application of P2P systems. This means that an experimental study in this field might be valuable and could continue this work. Moreover, the following sub-sections will discuss possible future research.

7.1 Differential testing

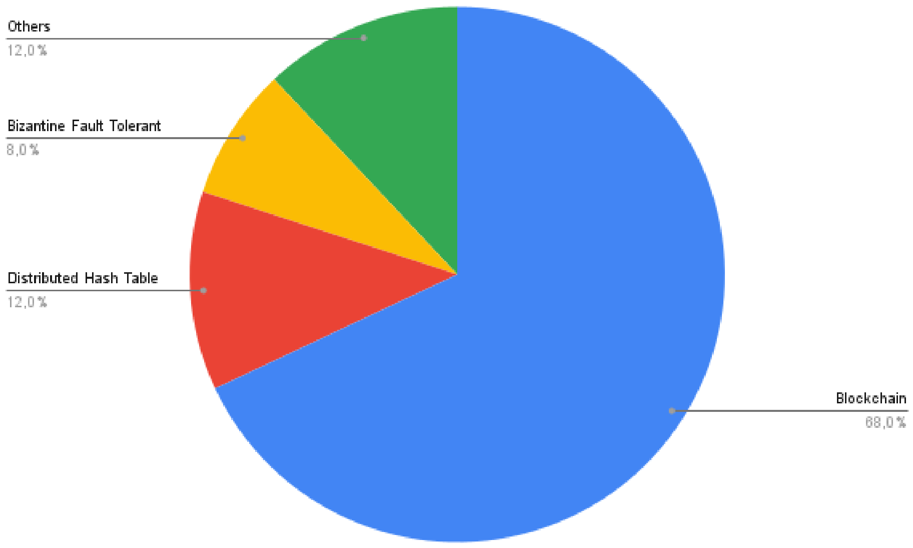
In a P2P network, nodes may be running different systems that implement the same protocol. In this context, they should have the same behavior, and some studies have explored differential testing between these systems. Specifically, differential fuzzing has been used to find discrepancies between them.

However, a P2P network be majority composed by the peers running the same implementation is common, but we cannot ensure they are running on the same operational system and hardware setups, compiled with same compilers, etc. For this reason, exploring differential and even integration testing considering these aspects might be promising. A recent study introduced a compiler-driven differential testing tool to find unstable code in C/C++ programs (Li and Su, 2023). They found more than 70 bugs in different projects which some of them cannot be detected by sanitizers.

7.1.1 Analysis of CI/CD setups

As highlighted before, a P2P network can be composed of peers running the same software but compiled with different compilers, running on different operational systems, hard-

Figure 4. Comparison of the applications covered in the articles



ware setups, and others. It means that projects of P2P systems should test their systems considering these different scenarios. It would be valuable to analyze the CI/CD setups of open-source P2P systems projects to know:

- Are the tests run in different operational systems?
- Is the system compiled with different compilers and sanitizers?
- Is there any integration testing with previous releases?
- Is mutation testing adopted? Is there a test coverage report?
- In case there are other implementations, is differential testing adopted?

7.2 Mutation testing

Mutation testing has not been explored much, but it might be used to evaluate testing proposals. One of the studies shows that some P2P projects should not only rely on the code coverage metric but also take into consideration the mutation score Jain et al. (2023).

Since nodes do not always run the same system or version in a P2P network and a specific application can be used as a reference for other ones (Groce et al., 2022), we identified that mutation testing could be explored with differential testing, especially to evaluate the test suite of an application comparing with another application of the same protocol. Since they are implementations of the same protocol, test cases from one project can be used by others. Mistakes made by developers during the development of an existing implementation can inspire the creation of new mutation operators that could be used by other implementations.

Also, differential fuzzing is promising to identify equivalent mutants (Garcia et al., 2024). In fuzzing, *seed corpus* is a collection of input data files that serve as starting points for generating test cases, and the study by Garcia et al. (2024) shows that doing differential fuzzing with a good seed corpus can increase the efficiency of identifying equivalent mutants. That is, a new implementation could use the available seed corpus from another project since they are compatible.

7.3 Concurrency testing

It is important to highlight that only one paper proposes a specific testing approach for concurrency. There is an open gap about concurrent events, especially for consensus algorithms, that should be taken in consideration on the testing activity.

7.4 P2P networks operating across multiple types of networks

P2P networks can operate across multiple layers, such as clearnet, Tor, CJDNS, and I2P, providing flexibility and resilience by enabling seamless connectivity in varied environments. This multi-layer capability allows users to navigate network restrictions, enhance privacy, and ensure secure communication. The ability to interact over different layers also supports network redundancy, ensuring that disruptions on one layer do not isolate the entire network. Testing these multi-layered systems is essential to maintain their security, stability, and efficiency. Each layer introduces unique considerations, such as latency, routing complexity, and differing levels of anonymity, which require thorough evaluation. Testing ensures that the P2P network can handle data flow smoothly and securely across layers and optimize performance for various use cases.

For example, Sakib et al. (2024) explore the networking performances of anonymous routing with a focus on their impacts on the Bitcoin consensus protocol. However, we did not find any study approaching this context in software testing for P2P systems.

7.5 LLMs and Software Testing

Some studies propose to use Large Language Models (LLMs) to generate test cases (Schäfer et al., 2024) (Plein et al., 2024) (Alshahwan et al., 2024) (Etemadi et al., 2024). Schäfer et al. (2024) highlights that LLM-based test generation already outperforms state-of-the-art previous test generation methods. Alshahwan et al. (2024) explores the usage of LLMs to automatically improve existing human-written tests. In their experiments, 73% of its recommendations were accepted for production deployment by software engineers. Although no study from this review has addressed this in the context of P2P systems, we mapped a promising use.

In Section 4, we presented that one of the nuances of P2P networks is that a P2P protocol can have multiple implementations. That is, a P2P network can be composed of nodes running different systems or different versions of the same system. In this context, LLMs can be explored to generate test cases based on the specification of the protocol and could be used by any implementation.

8 Threats to validity

To ensure the robustness of our findings, we seek to maintain a comprehensive approach to minimize the following threats:

- *Internal validity*: It refers to the extent to which the study accurately measures what it intends to measure within the context of the study (Zhou et al., 2016).
  The choice of the inclusion and exclusion criteria could have affected this study. Certain relevant studies might be overlooked, or irrelevant studies might be included, which could skew the findings. To mitigate this, we first developed clear inclusion and exclusion criteria and we tested these criteria in a sample of articles that we previously mapped as relevant for this study as well as for articles that seems to be interesting for this study but in fact they are not.
- *External validity*: It refers to the extent to which the findings of the study can be generalized beyond the specific context in which the study was conducted (Zhou et al., 2016).
  The choice of terminologies and keywords to construct our search query might have excluded relevant papers. This limitation could have resulted in an incomplete dataset from the existing literature. To mitigate this, we applied the snowballing technique, which involves reviewing the references of the initially identified papers to uncover additional relevant studies. Also, we selected various relevant databases for the search step, we did not limited our search to a specific application of P2P system and we did not limit study designs during the selection.
- *Construct Validity*: It involves the degree to which the study accurately measures the theoretical constructs or concepts it aims to measure (Zhou et al., 2016).
  We mitigated this risk by clearly defining our methodology to construct this study which is aligned to the literature. We presented our research questions, criteria, control group, search string, etc.
- *Conclusion Validity*: It demonstrates that the study can be replicated with the same results (Zhou et al., 2016).
  During the paper selection and reading process, there is a risk of misinterpretation of the content and findings of the studies. This threat is inherent in systematic reviews due to the subjective nature of interpreting research work, and we mitigated this by double-checking the studies and involving more than one researcher in this process. In addition to that, we took care of the choice of keywords to create the search string to include all types of P2P applications. We included generalist words like "P2P" and "peer-to-peer" as well as explicitly put specific words like "blockchain", "DHT", "distributed ledger", and others.

9 Conclusion and Future work

P2P systems have emerged as a fundamental architecture for distributed computing, enabling decentralized communication and resource sharing among interconnected systems (Fox, 2001). This architecture is the heart of critical protocols, such as blockchain applications. This paper presents a systematic mapping of P2P system testing. We seek to map and discuss the nuances and challenges of testing P2P systems, as well as the state-of-the-art.

The data and discussions provided show that testing P2P systems presents a series of unique challenges due to their inherent characteristics, such as: **Validation of global properties and consensus**, **volatility**, **large volume of data**, **malformed data**, **reproducibility and non-determinism**, **network/environment errors or limitations**, **multiple implementations**, **concurrency and parallelism**. Consequently, various testing techniques and approaches have been explored in this context. We can highlight:

- **Model-based testing**: Facilitates the creation of test cases by abstracting a variety of characteristics that a P2P system may have and facilitating the testing of global properties.
- **Conformance testing**: Most used to test the conformity of a P2P system about the characteristics and limitations of the environment and the physical network.
- **Fuzzing**: The technique most used to test the communication of nodes on the network and the impact of malformed and unexpected messages in P2P systems. Most of the papers that have explored this technique have found bugs and vulnerabilities in the systems tested.
- **Stress testing**: Important for testing system behavior in high-volume scenarios.
- **Concurrency testing**: This must be considered as P2P systems are susceptible to concurrency situations. Concurrent requests happen frequently in a P2P network.

It is essential to note that emerging trends in P2P systems, such as decentralized finance (DeFi) or distributed machine learning, may lead to new challenges in relation to software testing activity. However, our SM did not find primary studies related to software testing for these emerging trends, providing an opportunity for future research directions.

As future work, we intend to investigate more deeply how testing is handled in open-source projects of P2P systems. We intend to analyze the techniques and tools used in practice as well as how testing is handled in the CI/CD environment.

Also, as with any review, it is unlikely that we found and explored all primary studies which we aim to update this study in the future.

We hope this mapping contributes to future research on the subject and improves the reliability of P2P systems. The importance of this review for the research community lies in its ability to consolidate and synthesize knowledge on P2P system testing, a critical area given the growing reliance on decentralized networks. By mapping existing studies and identifying gaps, this review not only provides a comprehensive understanding of the current scenario but also serves as a guide for future research. Researchers can leverage this synthesis to prioritize areas with the greatest potential for impact on the challenges of testing P2P systems. Additionally, this review presents best practices and effective testing strategies, which are valuable both for researchers and practitioners. The insights offered here can drive innovation, facilitate collaboration, and ultimately support the development of more secure and resilient P2P systems.

10 Acknowledgement

Simone R. S. Souza is supported by a research grant from the Brazilian funding agency CNPq with reference number: 308445/2021-0.

References

Adamsky, F., Kaiser, D., Steglich, M., and Engel, T. (2021). Locust: Highly concurrent dht experimentation framework for security evaluations. In *Proceedings of the 2020 10th International Conference on Communication and Network Security*, ICCNS '20, page 115–122, New York, NY, USA. Association for Computing Machinery.

Ahmad, A. A.-S., Brereton, P., and Andras, P. (2017). A systematic mapping study of empirical studies on software cloud testing methods. In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 555–562.

Alshahwan, N., Chheda, J., Finogenova, A., Gokkaya, B., Harman, M., Harper, I., Marginean, A., Sengupta, S., and Wang, E. (2024). Automated unit test improvement using large language models at meta. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, FSE 2024, page 185–196, New York, NY, USA. Association for Computing Machinery.

Amaral, M., Pardal, M. L., Mercier, H., and Matos, M. (2020). Faultsee: Reproducible fault injection in distributed systems. In *2020 16th European Dependable Computing Conference (EDCC)*, pages 25–32.

Arsat, N., Bakar, N. S. A. A., and Yahya, N. (2022). Testing in blockchain-based systems: A systematic review. In *2022 10th International Conference on Cyber and IT Service Management (CITSM)*, pages 1–6.

Aziz, A., Riaz, M. T., Jahan, M. S., and Ayub, K. (2020). Meta-model for stress testing on blockchain nodes. In *2020 3rd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pages 1–4.

Boumaiza, A. and Sanfilippo, A. (2024a). A testing framework for blockchain-based energy trade microgrids applications. *IEEE Access*, 12:27465–27483.

Boumaiza, A. and Sanfilippo, A. (2024b). A testing framework for blockchain-based energy trade microgrids applications. *IEEE Access*, 12:27465–27483.

Chen, Y. (2024). Chronos: Finding timeout bugs in practical distributed systems by deep-priority fuzzing with transient delay. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 108–108, Los Alamitos, CA, USA. IEEE Computer Society.

Chen, Y., Ma, F., Zhou, Y., Jiang, Y., Chen, T., and Sun, J. (2023). Tyr: Finding consensus failure bugs in blockchain system with behaviour divergent model. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2517–2532.

Chepurnoy, A. and Rathee, M. (2018). Checking laws of the blockchain with property-based testing. In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 40–47.

Cvitic, P. H., Dobslaw, F., and de Oliveira Neto, F. G. (2023). Investigating software testing and maintenance of open-source distributed ledger. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 886–896.

Dalal, S. R., Jain, A., Karunanithi, N., Leaton, J. M., Lott, C. M., Patton, G. C., and Horowitz, B. M. (1999). Model-based testing in practice. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, page 285–294, New York, NY, USA. Association for Computing Machinery.

Dinesh, B., Kavya, B., Sivakumar, D., and Ahmed, M. R. (2019). Conforming test of blockchain for 5g enabled iot. In *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 1153–1157.

Eisenbarth, J.-P., Cholez, T., and Perrin, O. (2021a). A comprehensive study of the bitcoin p2p network. In *2021 3rd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS)*, pages 105–112.

Eisenbarth, J.-P., Cholez, T., and Perrin, O. (2021b). An open measurement dataset on the bitcoin p2p network. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 643–647.

Etemadi, K., Mohammadi, B., Su, Z., and Monperrus, M. (2024). Mokav: Execution-driven differential testing with llms.

Fox, G. (2001). Peer-to-peer networks. *Computing in Science Engineering*, 3(3):75–77.

Garcia, B. E. R., Delamaro, M. E., and Souza, S. d. R. S. d. (2024). Towards differential fuzzing to reduce manual efforts to identify equivalent mutants: A preliminary study. *Anais*.

Groce, A., Jain, K., van Tonder, R., Kalburgi, G. T., and Goues, C. L. (2022). Looking for lacunae in bitcoin core's fuzzing efforts. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineer-*

ing in Practice, ICSE-SEIP '22, page 185–186, New York, NY, USA. Association for Computing Machinery.

Gulzar, M. A., Zhu, Y., and Han, X. (2019). Perception and practices of differential testing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 71–80.

Hassanzadeh-Nazarabadi, Y., Rybalov, M., and Claybon, K. (2023). Bft testing framework for flow blockchain. In Machado, J. M., Prieto, J., Vieira, P., Peixoto, H., Abelha, A., Arroyo, D., and Vigneri, L., editors, *Blockchain and Applications, 5th International Congress*, pages 338–347, Cham. Springer Nature Switzerland.

Hsueh, M.-C., Tsai, T., and Iyer, R. (1997). Fault injection techniques and tools. *Computer*, 30(4):75–82.

Imtiaz, M. A., Starobinski, D., Trachtenberg, A., and Younis, N. (2019). Churn in the bitcoin network: Characterization and impact. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 431–439.

Jabbar, R., Krichen, M., Shinoy, M., Kharbeche, M., Fetais, N., and Barkaoui, K. (2020). A model-based and resource-aware testing framework for parking system payment using blockchain. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pages 1252–1259.

Jain, K., Kalburgi, G., Goues, C. L., and Groce, A. (2023). Mind the gap: The difference between coverage and mutation score can guide testing efforts. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pages 102–113, Los Alamitos, CA, USA. IEEE Computer Society.

Kim, S. and Hwang, S. (2023). Etherdiffer: Differential testing on rpc services of ethereum nodes. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, page 1333–1344, New York, NY, USA. Association for Computing Machinery.

Li, R., Zhu, L., Li, C., Wu, F., and Xu, D. (2023). Bns: A detection system to find nodes in the bitcoin network. *Mathematics*, 11(24).

Li, S. and Su, Z. (2023). Finding unstable code via compiler-driven differential testing. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS 2023, page 238–251, New York, NY, USA. Association for Computing Machinery.

Li, W., Feng, C., Zhang, L., Xu, H., Cao, B., and Imran, M. A. (2021). A scalable multi-layer pbft consensus for blockchain. *IEEE Transactions on Parallel and Distributed Systems*, 32(5):1146–1160.

Long, N. and Thomas, R. (2001). Trends in denial of service attack technology. *CERT Coordination Center*, 648(651):569.

Ma, F., Chen, Y., Ren, M., Zhou, Y., Jiang, Y., Chen, T., Li, H., and Sun, J. (2023a). Loki: State-aware fuzzing framework for the implementation of blockchain consensus protocols. *Proceedings 2023 Network and Distributed System Security Symposium*.

Ma, F., Chen, Y., Zhou, Y., Sun, J., Su, Z., Jiang, Y., Sun, J., and Li, H. (2023b). Phoenix: Detect and locate resilience issues in blockchain via context-sensitive chaos. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 1182–1196, New York, NY, USA. Association for Computing Machinery.

Machado, N., Maia, F., Neves, F., Coelho, F., and Pereira, J. (2020). Minha: Large-Scale Distributed Systems Testing Made Practical. In Felber, P., Friedman, R., Gilbert, S., and Miller, A., editors, *23rd International Conference on Principles of Distributed Systems (OPODIS 2019)*, volume 153 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:17, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Meftah, L., Gomez, M., Rouvoy, R., and Chrisment, I. (2017). Androfleet: Testing wifi peer-to-peer mobile apps in the large. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 961–966.

Milka, G. and Rzadca, K. (2018). Dfuntest: A testing framework for distributed applications. In Wyrzykowski, R., Dongarra, J., Deelman, E., and Karczewski, K., editors, *Parallel Processing and Applied Mathematics*, pages 395–405, Cham. Springer International Publishing.

Miller, B. P., Fredriksen, L., and So, B. (1990). An empirical study of the reliability of unix utilities. *Communications of the ACM*, 33(12):32–44.

Motlagh, S. G., Mišić, J., and Mišić, V. B. (2020). Impact of node churn in the bitcoin network with compact blocks. In *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pages 1–6.

Pankov, K. N. (2020). Testing, verification and validation of distributed ledger systems. In *2020 Systems of Signals Generating and Processing in the Field of on Board Communications*, pages 1–9.

Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M. (2008). Systematic mapping studies in software engineering. *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, 17.

Plein, L., Ouédraogo, W. C., Klein, J., and Bissyandé, T. F. (2024). Automatic generation of test cases based on bug reports: a feasibility study with large language models. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, ICSE-Companion '24, page 360–361, New York, NY, USA. Association for Computing Machinery.

Reddivari, S., Orr, J., and Reddy, R. (2023). Blockchain-oriented software testing: A preliminary literature review. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 974–975.

Rozsíval, M. and Smrčka, A. (2023). Netloiter: A tool for automated testing of network applications using fault-injection. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 207–210.

Saad, M., Chen, S., and Mohaisen, D. (2021). Root cause analyses for the deteriorating bitcoin network synchronization. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 239–249.

Sakib, N., Wuthier, S., Zhang, K., Zhou, X., and Chang, S.-Y. (2024). From slow propagation to partition: Analyzing bitcoin over anonymous routing. In *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 377–385.

Schollmeier, R. (2001). A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proceedings First International Conference on Peer-to-Peer Computing*, pages 101–102.

Schäfer, M., Nadi, S., Eghbali, A., and Tip, F. (2024). An empirical evaluation of using large language models for automated unit test generation. *IEEE Transactions on Software Engineering*, 50(1):85–105.

Shala, B., Wacht, P., Trick, U., Lehmann, A., Shala, B., Ghita, B., and Shiaeles, S. (2017). Framework for automated functional testing of p2p-based m2m applications. In *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 916–921.

Sit, E. and Morris, R. (2002). Security considerations for peer-to-peer distributed hash tables. In *International Workshop on Peer-to-Peer Systems*, pages 261–269. Springer.

Sommerville, I. (2011). *Software Engineering, 9/E*. Pearson Education India.

Steinmetz, R. and Wehrle, K. (2005a). *Peer-to-peer systems and applications*, volume 3485. Springer.

Steinmetz, R. and Wehrle, K. (2005b). *Peer-to-Peer Systems and Applications*. Information Systems and Applications, incl. Internet/Web, and HCI. Springer.

Sunyé, G., de Almeida, E. C., Le Traon, Y., Baudry, B., and Jézéquel, J.-M. (2014). Model-based testing of global properties on large-scale distributed systems. *Information and Software Technology*, 56(7):749–762.

van Meerten, M., Ozkan, B. K., and Panichella, A. (2023). Evolutionary approach for concurrency testing of ripple blockchain consensus algorithm. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 36–47.

Veldkamp, L., Olsthoorn, M., and Panichella, A. (2023). Grammar-based evolutionary fuzzing for json-rpc apis. In *2023 IEEE/ACM International Workshop on Search-Based and Fuzz Testing (SBFT)*, pages 33–36.

Wang, J., Zhang, B., Wang, K., Wang, Y., and Han, W. (2024). Bftdiagnosis: An automated security testing framework with malicious behavior injection for bft protocols. *Computer Networks*, 249:110404.

Winter, L. N., Buse, F., de Graaf, D., von Gleissenthall, K., and Kulahcioglu Ozkan, B. (2023). Randomized testing of byzantine fault tolerant algorithms. *Proc. ACM Program. Lang.*, 7(OOPSLA1).

Yang, Y., Kim, T., and Chun, B.-G. (2021). Finding consensus bugs in ethereum via multi-transaction differential fuzzing. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 349–365. USENIX Association.

Yutia, S. N. and Fathiyana, R. Z. (2022). Performance analysis of blockchain in tendering process based on hyperledger framework. In *2022 10th International Conference on Information and Communication Technology (ICoICT)*, pages 190–193.

Zhong, W., Yang, C., Liang, W., Cai, J., Chen, L., Liao, J., and Xiong, N. (2023). Byzantine fault-tolerant consensus algorithms: A survey. *Electronics*, 12(18).

Zhou, X., Jin, Y., Zhang, H., Li, S., and Huang, X. (2016). A map of threats to validity of systematic literature reviews in software engineering. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, pages 153–160.