

# bradscholars

## Functional and performance analysis of discrete event network simulation tools

Item Type	Article
Authors	Musa, Ahmad S.;Awan, Irfan U.
Citation	Musa A and Awan I (2022) Functional and performance analysis of discrete event network simulation tools. Simulation Modelling Practice and Theory. 116: 102470.
DOI	<a href="https://doi.org/10.1016/j.simpat.2021.102470">https://doi.org/10.1016/j.simpat.2021.102470</a>
Publisher	Elsevier
Rights	© 2021 Elsevier B.V. All rights reserved. Reproduced in accordance with the publisher's self-archiving policy. This manuscript version is made available under the CC-BY-NC-ND 4.0 license.
Download date	2026-03-08 23:10:11
Link to Item	<a href="http://hdl.handle.net/10454/18882">http://hdl.handle.net/10454/18882</a>

# Functional and Performance Analysis of Discrete Event Network Simulation Tools

Ahmad Musa and Irfan Awan

Department of Computer Science, Faculty of Engineering and Informatics

University of Bradford, United Kingdom

## Abstract:

Researchers have used the simulation technique to develop new networks and test, modify, and optimize existing ones. The scientific community has developed a wide range of network simulators to fulfil these objectives and facilitate this creative process. However, selecting a suitable simulator appropriate for a given purpose requires a comprehensive study of network simulators. The current literature on network simulators has limitations. Limited simulators have been included in the studies with functional and performance criteria appropriate for comparison not been considered, and a reasonable selection model for selecting the suitable simulator has not been presented. To overcome these limitations, we studied twenty-three existing network simulators with classifications, additional comparison parameters, system limitations, and comparisons using several criteria.

Discrete event, simulation, network simulators, structured, unstructured

## 1. Introduction:

It is an expectation of the scientific communities to perform research, evaluate, and present research developments. The results must be reproducible for other peers to verify any findings. Simulations, experiments, and mathematical models are some of the standard methods used to produce verification. Simulators must imitate the heterogeneous nature of these networks in a controllable environment fitting to answer *what-if questions* since most of these networks and applications were intended to be used on the Internet. Likewise, research tools must support the configuration of large-scale experiments to mimic how a system would work during the interaction with a vast number of nodes.

Network simulation is undeniably one of the most leading evaluation methodologies in the field of computer networks. It is mainstream for the development of new network protocols and communication architectures [1]. Network simulators support the modelling of a random computer network by defining both the communication channels and the behaviour of the network nodes. For instance, a network simulator is usually used to investigate the characteristics of a new routing protocol. Eventually, the routing behaviour can be studied in various topologies, given that the network topology is only a set of simulation parameters. Most of the network simulation toolkits available are based on the Discrete Event-based Simulation (DES) paradigm [2]. In the DES, the simulated network nodes trigger events. For example, when a packet is sent to another node. The simulator manages an event queue classified by the scheduled event execution time. The simulation is conducted by sequentially processing the events in the queue.

The first published approaches where DES was used to simulate computer networks were about two decades ago [3,4]. Until the 1950s, computer simulation did not entice many people because it took overly long to generate credible results and required many skills and resources. IBM used a discrete-event computer simulation for the telephone system but unfortunately took

extremely long [5]. Network simulation has been an essential resource for functional and performance analysis of network protocols. The number of widely used network simulators is currently significant, and new tools and systems are being developed daily to overcome various problems and dysfunctionalities.

While numerous simulators exist for modelling various networks, we examined the twenty-three leading network simulators underlining their architecture, usability, scalability, result from statistics, portability, and system limitations. These simulators were chosen based on their usage, popularity, features, published results, and compelling characteristics.

The majority of network simulators are studied and surveyed in [6–14]. However, there is room for improvement in the existing surveys of network simulators as follows:

- New simulators are being proposed regularly with the growing demand and requirements of computer networking. The existing survey [6–14] may lack in the discussion of the recent functional simulators such as NS2[15], NS3[15], OPNET [16], OMNET++[17] and NetSquid [18]. Besides, with the vast amount of available simulators now, a credible survey requires to compare at least ten network simulators to be comprehensive.
- With the development of new network simulators, compendious features or criteria are also required to compare the functionality and performance of network simulators effectively.
- In addition, a comparison of previous and recent simulators with highly desirable features of high scalability, a compilation of statistics on simulation results, and the availability of GUI for visualization is needed.

These observations motivated us to foreground the survey of network simulators with a wide range of simulators, new evaluation criteria, and documentation. Thus, our contributions are as follows:

- We present a comprehensive survey of twenty-two network simulators, including the most recent network simulators such as OPNET[16], OMNET++[17], NS3[15], and NetSquid[18].
- The main contribution of this paper is the categorization of the architecture and performance of network simulators. This categorization is valuable to researchers and scientific communities for identifying the best network simulator for their case studies. We also categorized the simulators based on their functionality viz: generic or domain specific simulators.
- In addition, we classified each of the twenty-three reviewed simulators based on several evaluation criteria, such as their ability for statistics gathering, portability, and system limitations. We verified and presented all the simulators' accuracy and efficiency by putting each simulator under the rigorous microscopic view of research, downloading their source codes, and running the simulations repeatedly.

We discuss in detail the characteristics of network simulators in section 2, followed by the evaluation methodology we used in analyzing the simulators in section 3. Various network simulators are presented and classified according to their characteristics in section 4. Next, we present a discussion and result comparison in section 5 that visualizes the analysis in section 4. Lastly, we discuss the conclusion of the network simulators and the future direction of such a study.

## 2. FEATURES/COMPONENTS OF PEER-TO-PEER SIMULATORS

The simulators can be evaluated based on numerous criteria that follow, highlighting the strengths and weaknesses of the applications according to their properties[6–10,13,14].

### 2.1 Simulator Architectural Overview

Simulator architecture is a fundamental component that enables the formation of intricate simulations models. Simulators have different focal points and are designed with varying functions for research domains; hence they differ in architecture. It is, therefore, necessary to compare the simulator architectures as the main characteristic when evaluating different simulation frameworks. Simulator architecture can be categorized based on its protocol, simulation engine, underlying network simulation, underlying protocol simulation, parallel simulation, and churn support.

This criterion suggests; whether the simulator supports structured or unstructured overlays or both, whether the simulator uses a scheduler that synchronizes messages shared among nodes, or the simulation loops through each node adding delay as necessary, using either a discrete event simulation model or query-cycle. It also notes how identifiers are determined, how remote procedures are implemented, how node behaviour is simulated and whether distributed simulation is allowed, measuring if multiple simulations can be run to enable significant scaling?

### 2.2 Usability/Documentation

Documentation is a critical feature of assessing how user-friendly a simulator is, i.e., how easy the simulator is to use and learn. This means the simulator source code, manuals, user guides should be well documented, extendable, and easily understood. The availability of online supports in a website or mailing list to promote user experience is also a key component. Finally, usability assesses how experiments are set-up; the availability of script language, its documentation, and ease of use.

### 2.3 Scalability

The scalability of a simulator is regularly interpreted as the feasible network size as to the number of nodes that can be simulated. It is an essential and challenging feature for verifying the performance of a simulator because some applications, e.g., P2P protocols, have a relatively large network size. Therefore, one of the most crucial evaluations a simulator can offer is how the protocol scales to millions of nodes, mainly as this, if not impossible, would be a challenging experiment to try using a real network.

### 2.4 Statistics

Another key parameter for analysing a simulator is the output it provides. The output needs to be easily manipulated and close to the desired assumptions for statistical analysis and graph productions. The simulator should have a reproducible mechanism that allows result verification from its state of analysis.

### 2.5 Portability

A simulation code is portable if the code can be reusable with slight alterations.

### 2.6 System Limitations

The ability of a simulator to utilize the available computing resources is measured under this parameter. Also, the ability of the simulator to scale will be reduced if the resources are used inefficiently.

### **3. EVALAUTION METHODOLOGY**

We employed a thorough approach as an evaluation methodology for each simulator based on the following available criteria:

- The latest version of the simulator
- The documentation and manuals available
- Source codes
- Whitepapers and research papers
- Running the experiments to compare results and statistics

All the simulators were evaluated based on the above criteria, with no simulator satisfying the evaluation criteria and some showing greater limitations than others. We performed experiments for simulator packages to compare the output with any given output to ensure the simulator's functionality. The lack of support for getting statistics from simulation runs was of most concern. Many authors claimed that they performed experiments to ascertain the scalability of some of the simulators by testing the number of nodes that can communicate and connect successfully. Yet this remains an estimate, not comparable to the numbers realized when simulating a real network system.

### **4. VARIOUS NETWORK SIMULATORS**

We studied several network simulators and classified them according to their relevance and functionality to networks. These are generic simulators that can simulate all kinds of networks and domain-specific simulators designed for specific domains.

#### **4.1 Generic Simulators:**

Generic network simulator offers numerous standard modules of network simulations, so users only need to adjust the core part of the network or application. The simulator can easily be modified or extended with components and applications defined by their clear hierarchy and modular structure. Overlay Weaver, PeerSim, 3LS, P2PSim are a few examples of generic simulators.

##### **4.1.1 Peersim – P2P**

PeerSim is a Java-based P2P simulator made of two simulation engines: a cycle-based (simplified) and event-driven engine [12]. The engines are made with a flexible configuration mechanism, and many extendable, simple, and pluggable elements are supported. To allow for scalability, the cycle-based engine employs some simplifying hypotheses, such as overlooking the section of the transport level in the communication protocol domain. The event-based engine is more realistic but less efficient [13]. In addition, it supports transport layer simulation, among other things. However, cycle-based protocols can be run by the event-based engine too.

Simulating large P2P networks is possible due to its design of been scalable and dynamic. PeerSim can support the simulation of both structured and unstructured overlays. It began with EU projects BISON and DELIS, with more support from the Nepa-Wine projects and others but was released under the GPL open-source license.

Statistics

PeerSim offers class packages to perform statistical computations, user-defined data collection coding, and standard statistics calculations. It also provides some class packages that support notable models, such as BA, random and lattice graphs[11]. However, it gives neither debugging facilities nor any graphical user interface.

### Scalability

PeerSim is often associated with high scalability with a network scale of up to  $10^6$  nodes, but the scalability is at the expense of ignoring the behaviour of the underlying communication network.

### Usability

The PeerSim website offers reusable and straightforward API documentation that can accommodate more elements based on requirements. Most of the API documentation is based on the cycle-based model, while the event-based model's documentation is significantly below par in comparison. However, if the need arises, the cycle-based application can readily be modified to the event-based application.

### Portability

PeerSim admits user-defined entities to substitute nearly all predefined entities in it. It supports pluggable and extendable segment features. A plain ASCII file is used for an adjustable configuration consisting of key-value pairs.

### System Limitations

PeerSim does not apply distributed simulation. Node identifications are generated gradually as integers, but they can also be tailored by user-defined mechanisms.

### Analysis

PeerSim inputs are predefined (set by default) in a text file as a command base simulator. Therefore, the user cannot adjust the information on run-time because there is no GUI, which is tough to use. The generated results are challenging to analyse. The API documentation for the cycle-based model is excellent, but poor C++ API documentation for event-driven. However, it only explains the basics of how to compile and run the simulator in both models. There is no in-depth deliberation concerning the extension of the simulator's source code. PeerSim results are not adequate to perform a reliable statistical analysis.

## 4.1.2 PeerThing

PeerThing is an open-source P2P network simulator that was released in 2006. The simulator is written on Java, with Eclipse serving as the GUI interface.

### Simulator Architecture

The architecture of PeerThing is generally classified into system behaviour and system scenario. System behaviour consists of sets of nodes, states, transitions, actions, and tasks. The system behaviour allows defining the behaviour of each node in the network. The system behaviour takes a peer-cantered position by using a single peer's behaviour to define the behaviour of the whole network [9]. A node can have several states connected with several transitions to perform specific tasks and actions.

In contrast, the system scenario defines the number of node connection's properties such as delays, uplinks and downlink speeds, actions, and loops [19]. Other behaviours can also be added and used by calling the behaviour with the CallBehaviour attribute. The scenario defines the resource allocation for each peer. Once the network has been set, its corresponding code is generated in XML (eXtended Markup Language) for system behaviour and system scenario.

#### Statistics

PeerThing provides a GUI interface for its users to run multiple simulations for a specified number of time steps or messages. The input of the system behaviour and system scenario simulations often generates an output in a specified log file. The generated results can be saved in .csv (comma-separated values) for further analysis.

#### Scalability

The authors claim that PeerThing can simulate up to 2000 nodes for the Gnutella model and 1000 nodes for the Napster model. However, we found that the simulator can only run up to 700 nodes successfully without generating any errors after testing. A stack of java space errors is generated when more than 700 nodes simulate the Gnutella model with its full functionalities.

#### Usability/Documentation

PeerThing has one of the best user manual in comparison to all the other evaluated simulators. There is sufficient information on how to build the network and view its corresponding results. However, the API documentation source code is not well explained with many interconnected things making it challenging to understand.

#### Portability

The generated results are visualized in tables and graphs from the simulator and stored in a log file before another simulation can be run. However, the visualization provided is limited to the simulator, and as such, results are not adequate for analysis on cases not supported by the simulator. But the ability to export files in .csv format provides an avenue for much-detailed analysis of the generated results using other software.

#### Extensibility

Extending the simulator for other protocols or functionalities is an arduous task because the API documentation source code is not well commented. However, some extensibility can be achieved in the behaviour of the given Gnutella and Napster main models.

#### System Limitations

PeerThing simulator is not highly scalable. After roughly 3000 ms of simulation time, and when tested with more than 600 nodes, it begins to present a java heap space error. The simulator does not hold enough memory to run the full simulation successfully.

### **4.1.3 RealPeer**

RealPeer is an open-source development framework written in Java for P2P systems[20]. The framework can be executed as a simulation model or an actual P2P application that connects to remote peers on a real network. Hence, a simulation model of a P2P system can be recycled as part of a real P2P system and vice versa.

#### Simulator Architecture

The RealPeer framework is a generic, highly modular simulator that uses discrete event to develop, model, and simulate structured and unstructured overlays. This allows a developer to combine and freely change the framework and model elements, thus providing a mechanism to reuse components[9]. The framework is as lightweight and scalable as possible for supporting the simulation of large-scale P2P systems.

The framework allows a developer to perform controlled simulation experiments with full internal validity to get reproducible and accurate simulation results. RealPeer utilizes a *message-passing mechanism* to send and receive messages across the physical network using *MessageProtocol*[20]. Additionally, the domain models are improved until they match the intended real P2P system at the end of the development process.

#### Scalability

RealPeer is quite scalable, and its developers have been able to test its scalability with 20,000 peers. However, it does not allow for parallel execution but aims to add functionality in the future.

#### Usability

There is available documentation of RealPeer on the web. The framework's architecture is extensible such that a developer can connect, freely exchange and reuse components of the framework.

#### Portability

Real peer is an object-oriented software framework with a heavy plug-in design pattern, allowing the simulator's extensibility.

#### Statistics

All commands need to be written on the command prompt; as such, no GUI is available for RealPeer. The framework outlines a set of *ObserverEvents* that encapsulate various types of simulation data that are utilized by the registered Observer plug-ins[20]. The plug-ins export the data for external tools for further analysis and reusability in other models.

#### System Limitations

Although RealPeer provides GUI-based predefined simulations for some set scenarios, it only uses a command-line program to run simulations. Simulation results for RealPeer are stored in text files. Nonetheless, the numeric value are stored with no matching variable name. The inputs to the simulator are to be defined in a text file that shows real-time simulation is not supported. However, the aforementioned problems will likely be addressed with time due to RealPeer being actively and extensively developed and still relatively new.

### 4.1.4 Query Cycle

The Query Cycle Simulator (or sometimes called P2PSim) [21] is a peer-to-peer simulator developed by Stanford University for its P2P sociology project. The main focus of the Query Cycle Simulator is to model user behaviour in a P2P file-sharing network accurately[7].

#### Simulator Architecture

The Query-Cycle simulator is a file-sharing simulator that was designed based on java and uses the query cycle model. As the name implies, each cycle in the Query Cycle Simulator is formed on queries generated by the network. Among parameters included for content distribution for

simulations are query activity, download behaviour, and uptime[11]. In each query cycle, a peer may be busy issuing a query or inactive and not responding to queries moving by. After issuing a query, a peer wait for response, selects a download source amongst those nodes that responded, and commences downloading the file. The query cycle only finishes when all peers that submit queries receive a suitable response.

#### Statistics

Query-Cycle simulator uses a graphical user interface from which the user can set parameters for the network attributes, the peer behaviours, and the content distribution. Once the simulation is running and committed, the attributes cannot be modified, but it is possible to pause, resume and save the simulation for subsequent execution. There is also a visualizer that shows the state of the network as cycles are completed in the network. Statistics may be collected for each peer, such as the number of queries sent and received.

#### Scalability

Query Cycle Simulator can simulate more than one million nodes, and its performance is excellent in modelling peer behaviour. However, it experiences limited scalability. There are some cases reported that the simulator does not scale more than 1000 peers when simulating.

#### Usability/Documentation

The Query Cycle developers provided a demo source code and API documentation on their website.

#### Portability

Query-cycle allows almost all predefined entities in it to be replaced by user-defined entities with sustained extendable and pluggable element features. With a GUI present, simulating for sociology studies is easier.

#### System Limitation

Although no specific system limitation is evident, extending the simulator is limited due to poor API documentation and source code.

#### Analysis

Query Cycle Simulator offers a GUI-based interface for users to run simulations. Therefore, usability is not hard in terms of the interoperability of the simulator. The documentation provided contains inadequate API documentation with very little information on the user manual and insufficient information on how to compile the simulator code. The source code appears extendible, but the shortage of suitable documentation negates its extensibility. The simulator does not support generated results for further analysis using another tool and is only limited to the simulator's predefined analysis. One of the significant feats gained by the simulator is the rate at which the number of downloads and uploads happens with each peer.

### **4.1.5 Neurogrid**

Neurogrid is a P2P search protocol project developed to simulate large-scale neural networks. In the early stages of the development, the simulator was initially designed to compare the performance of the Neurogrid, Gnutella, and Freenet protocols[22]. Lately, with the

advancement of computing power, the simulator has been extended to maintain DHT protocols such as Pastry and simulate large-scale neural networks.

#### Simulator Architecture

Neurogrid is a single-threaded discrete event simulator considered for protocol comparison, and it does not simulate the underlying network. The simulator operates on the overlay layer zone and can simulate both structured and unstructured protocols. It was built packaged with the implementations of the Gnutella, Freenet, and Neurogrid protocols[13].

#### Statistics

Neurogrid provides a flexible mechanism for gathering statistics by allowing comprehensive data to be extracted. Simulation scenarios are specified in a file of simple parameters. Also, Neurogrid only simulates the overlay layer simulation by assuming a graph topology as simulator input.

#### Scalability

The Neurogrid simulator design assumes all links between nodes have equal bandwidths, and no bandwidth data is connected with the list of joined nodes stored at each node. The authors [23] claim that up to 300,000 nodes have been simulated on a machine having no more than 4GB RAM. However, we failed at replication due to thread limits.

#### Usability/Documentation

Neurogrid has abundant documentation available on the internet but is disorganized in the form of wiki documentation.

#### Portability

Neurogrid simulator was designed for protocol comparison extensibility. The Simulator uses many abstract classes that are meant to be generic across various P2P implementations. Moreover, it does not support churn simulation, and identifiers are generated incrementally.

#### System Limitations

The simple file parameter does not seem to have the ability to schedule events at specified times. Although it may be easy to modify the simulator to implement other behaviours, node failure is not included with the current implementation.

#### Analysis

Neurogrid appears to be a reliable way to compare the performance of different P2P simulators. The availability of extensive documentation allowed the network to be extended to support the simulation of large-scale neural networks.

### **4.1.6 GPS - General Peer-to-Peer Simulator**

GPS is an event-driven P2P simulator that prioritizes modelling P2P protocols as accurately, realistic, efficient, and dynamic as possible. Written in Java, the simulator maintains its performance by modelling communication at the message level [24]. It supports the simulation of both structured and unstructured overlays.

#### Simulator Architecture

GPS does not have fixed synchronous increments. Rather, the processing and time advancement are triggered by events. The GT-ITM model of considering a Transit-Stub topology is used to model the underlying network topology as a message simulator partially [24]. It provides many different flow level models but does not model each packet. Packet level simulation is not implemented in order to preserve its performance.

#### Statistics

GPS excels in extensibility for modelling any P2P protocol, integrating GUI and network visualization, and providing topology generation tools. It includes simulation models for BitTorrent, which has not been modelled functionally before to the best of our knowledge.

#### Scalability

GPS has been tested with up to 512 BitTorrent peers, 1 BT tracker, and a 1,054 nodes graph. Three connections are considered within the underlying Transit-Stub topology: between transit nodes, between transit nodes and the stub, and between transit nodes within the stub[24]. The connection group can fix bandwidth and delay. It can also be set from a bandwidth and delay matrix per individual connection. GPS cannot simulate cross traffic.

#### Usability/Documentation

GPS is poorly documented, with no in-depth discussion on how to run the simulator. Additionally, using the simulator has a steep learning curve, and it is not easy to use.

#### Portability

GPS uses Java as the development engine and simulation language for portability, extensibility, and ease of development. It also models the network topology and characteristics to maintain accurate simulation. The GPS framework includes all the infrastructures needed for P2P simulation[11]. Hence new protocols can easily be run and plugged in on existing protocols.

#### System Limitations

The simulator was designed to simulate and be conjoined with the BitTorrent protocol. The simulator design has made it strenuous for researchers to implement protocols other than the BitTorrent protocol.

#### Analysis

GPS has a GUI environment for users' interaction and allows users to select macro models from a list of models. The extensibility of the code is not accessible because the API documentation of the GPS doesn't contain sufficient information, and it is hard to use. The simulations lack adequate information to determine the relationship between the inputs and the generated outputs with no user manual. Hence, it is challenging to analyze the simulation results. Its support for only the BitTorrent protocol and not any other protocol is a significant impediment.

### **4.1.7 P2PSim**

P2PSim has a distinct goal from other simulators. Its design focuses on three main goals: to simplify P2P protocol source codes; to ease comparison between protocols; and to have reasonable performance[9]. P2PSim is one of the few P2P simulators that utilise threads in simulation. This makes protocol implementations comparable to their pseudo-codes.

## Simulator Architecture

P2PSim is a discrete event simulator that can only simulate structured overlay network topologies. Node IDs are generated by a regular 160-bit SHA-1 hashing and can simulate node failures with support for both iterative and recursive lookups[11]. However, distributed simulation, cross-traffic, and huge fluctuations of bandwidths are not supported.

### Statistics

Coding is required before any set of statistics can be collected. The P2PSim has made many underlying network topologies available such as random graph; G2 graph; end-to-end time graph; GT-ITM, and Euclidean graph, which is the mostly used.

### Scalability

P2PSim has been tested to scale up to 3000 nodes for Euclidean constant failure model topology[1]. Furthermore, an experiment has been performed with the King data set to simulate a 1700-node Internet topology.

### Usability/Documentation

P2PSim has poor C++ API documentation

### Portability

The protocols in P2PSim can be extended. However, limited and other protocols such as the custom event generators can be developed and implemented by extending certain base classes.

### System Limitations

The main drawback of P2PSim is the lack of support for unstructured or semi-structured P2P routing protocols simulation.

## 4.1.8 3LS or 3 Layered Architecture

3LS is an open-source simulator written in Java for overlay networks and designed to solve the issues of extensibility and usability. It has a friendly user interface for the development of a new P2P protocol. In terms of the simulation granularity, it still pulls messages as the main simulation object[9].

### Simulator Architecture

3LS is a novel P2P simulator with a clock-based simulation engine that supports only unstructured overlays. The name 3LS was coined by dividing the simulator into 3 architectural layers; the network layer at the bottom, the protocol layer in the middle, and the user level at the top[7]. Communication in the 3-layered system can only occur with adjacent layers. The simulator uses four queues at each node to store pending message objects, viz; Outbox, Inbox-for-network-delay, Inbox-for-processor-delay, and Inbox. These queues are intended to enable the simulation of delays associated with network traffic and CPU delays.

### Statistics

3LS is integrated with GUI and uses main memory to store each event executed for visualization (R). Results are stored in a file when simulations are completed.

### Scalability

3LS is limited with scalability because of a high memory overhead incurred by the Network Layer. As a result, it can only simulate networks with network sizes of a couple of thousand peers on a regular machine (R).

#### Usability/Documentation

The 3LS API is well documented, but the simulator is not present on the web. The authors can share the source code with researchers upon request. However, it is not well-documented for P2P simulation.

#### Portability

3LS is an open-source simulator for overlay networks designed to solve the issues of extensibility and usability.

#### System Limitations

The simulator uses main memory to store simulation results, limiting the system performance in terms of nodes. It can also not properly simulate the transfer layer's detail, the routing, and the sources of the files P2P networks.

### **4.1.9 Overlay Weaver**

Overlay Weaver is written in Java as a P2P overlay construction toolkit for the easy development of routing algorithms and testing of P2P protocols. The toolkit includes a standard API for higher-level services such as multicast and Distributed Hash Table (DHT) for application developers[1]. The routing layer is separated from multicasting and DHT services. The simulator supports the rapid implementation by iterative testing of new, improved algorithms from the algorithm developers.

#### Simulator Architecture

The Overlay Weaver simulator supports only the simulation of structured overlays and not the simulation of the underlying network. It is packaged with Kademlia, Chord, Pastry, Koorde, and Tapestry implementations [25]. RPC can either be simulated using discrete-event message passing within the JVM or use real TCP/UDP to test the protocol on a real network. Distributed simulation, although possible, has never been experimented with it because of lack of efficient documentation.

#### Statistics

A message counter is used to record all the communication by the network during the messaging service implementation. The message counter allows the logging of statistics and analysis of the logs gathered in the execution time. An undocumented tool also collects statistics on the number of messages passed, but comprehensive data for statistical analysis requires significant adjustment of the source code.

#### Scalability

The Overlay Weaver documentation says that it can scale up to 4,000 nodes. Its distributed simulation abilities will scale further, overcoming software and hardware system limitations when used on a single machine. Also, a newly implemented algorithm can be tested, compared, and evaluated on the emulator that can host tens of thousands of virtual nodes.

#### Usability/Documentation

Overlay Weaver interface includes the emulator and a small amount of command-line tools. The API is well designed and readable, but the documentation is relatively sparse and does not incorporate many of the simulator's functions. With many undocumented tools, the source code needs to be analysed to understand how to use them.

#### Portability

Overlay Weaver was designed as a tool to support the initial design of P2P protocols making its simulation features secondary. There is a graphical real-time messaging visualiser tool that helps in understanding the operation of the protocol. However, the visualiser burdens the emulator by visualization and by doubling the number of messages, reducing the maximum number of emulated nodes.

#### System Limitations

There are many pitfalls for using the Overlay Weaver simulator, such as hardware and software limitations when scaling and the lack of proper documentation. The aspect of statistics gathering needs and the tools documentation needs a significant redesign.

### **4.1.10 PlanetSim**

PlanetSim is an object-oriented simulation framework written in Java for overlay networks and services. The simulator was initially developed by a research project called Planet and was released later under PlanetSim GPL license V3. It uses software engineering techniques to ease the design and implementation of new protocols and applications [26]. A clean API is available to implement overlay algorithms and application services using a well-structured design.

#### Simulator Architecture

PlanetSim is a discrete-event overlay network simulator that supports structured and unstructured overlays. It also supports both packet-based, Chord-SIGCOMM, and Symphony implementations. The architecture presents the Common API (CAPI) (R) for the decoupled development and analysis of overlay algorithms and that of applications[26]. The CAPI layers 0, 1, and 2 are mapped to the simulator's architecture's; network, overlay, and application layers, respectively. The layers use CAPI to communicate with each other by upcalls and downcalls.

#### Statistics

There are currently no standard mechanisms for collecting statistics. However, it is possible to collect basic statistics, such as, the total simulation time, number of messages used, and in-depth capabilities statistics that can be gathered through aspect-oriented programming (AOP) [11].

#### Scalability

The scalability of PlanetSim has been tested in (R) for Chord and Symphony networks. The tests show that Chord for 8 seconds was used to stabilize 1,000 nodes, 16 minutes to stabilize 10,000 nodes, and 46 hours to stabilize 100,000 nodes. In contrast, Symphony needs 2 seconds to stabilize 1,000 nodes, 98 seconds to stabilize 10,000 nodes, and 1.3 hours to stabilize 100,000 nodes. This shows the overhead imposed by Chord stabilization is huge compared to Symphony's maintenance algorithms.

#### Usability/Documentation

The PlanetSim website [27] provides extensive documentation of the simulator with an independently published research work [26]. It has an excellent and obvious hierarchy API with extension interfaces for entities. An existing entity can also easily be replaced to extend the implementation according to the simulation settings.

#### Portability

Simulation can be saved to disk for reuse and export the network topology graph in Pajek [26] and GML formats. The same simulations code can be reused for experiments with nodes communicating using TCP or UDP.

#### System Limitations

PlanetSim cannot gain critical accolades without external contributors contributing new algorithms and services. The simulator is considered complex and bulky. Although, the rise of simulators that can solve particular problems can be equated to the evasion of clear performance comparisons in a unified platform.

### 4.1.11 NARSES

Narses [28] is a scalable, application-level network simulator that enables researchers to use network models of different levels of accuracy and speed to simulate large distributed applications efficiently. Individual clocks with independent clock skews can be simulated with Narses. Node IDs are denoted as integers, with each new node allocated an incremented value.

#### Simulator Architecture

Four underlying network topology models vary from the least accurate and slowest to the most accurate: Naive, NaiveTop, FairTopo, and SafeFairTopo [28]. The transport layer in Narses includes two socket interface approximations: transport and reliable message transport. A flow-based model presents Narses as a considerably more efficient (at the expense of accuracy) network simulator than packet-level simulators.

#### Statistics

Considering Narses is written in Java, it relies on Java's garbage collector to control memory. Many transient flow objects are linked and left to the garbage collector to reclaim in simulations with large flows.

#### Scalability

Narses' accuracy has been tested to scale up to 10,000 simultaneous flows for a flow size of 200KB with a difference of 7.6% by 1.53seconds. Narses can run distributed simulation jobs and exchange the data using Java RMI.

#### Usability/Documentation

There is extensive documentation available on the sourceforge.net website on Narses. The developers of Narses also published research work documenting the usability of the simulator[28].

#### Portability

Narses' performance is targeted towards large distributed applications. A transport layer interface for which the application can send and receive data is provided by Narses. The

transport layer interface is comparable to a UNIX socket interface, allowing users to easily port their simulated applications to a real operating system.

#### System Limitations

Narses is restricted to Internet-like hierarchical topologies and cannot simulate bottleneck links that are not first-hop. Since it is targeted towards large applications, Narses cannot run simulations with lower-layer protocol dynamics. Different bit error rates simulations for physical channels are also inefficient with Narses.

#### 4.1.12 OMNeT++

OMNeT++[17] is an extensible, modular, component-based simulator and framework for building network simulators. It is written in C++ but supports alternative languages such as Java or C# and many other functions. It also allows wireless and wired communication networks, queuing networks, peer-to-peer networks, cloud computing, on-chip networks, etc., to be modelled. The framework model supports domain-specific functionalities, multiprocessors, and other distributed systems developed as independent projects. Although OMNeT++ was not designed as a network simulator, it has become popular as a network simulation platform in scientific and industrial environments. It continues to build up a large user community.

##### Simulator Architecture

OMneT++ is an open-architecture simulation environment that supports only structured overlays. It also supports discrete event simulation in which modules interact through message passing. Additionally, OMNeT++ promotes parallel distributed simulation execution.

##### Statistics

OMNeT++ has strong GUI support and an embeddable simulation kernel that helps to visualize user interaction. The interactions are logged between modules onto a file. This log file is accessible during or after the simulation run and can be used to map interaction diagrams.

##### Scalability

It can be extended to scale for real-time simulation, network emulation, and database integration. It has been reported to scale up to 3,025 nodes in 1225 seconds in a controlled experiment [15].

##### Usability/Documentation

OMNet++ has a reusable and well-documented API and source code that is robustly backed by many developers and researchers under the Academic Public License. The simulator supports the design of simple modules or components that can be grouped into compound modules and split back to simpler modules when needed. A high programming language NED (NEtwork Description) supports the formation of the compound modules by communicating with the extensible modules through messages with the framework[17].

##### Portability

The main feature of OMNeT++ is the reusability of models. OMNeT++ has an extensive GUI support, and owing to its modular architecture; it can easily be embedded into other applications. It has been successfully used in different fields such as the simulation of IT

systems, hardware architectures, queuing networks, and business processes. OMNeT++ also supports many contributed models and multi-tier topologies. Java interoperability is provided by the JSimpleModule, an extension that supports OMNeT++ modules to be written in Java [11].

### System Limitations

The last ten years have confirmed that the OMNeT++ strategy is viable, and various research groups and individuals have published several OMNeT++-based open-source simulation models and frameworks.

#### 4.1.13 D-P2P-Sim

D-P2P-Sim[25] is a distributed simulation environment that uses a graphical user interface for asynchronous message passing, multi-threading, and distributed environment. It is written in Java to evaluate the performance of different protocols by blending a set of tools in a single software solution. D-P2P-Sim+ enhanced D-P2P-Sim to produce failure-recovery models simulation abilities, multi-million node simulation support, and added statistics[25].

#### Simulator Architecture

D-P2P-Sim is an event-driven, multi-threading simulator that uses the event-driven approach as a pooling technique to generate thousands/millions of nodes, making its simulations more realistic. The architecture of D-P2P-Sim is categorized into four components: the overlay network, the message passing environment, the remote services, and the simulator's services. D-P2P-Sim promotes churn for peer join/leave processes. It contains four critical characteristics[25]:

- (i) Unbiased: meant for the collection of performance data as an independent from the protocol implemented mechanism,
- (ii) Realism: implemented as tight as possible to an application-level P2P software,
- (iii) Distributed: connects many computers on a network and clusters,
- (iv) Pluggable and Extensible: uses an extensible API based on the plugin mechanism of Java.

#### Statistics

The integrated Graphical User Interface incorporates a graphical statistics functionality. The network containing messages is managed by the network monitor and refined by the network filter to extract useful statistical data that the overlay monitor will further process.

#### Scalability

The limited documentation reports that up to 400,000 nodes of different network sizes have been simulated on D-P2P-Sim. Multiple workstations running on the simulator may be connected to produce larger simulated network sizes.

#### Usability/Documentation

D-P2P-Sim uses pluggable and extensible API that is available for additional development. It also includes a dummy implementation with all code and specifications for the basic design of a P2P protocol with its architectural outline, message transaction mechanism, and routing

management. There is limited documentation on this simulator apart from a short paper and a poster.

#### Portability

It uses a GUI button to plot the results and a visualizer to visualize the topology not specified in the literature. The only implemented overlay algorithm is Chord. Since the system is extensible, other algorithms could be implemented.

#### System Limitations

Chord is the only implemented overlay algorithm to be tested. There is limited documentation available to facilitate the extension of the simulator.

### 4.1.14 NS-2

NS-2[8][12][29] is currently one of the most broadly used network simulators in academia and industry. It was designed as a networking research tool that contributes a rich component library on Windows, Linux, UNIX, and other operating system programs. NS-2 supports simulation using a hybrid of C++ and OTCL (object-oriented version of TCL)[10]. The protocols are implemented in C++ while the TCL script specifies the nodes and characteristics of communication links. NS-2 was hardly used in P2P simulation at first but has now been improved to support P2P overlays efficiently. The NS-2 distribution comprises the models together with their supporting infrastructure as one inseparable system.

#### Simulator Architecture

NS2 is a discrete event simulator, originally not designed to support P2P networks but was redesigned later due to its easy configuration and code. It supports structured and unstructured networks and can run in parallel with many other machines.

#### Statistics

For statistics plotting, external tools such as Xgraph or Gnuplot can be used for statistical analysis. An extended version of nam called the environment confirmation tool (iNSpect) and interactive NS-2 protocol was introduced by [29] to provide the animation and visualization of NS-2-based wireless simulations.

#### Scalability

In a controlled experiment [29], NS-2 was observed to have irregular behavior when the number of nodes was 1000. It subsequently failed multiple times to run the simulation with that many nodes. It has been tested to simulate upto 5,000 nodes.

#### Usability/Documentation

It provides substantial support for routing, TCP, and multicast protocol simulation across wireless and wired networks organized in a structured or unstructured manner. There is extensive documentation available for the scripting of NS-2.

#### Portability

NAM (Network AniMator) is used to provide visualization for NS2. The network topology is represented as part of the Tcl script, which normally deals with many other things, from setting

parameters to attaching application behavior and recording statistics. This architecture makes it almost impossible to build graphical editors for NS-2 models.

#### System Limitations

NS2 lacks the graphical presentations of simulation result data. The raw data must be refined using scripting languages such as 'awk' or 'perl' to generate data in an acceptable format for tools like Gnuplot or Xgraph [10]. Another limitation of NS2 is that it is not user-friendly due to its text-based interface, and many researchers complain about NS2's steep learning curve.

#### 4.1.15 NS3

NS-3[30][31] is an open-source simulator, licensed under the GNU GPLv2 license to develop a preferred and open network simulation environment [29]. It is intended to align with modern networking research's simulation needs and encourage community contribution and software validation. Like NS-2, NS-3 uses C++ for the implementation of simulation models. However, ns-3 does not use oTcl scripts to manage the simulation, therefore leaving the difficulties started by combining C++ and oTcl in NS-2. Alternatively, network simulations in NS-3 can be implemented in C++, while the simulation elements can also be completed using Python[30].

#### Simulator Architecture

NS-3 is a discrete event network simulator that is implemented in a modular architecture. Its components can easily be reused in various fields than their original ones. It has recently integrated C++ or Python to enable users to take full advantage of the available support of each language.

#### Statistics

Logging can be requested in NS-3 for statistics and various purposes or levels: error, warning, logic, debugs, and information function. Users can select the level of logging in which packets can be traced. The trace helper group can be requested at the points of interest through the code.

#### Scalability

NS-3 integrates architectural theories and code from GTNetS[31], a simulator with great scalability properties. These design decisions were made at the cost of compatibility that NS-2 models need to be manually ported to ns-3. The NS-3 architecture supports distributed simulation in order to manage the scalability of a massive number of simulated network components.

#### Usability/Documentation

NS-3 endeavors to spread the workload of continuous documentation across a large community of developers and users. Hence, a new stable version of NS-3 is released with newly developed models and documentation to be validated and managed by enthusiastic researchers every three months. The developers [15] encourage third-party users to validate these models through mailing lists to ensure that new models remain of the highest quality possible.

#### Portability

NS-3 supports software-defined networking (SDN), the new paradigm for communication to split the control plane from the data path[8]. This ability provides user flexibility, enabling

them to develop their algorithms to regulate data from different applications operating on the network.

#### System Limitations

NS-3 remains under extensive development, and as such graphic display is still under development. It is a relatively new simulator with a continuous update of modules. Hence, it is not compatible with NS-2.

#### 4.1.16 OverSim

OverSim [15][12] is an open-source simulation framework written in C++ and based on OMNeT++ for Peer-to-Peer and overlay networks. It implements three network models viz. Simple, SingleHost, and INET. In the Simple model, a global routing table sends data packets from one overlay node to another to take care of packet delays depending on node distance in Euclidean space. While the SingleHost model recycles overlay protocol implementations in real networks like PlanetLab without code modifications. The INET underlay model is obtained from the INET framework of OMNeT++ that incorporates simulation models of all network layers from the MAC layer.

#### Simulator Architecture

OverSim is an overlay P2P protocol simulator that supports both structured and unstructured overlays. It applies discrete-event simulation to simulate the processing and exchange of network messages. In addition to supporting three network models of Simple, SingleHost, and INET, it also supports a layered architecture that comprises an application, overlay, and underlay layer [12]. It further supports churn using ParetoChurn and LifeTimeChurn models.

#### Statistics

The simulator records multiple statistical data such as successful or unsuccessful packet delivery, sent, received, or forwarded network traffic per node, and packet hop count. The incorporated Python scripts support the generation of Gnuplot output and easy post-processing of statistical reports.

#### Scalability

OverSim was designed with performance in mind and has successfully simulated network sizes of up to 100,000 nodes. A modern PC simulates a typical Chord with a network size of 10,000 nodes in real-time.

#### Usability/Documentation

The simulator has an outstanding overlay layer's interface to improve, develop and replace the overlay layer's protocol. It also has an excellent GUI for convenient testing and debugging. It has a reusable implementation of overlay protocols. There is an extensive API documentation and source code available on the OverSim website [15].

#### Portability

The various implementations of overlay protocols can be reused for real network applications. Researchers can validate the simulator framework outputs by comparing them to the results like PlanetLab, a real-world test network. Hence, the simulation framework can handle and collect actual network packets and communicate with the same overlay protocol of other

implementations. The use of the Common API and modular design helps the extension with new protocols or features. Module behavior can quickly be tailored by specific parameters in a human-readable configuration file.

#### System Limitations

The OverSim SingleHost instance only simulates a single host like the Internet to be connected to other instances over existing networks.

#### 4.1.17 Optimal-Sim

Optimal-Sim[32] is a Java-based simulator that enables Internet-like topologies to be realistically used in the simulation. The underlying network topologies are generated by a universal topology generation tool called BRITE [32]. Researchers use BRITE to generate internet topologies constituted by routers and autonomous systems (AS).

##### Simulator Architecture

Optimal-sim is a P2P simulator that supports event-driven simulation techniques to generate overlay network topologies. Peer processes are simulated as events such as peer join, peer departure, and peer failure. Hence, the churn model of P2P systems is simulated by message exchanging and dynamic topology change of P2P networks.

##### Statistics

Optimal-Sim uses BRITE's object-oriented architecture to allow users to import from and export to custom topology files for results statistics.

##### Scalability

Optimal-sim shows the capacity to simulate up to 2,000 nodes with network topology optimization and up to 1,000 without network topology optimization in a controlled experiment.

##### Usability/Documentation

It provides an algorithm that describes the API in P2P systems. The algorithm is used to show the whole topology generation and optimization process of the simulator. There is very limited documentation available for optimal-sim.

##### Portability

The BRITE tool of optimal-sim is implemented in Java and C++. It allows extending or combining topologies with other topologies and importing topologies from other topology generators.

##### System Limitations

The current network topology of optimal-sim cannot simulate more than 2,000 nodes but shows more effectiveness as the network grows larger. This proves that when extended, it can be more effective for large-scale networks.

#### 4.1.18 ProtoPeer

ProtoPeer [33][34] is a distributed systems prototyping toolkit, licensed under the GPLv2 license and written in Java. It allows for switching between the live network deployment and

event-driven simulation without changing the application code. The network interface is encapsulated with loss and delay modeling to switch the simulation model to a live network easily.

#### Simulator Architecture

ProtoPeer is a discrete-event simulator with an event-driven engine that supports both structured and unstructured overlays. It uses an event injection system to support churning, i.e., peers' arrival, departure, and failure. The peers communicate with each other through message passing. ProtoPeer applications have their own set of messages, message handlers, timer handlers, and define timers.

#### Statistics

The statistics can be calculated at different aggregation levels: per peer, time window, and measurement tag through the measurement instrumentations by placing appropriate calls to the measurement API in the application code. The average, sum, and variance statistics are computed on the fly and logged into measurement log files for analysis.

#### Scalability

ProtoPeer has been tested to scale up to tens of thousands of peers on a 3GB laptop [33]. Once deployed, live runs do not use any centralized components and can be scaled indefinitely in theory.

#### Usability/Documentation

ProtoPeer applications can be modularized into peerlets that are unit-testable, reusable, and composed together to produce the desired peer functionality[34]. In addition, it enables users to include their own implementation. However, only limited documentation is provided.

#### Portability

ProtoPeer has an API for developing arbitrary message-passing mechanisms that support message queuing, network I/O, and message serialization.

#### System Limitations

ProtoPeer is experiencing stagnant growth that has not seen it reach its full potential. Documentation and online activity were seized a decade ago with no actual results or data to act on.

### **4.1.19 PeerfactSim.KOM**

PeerfactSim.KOM [35][12] is a general-purpose P2P simulator written in Java for distributed CDNs, overlay-based systems, and streaming applications. A framework-like program that is based on the concept of pluggable layers has been designed for the simulator. An XML-based configuration file is used to commence the simulation that denotes the layers incorporated.

#### Simulator Architecture

PeerfactSim.KOM is a discrete-event simulator that supports both structured and unstructured overlays. The simulator comprises a layered architecture: application layer, service layer, overlay layer, transport layer, and network layer that works with the various aspects of the P2P network[35]. Each layer uses one or more interfaces that offer functionality to the remaining

layers. The network layer is used for communication and, as such, supports message-based packet-level transmission extensively. It also uses a churn generator based on a mathematical operation that works the node joining or departing.

#### Statistics

PeerfactSim.KOM uses its architecture for collecting data of ongoing simulations. Logging architecture is used to trace and debug a simulation and a statistics architecture to record the critical data for on-the-fly statistics or post-processing analysis.

#### Scalability

The PeerfactSim.KOM has been tested to simulate over 100,000 peers with a network size of 10,000 for multi-layered systems.

#### Usability/Documentation

The website of PeerfactSim.KOM has extensive documentation available. It implements interfaces at each layer that extend services to the other layers. The simulator also uses the concept of default and skeletal implementations based on the interfaces[12].

#### Portability

The integrated visualization element supports the visualization of the topology and the messages exchanged on the simulated P2P network. The visualizer can organize the peers by viewing the topology presentation or based on the network layers provided coordinates or arranging them in a ring-like topology.

#### System Limitations

PeerfactSim.KOM tried to evaluate the interdependences in multi-layered P2P systems. It incorporates models of lower layers but does not yet cover TCP.

### **4.1.20 OPNET**

OPNET (Optimized Network Engineering Tools) [16][29][36] simulator is one of the many tools from the OPNET Technologies suite. It is a tool used to simulate the behaviour and performance of any network. The main difference between OPNET and other Network Simulators lies in its power and versatility. Initially built for the simulation of fixed networks, OPNET can be used as a research tool or a network design/analysis tool. It inherently has three main functions: modelling, simulation and analysis.

#### Simulator Architecture

OPNET is an event-based network simulation tool that operates at “packet-level”. It includes a high-level graphical user interface constructed from C and C++ source code blocks with a vast library of OPNET specific functions[36]. OPNET is divided into three main domains:

- Network domain that includes networks + sub-networks, network topologies, geographical coordinates, and mobility.
- Node domain includes single network nodes such as routers, workstations, mobile devices, etc.
- Process domain includes single modules and source code inside network nodes such as data traffic source model, IP protocol, etc.

It is also possible to run external code components such as External System Domain (ESD) with OPNET[29].

## Statistics

OPNET employs an integrated GUI-based debugging and analysis feature for statistical analysis. Object-Oriented Programming technique [16] generates a mapping from the graphical design to the implementation of the actual systems. Simulation results and all topologies configurations are presented visually.

## Scalability

OPNET has been reported to scale up to  $10^6$  network size.

## Usability/Documentation

There is extensive documentation and a user manual available for OPNET. It also has a source code available for commercial/educational use.

## Portability

GUI.OPNET enables the parameters to be adjusted and for experiments to be repeated easily based on a discrete event system mechanism.

## System Limitations

OPNET employs a complex GUI operation, and it does not allow many numbers of nodes within a single connected device to be simulated. The sampling resolution limits the accuracy of the results, and the simulation is inefficient with no activity for a long duration.

## 4.2 Domain Specific Simulators

Domain-specific network simulators offer solutions to a specific domain, such as quantum networks with NetSquid and Peer-to-Peer Networks with P2PRealm.

### 4.2.1 NetSquid

NetSquid (NETwork Simulator for QUantum Information using Discrete events) [18][37] is a simulation tool written in Python for simulating accurate quantum networking and modular computing networks under physical non-idealities. It provides us with a design tool for future quantum networks. NetSquid supports the modelling of all physical devices in a network that can be mapped to qubits [37]. Because it is entirely modular, it allows users to set up large-scale simulations of complex networks and explore network design variations.

#### Simulator Architecture

NetSquid integrates several fundamental technologies, such as a discrete-event simulation engine and a specialised quantum computing library. It also supports an asynchronous programming framework for defining quantum protocols and a modular structure for modelling quantum hardware devices. These technologies show NetSquid potential as generic and versatile design software for quantum networks, in addition to modular quantum computing architecture [18].

## Statistics

The simulation is performed for a typically high number of independent runs to collect statistics to determine the network's performance. Because each run is independent, simulations can be highly parallelised and thereby efficiently executed on network clusters.

## Scalability

NetSquid has been simulated on a 9-node network, and its flexibility to choose a quantum state representation allows network scalability of up to 1000 nodes.

#### Usability/Documentation

There is extensive documentation available on the NetSquid website alongside the user manual and source code.

#### Portability

The versatility of the simulator to model a range of networkable quantum devices that depend on different physical principles, such as ion traps, defect centres in diamond, and atomic ensembles, is essential.

#### System Limitations

NetSquid is a relatively new technology still in development. It is yet to be verified by the researchers as a capable modelling and simulation tool for validating quantum technology.

### 4.2.2 DHTSim

DHTSim [1][11] is a structured overlay simulator written in Java to facilitate the teaching of DHT protocols. It is the foundation for teaching the implementation of DHT protocols. A discrete event-based message passing within the JVM is implemented with the Remote Procedure Call (RPC). Identifiers are randomly assigned.

#### Simulator Architecture

DHTSim is a flow-based simulator that implements RPC at the overlay layer and is designed with a discrete event-based message passing simulation engine. Simulator scenarios are defined using a basic script file[13]. Churn can be simulated with two script commands that enable many nodes to join over some time or several randomly selected nodes to leave.

#### Statistics

It does not include much functionality for extracting statistics because it was designed as a foundation for teaching the implementation of DHT protocols.

#### Scalability

DHTSim has been tested to simulate up to 10,000 nodes.

#### Usability/Documentation

The documentation is limited, although it was designed to facilitate teaching with a relatively straightforward API.

#### Portability

Simulator scenarios are defined using a basic script file. Therefore, it partially supports the churn.

#### System Limitations

DHTSim does not simulate node failure and provides little functionality for extracting statistics. It also does not support distributed simulation.

### 4.2.3 P2PRealm

Peer-to-Peer Realm (P2PRealm) [7][10] is a Peer-to-Peer network simulator written in Java to simulate and optimize neural networks. It was conceived as part of the Cheese Factory P2P research project [7]. The simulator has been classified into four segments; P2P network, P2P algorithms, input/output interface, and neural network optimization.

#### Simulator Architecture

P2PRealm is a training generation and field-specified message passing simulator with simulation speed as an essential criterion. It provides support for dynamic networks and parallel programming. By utilizing P2PRealm, one can validate P2P networks for a topology management algorithm and then produce a neural network output[10].

#### Statistics

The query performance of each neural network is recorded as statistics when queries are forwarded in one or more P2P networks.

#### Scalability

P2PRealm has been tested to scale up to 100,000 nodes.

#### Usability/Documentation

P2PRealm uses several input parameters such as query pattern, resource distribution, number of training generations, the neuron structure of neural networks, and the number of neural networks, optimization process through a configuration file. It produces an output file as best and all neural networks of each generation, neighbour and topology distribution, and query routes beginning from each node of the P2P network. P2PRealm uses a P2P Distributed Computing platform (P2PDisCo) to support the distribution of simulation events with multiple machines[10]. The documentation of the simulator is not well defined. However, P2PRealm is still under development and has little documentation available.

#### Portability

P2PRealm uses the Peer-to-Peer Studio (P2PStudio) visualization tool to draw network topology and different graphs.

#### System Limitations

P2PRealm has very poor documentation and has seen its development stopped because of it.

## 5. Results and Analysis

The analyzing components can be conducted based on architecture, statistics, scalability, portability, usability, and documentation. The summary of all the simulator analysis is described in Table 1. Architecture determines that whether it can manage the discrete events for structured and unstructured networks. It can be observed that all the network simulators run on the discrete-event architecture. Furthermore, only GPS, PlaneSim, NS2, NS3, Peerfact-Sim, OPNET and NetSquid have good accuracy and efficiency alongside flexible architecture.

Network researchers and developers must choose a good simulator that allows flexibility in model construction and validation. A good simulator must include a suitable analysis of simulation results, a reliable simulation engine, and statistical accuracy of the simulation

output. Table 2 provides all the information needed to identify what makes a good simulator in summary.

Another concern in network simulations is how to ensure a model is credible and constitutes reality. If this can't be guaranteed, the model has no actual value and can't be applied for reasonable network simulation [38]. Therefore, it is also essential to have a valid and credible simulation model after choosing a suitable simulator for network simulation tasks. Hence, the special features of a simulator can be used to ensure credibility. Since modelling only produces approximate answers, the final result in a simulation study must also be considered. The random number generators must be used to produce credible statistical outputs. Statistics has always been used as a tool for interpreting the results [39].

Table 1. Summary of Various Network Simulators

S/N	Simulator	Architecture	Statistics	Scalability	Usability/Documentation	Portability	Language	GUI Support
1	PeerSim [12]	Discrete-event	Limited statistics	Very high (10 <sup>6</sup> peers)	Poor user manual /command prompt/ detailed API documentation	Designed to be extensible	Java	No
2	PeerThing[19]	Discrete-event	Set of visualizes supported	Very low (2,000)	Good user manual but poor API documentation and uncommented source code	Extensible and reusable	Java	Yes
3	RealPeer[20]	Discrete-event	Limited to analysis and hard to read	Upto 20,000 peers	Well commented source code but poor API documentation	Poor documentation limits extensibility	Java	No
4	Query Cycle[21]	Discrete-event	Limited predefined analysis	Very high (10 <sup>6</sup> peers)	Poor API documentation and user manual	Limited extensibility	Java	Yes
5	Neurogrid[22]	Discrete-event	Unreadable result	Medium (300,000 nodes)	Good API documentation and user manual but uncommented source code	Designed to be extensible	Java	Yes
6	GPS[40]	Discrete-event	Poor information with statistical analysis	Very low (512 nodes)	Poor Documentation	Poor documentation limits extensibility	Java	Yes
7	3LS[7]	Discrete-event	NA	Absymal (20 nodes)	Poor documentation with one short paper	Theoretically extensible but requires practical implementation	Java	Yes
8	P2PSim[9]	Discrete-event	Limited statistics	Very low (3,000)	Poor documentation /Command prompt	Limited extensibility and complex to handle	C++	Yes
9	Overlay Weaver[1]	Discrete-event	Uses message counter to collect comprehensive data	Very low (4,000 nodes)	Good API documentation and user manual but unreadable source code	Limited extensibility	Java	Yes
10	PlanetSim[26]	Discrete-event	Only basic statistics	Medium (100,000 nodes)	Good API documentation and user manual	Designed to be extensible	Java	Yes
11	Narses[28]	Discrete-event	Limited statistics	Low (10,000 nodes)	Extensive API documentation available	Limited extensibility	Java	Yes
12	DHTSim[13]	Discrete-event	Limited statistics	Low (10,000)	Limited documentation	Limited extensibility	Java	No
13	OMNET++ [17]	Discrete-event	Limited statistics	Very low (3,025 nodes)	Extensive API documentation, user manual and source code is available	Extensible and reusable	C++	Yes
14	P2PRealm [10]	Discrete-event	Only basic statistics	Medium (100,000 peers)	Limited documentation	Limited extensibility	Java	Yes

15	NS2[8]	Discrete-event	Limited statistics	Very low (5,000)	Extensive API documentation available	Limited extensibility	C++ and TCL script	Yes
16	NS3[30]	Discrete-event	Available through logging	Very high	Extensive API documentation, user manual and source code is available	Extensible and reusable	C++ and Python	Yes
17	OverSim[15]	Discrete-event	Available	Medium (100,000)	Extensive API documentation, user manual and source code is available	Extensible and reusable	C++	Yes
18	Optimal-Sim[32]	Discrete-event	Limited statistics	Very low (2,000)	Limited documentation	Extensible and reusable	Java	Yes
19	ProtoPeer[33]	Discrete-event	Available	High	Limited documentation	Limited extensibility	Java	No
20	Peerfact-Sim[35]	Discrete-event	Available	Medium (100,000)	Extensive API documentation, user manual and source code is available	Extensible and reusable	Java	Yes
21	D-P2P-Sim[25]	Discrete-event	Available	Medium (400,000)	Limited documentation with one short paper	Extensible and reusable	Java	Yes
22	OPNET[16]	Discrete-event	Available	Very high	Extensive API documentation, user manual and source code is available	Extensible and reusable	C and C++	Yes
23	NetSquid [37]	Discrete-event	Available	Very low (1,000)	Extensive API documentation, user manual and source code is available	Extensible and reusable	Python	Yes

## 6. Conclusion

In this paper, we evaluate the performance of twenty-three network simulators based on different parameters. Most of the network simulators examined have some functionality missing which might be of importance. While poor documentation is an obstacle, it is an obstacle that can be overcome, but it is unacceptable that most of the simulators have no mechanism that supports a user to collect statistics of a simulation run. Only OverSim, NS3, ProtoPeer, Peerfact-Sim, D-P2P\_sim, OPNET, and NetSquid have a mechanism that collects statistics. We believe that the poor state of existing network simulators is why much-published research uses custom-built simulators. However, the problem does not go away; it only complicates the task of validating research and reproducing results. Hence, we need credible simulators that offer model flexibility, perform large-scale network simulations, and include appropriate analysis and accuracy of simulation results. We conclude that NS3, OMNET, and OPNET can be extended and modified to fulfil network simulation requirements based on the parameters we examined.

Given the current state of simulators used for networking, we believe that there is a need for a network simulator that meets the requirements of network researchers. As a future direction, we believe that this can be achieved by extending NS3, OMNET, and OPNET simulators to meet the needs of researchers. The next priority would be to implement a roadmap for the extension project and seek feedback from the network simulation research communities concerning the challenges ahead.

Table 2. Architecture Comparison of Network Simulators

S/N	Simulator	Structured/ Unstructured	Engine Mode	Distributed Simulation	Special Features	Accuracy	Efficiency
1	PeerSim	Both	Event and cycle based	N/A	N/A	Moderate	Less
2	PeerThing	N/A	Event based	Yes	System behaviour and System Scenario	High	High
3	RealPeer	Unstructured	Discrete - event	N/A	Development environment; reproducible results	Moderate	Moderate
4	Query Cycle	Unstructured	Query based	N/A	For file sharing	Less	Less
5	Neurogrid	Both	Discrete event based	N/A	N/A	Moderate	Less
6	GPS	Both	Discrete event based	Yes	Macroscopic models	High	High
7	3LS	Unstructured	Event based	N/A	3 separate architecture levels, Network, Protocol and User	Less	Less
8	P2PSim	Structured	Discrete event based	No	For understanding P2P source code	Less	Less
9	Overlay Weaver	Structured	Discrete- event based	Yes	Construction toolkit for multicasting and DHT services	Moderate	Moderate
10	PlanetSim	Both	Discrete- event based	N/A	For Chord and Symphony networks	High	High
11	Narses	Structured	Discrete - event	Yes	For large distributed networks	High	High
12	DHTSim	N/A	Discrete - event	No	For DHT Protocols	Moderate	Moderate
13	OMNET++	Structured	Discrete- event	Yes	Supports alternative languages such as Java and C#	High	High
14	P2PRealm	Unstructured	Discrete- event	Yes	Optimize neural networks	Moderate	Moderate
15	NS2	Both	Discrete- event	Yes	Hybrid of C++ and OTCL	High	High
16	NS3	Both	Discrete- event	Yes	Hybrid of C++ and Python	High	High
17	OverSim	Both	Discrete- event	N/A	For Simple, SingleHost and INET	Moderate	Moderate
18	Optimal- Sim	N/A	Discrete- event	N/A	Uses BRITE tool	Less	Less
19	ProtoPeer	Both	Discrete- event	N/A	Prototyping toolkit	Moderate	Moderate
20	Peerfact- Sim	Both	Discrete- event	Yes	Layered architecture	High	High
21	D-P2P-Sim	N/A	Discrete- event	Yes	For Unbiased, Realism, Distributed and Extensible	Moderate	Moderate
22	OPNET	Unstructured	Discrete- event	Yes	For modelling, simulation and analysis	High	High
23	NetSquid	N/A	Discrete- event	N/A	For quantum networking	High	High

**Acknowledgement:**

This work was supported by the Petroleum Technology Development Fund (PTDF) Nigeria with grant number PTDF/ED/PHD/MAS/179/17.

## References:

- [1] S. Naicken, A. Basu, B. Livingston, S. Rodhetbhai, I. Wakeman, Towards yet another peer-to-peer simulator, Proc. Fourth Int. Work. Conf. Perform. Model. Eval. Heterog. Networks (HET-NETs' 06). (2006) 1–10.  
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.109.4091&rep=rep1&type=pdf>.
- [2] B. Schmeiser, Simulation experiments, Handbooks Oper. Res. Manag. Sci. 2 (1990) 295–330. [https://doi.org/10.1016/S0927-0507\(05\)80171-9](https://doi.org/10.1016/S0927-0507(05)80171-9).
- [3] A. Dupuy, J. Schwartz, Y. Yemini, D. Bacon, NEST: a network simulation and prototyping testbed, Commun. ACM. 33 (1990) 63–74.  
<https://doi.org/10.1145/84537.84549>.
- [4] S. Keshav, REAL: A Network Simulator, Tech. Rep. 88/472, Dept. Comput. Sci. UC Berkeley. (1988) 1–16. <http://www2.eecs.berkeley.edu/Pubs/TechRpts/1988/CSD-88-472.pdf>.
- [5] S. Keshav, Reflections on “analysis and simulation of a fair queueing algorithm,” ACM SIGCOMM Comput. Commun. Rev. 49 (2019) 46–47.  
<https://doi.org/10.1145/3371934.3371952>.
- [6] E. Weingartner, H. vom Lehn, K. Wehrle, A Performance Comparison of Recent Network Simulators, in: 2009 IEEE Int. Conf. Commun., IEEE, 2009: pp. 1–5.  
<https://doi.org/10.1109/ICC.2009.5198657>.
- [7] N. Kotilainen, M. Vapa, T. Keltanen, A. Auvinen, J. Vuori, P2PRealm - Peer-to-peer network simulator, 2006 11th Int. Work. Comput. Model. Anal. Des. Commun. Links Networks. 2006 (2006) 93–99. <https://doi.org/10.1109/CAMAD.2006.1649724>.
- [8] R. Khan, S.M. Bilal, M. Othman, A Performance Comparison of Network Simulators for Wireless Networks, A Perform. Comp. Netw. Simulators Wirel. Networks. (2013) 1–6. <https://arxiv.org/ftp/arxiv/papers/1307/1307.4129.pdf>.
- [9] M. Ebrahim, S. Khan, S.S.U.H. Mohani, Peer-to-Peer Network Simulators: an Analytical Review, (2014). <http://arxiv.org/abs/1405.0400>.
- [10] N.I. Sarkar, S. Member, S.A. Halim, A Review of Simulation of Telecommunication Networks: Simulators, Classification, Comparison, Methodologies, and Recommendations, J. Sel. Areas Telecommun. . (2011).  
<http://orapp.aut.ac.nz/bitstream/handle/10292/4145/Nur-Syaf-Simulation-CberJ.pdf?sequence=2&isAllowed=y>.
- [11] S. Naicken, A. Basu, B. Livingston, S. Rodhetbhai, A survey of peer-to-peer network simulators, Proc. Seventh Annu. Postgrad. Symp. Liverpool, UK. (2006).  
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.136.7073&rep=rep1&type=pdf>.
- [12] F. Chowdhury, J. Furness, M. Kolberg, Performance analysis of structured peer-to-peer overlays for mobile networks, Int. J. Parallel, Emergent Distrib. Syst. 32 (2017) 522–548. <https://doi.org/10.1080/17445760.2016.1203917>.
- [13] A. Basu, S. Fleming, J. Stanier, S. Naicken, I. Wakeman, V.K. Gurbani, The state of peer-to-peer network simulators, ACM Comput. Surv. 45 (2013) 1–25.  
<https://doi.org/10.1145/2501654.2501660>.

- [14] S. Surati, D.C. Jinwala, S. Garg, A survey of simulators for P2P overlay networks with a case study of the P2P tree overlay using an event-driven simulator, *Eng. Sci. Technol. an Int. J.* 20 (2017) 705–720. <https://doi.org/10.1016/j.jestch.2016.12.010>.
- [15] A. Zarrad, I. Alsmadi, Evaluating network test scenarios for network simulators systems, *Int. J. Distrib. Sens. Networks.* 13 (2017) 155014771773821. <https://doi.org/10.1177/1550147717738216>.
- [16] M. Chen, Y. Miao, I. Humar, *OPNET IoT Simulation*, 2019. <https://doi.org/10.1007/978-981-32-9170-6>.
- [17] A. Varga, R. Hornig, An overview of the OMNeT++ simulation environment, *SIMUTools 2008 - 1st Int. ICST Conf. Simul. Tools Tech. Commun. Networks Syst.* (2008). <https://doi.org/10.4108/ICST.SIMUTOOLS2008.3027>.
- [18] T. Coopmans, R. Knegjens, A. Dahlberg, D. Maier, L. Nijsten, J. de Oliveira Filho, M. Papendrecht, J. Rabbie, F. Rozpędek, M. Skrzypczyk, L. Wubben, W. de Jong, D. Podareanu, A. Torres-Knoop, D. Elkouss, S. Wehner, NetSquid, a NETwork Simulator for QUantum Information using Discrete events, *Commun. Phys.* 4 (2021) 164. <https://doi.org/10.1038/s42005-021-00647-8>.
- [19] M.F.R. Ms-cs, Modelling Virus Propagation in P2P Networks, *Int. J. Comput. Sci. Issues.* 9 (2012) 580–587. <https://core.ac.uk/download/pdf/25848441.pdf>.
- [20] D. Hildebrandt, W. Hasselbring, Simulation-based development of Peer-to-Peer systems with the RealPeer methodology and framework, *J. Syst. Archit.* 54 (2008) 849–860. <https://doi.org/10.1016/j.sysarc.2008.01.010>.
- [21] M.T. Schlosser, T.E. Condie, S.D. Kamvar, Simulating a File-Sharing P2P Network, *Distribution.* (2003) 1–5. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.8888>.
- [22] A.A. Jamal, W.J. Teahan, Lightweight Blackboard Resource Discovery Mechanism For Unstructured Peer To Peer Networks, 35 (2017) 10–13. <https://doi.org/10.5829/idosi/wasj.2017.10.13>.
- [23] Y.H. Guo, L. Liu, Y. Wu, J. Hardy, Interest-aware content discovery in peer-to-peer social networks, *ACM Trans. Internet Technol.* 18 (2018). <https://doi.org/10.1145/3176247>.
- [24] W. Yang, N. Abu-Ghazaleh, GPS: A general peer-to-peer simulator and its use for modeling BitTorrent, *Proc. - IEEE Comput. Soc. Annu. Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst. MASCOTS. 2005* (2005) 425–432. <https://doi.org/10.1109/MASCOTS.2005.31>.
- [25] V. de C. Fernandes, *Distributed Peer-to-Peer Simulation*, (2011). <https://www.inesc-id.pt/ficheiros/publicacoes/7189.pdf>.
- [26] P. García, C. Pairet, R. Mondéjar, J. Pujol, H. Tejedor, R. Rallo, PlanetSim: A new overlay network simulation framework, *Lect. Notes Comput. Sci.* 3437 (2005) 123–136. [https://doi.org/10.1007/11407386\\_10](https://doi.org/10.1007/11407386_10).
- [27] PlanetSim: An Overlay Network Simulator, (2006). <https://sourceforge.net/projects/planetsim/files/planetsim/>.
- [28] T. Giuli, M. Baker, Narses: A Scalable Flow-Based Network Simulator, *Arxiv Prepr.*

- Cs0211024. (2002) 6. <http://arxiv.org/abs/cs/0211024>.
- [29] M. Kabir, S. Islam, M. Hossain, S. Hossain, Detail comparison of network simulators, *Int. J. Sci. Eng. Res.* 5 (2014) 203–218. <https://www.ijser.org/researchpaper/Detail-Comparison-of-Network-Simulators.pdf>.
- [30] S. Rampf, *Network Simulation and its Limitations*, (2013). [https://doi.org/http://doi:10.2313/NET-2013-08-1\\_08](https://doi.org/http://doi:10.2313/NET-2013-08-1_08).
- [31] K.P.-J. 1 and M.-V.F. 2 Igor Linkov, Benjamin D. Trump, *Computer Network Simulation with ns-3: A Systematic Literature Review*, *Remote Sens.* 6 (2017). <http://www.mdpi.com/2071-1050/5/12/4988/htm>.
- [32] Haoyi Wan, N. Ishikawa, Design and implementaion of a simulator for peer-to-peer networks: optimal-sim, in: *PACRIM. 2005 IEEE Pacific Rim Conf. Commun. Comput. Signal Process.* 2005., IEEE, 2005: pp. 105–108. <https://doi.org/10.1109/PACRIM.2005.1517236>.
- [33] W. Galuba, K. Aberer, Z. Despotovic, W. Kellerer, *ProtoPeer: Distributed systems prototyping toolkit*, in: *2009 IEEE Ninth Int. Conf. Peer-to-Peer Comput.*, IEEE, 2009: pp. 97–98. <https://doi.org/10.1109/P2P.2009.5284513>.
- [34] E.M. Ahrari, *The Proto-Peer Competitor and the Hegemon*, in: *Gt. Powers versus Hegemon*, Palgrave Macmillan UK, London, 2011: pp. 26–75. [https://doi.org/10.1057/9780230348431\\_2](https://doi.org/10.1057/9780230348431_2).
- [35] K. Graffi, *PeerfactSim.KOM: A P2P system simulator &#x2014; Experiences and lessons learned*, in: *2011 IEEE Int. Conf. Peer-to-Peer Comput.*, IEEE, 2011: pp. 154–155. <https://doi.org/10.1109/P2P.2011.6038673>.
- [36] S.A. Abdulazeez, A. El Rhalibi, D. Al-Jumeily, *Simulation of Massively Multiplayer Online Games communication using OPNET custom application*, *Proc. - IEEE Symp. Comput. Commun.* 2016-Augus (2016) 97–102. <https://doi.org/10.1109/ISCC.2016.7543721>.
- [37] *NetSquid The Network Simulator for Quantum Information using Discrete events*, (n.d.). <https://netsquid.org/>.
- [38] R.M. Fujimoto, K. Perumalla, A. Park, H. Wu, M.H. Ammar, G.F. Riley, *Large-Scale Network Simulation : How Big ? How Fast ? 2 . Scalability Limits of Network Simulators*, Event (London). (2003). <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.87.5444&rep=rep1&type=pdf>.
- [39] D. Mödinger, J.-H. Lorenz, R.W. van der Heijden, F.J. Hauck, *Unobtrusive monitoring: Statistical dissemination latency estimation in Bitcoin’s peer-to-peer network*, *PLoS One.* 15 (2020) e0243475. <https://doi.org/10.1371/journal.pone.0243475>.
- [40] W. Yang, N. Abu-Ghazaleh, *GPS: A general peer-to-peer simulator and its use for modeling BitTorrent*, in: *Proc. - IEEE Comput. Soc. Annu. Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst. MASCOTS*, 2005: pp. 425–432. <https://doi.org/10.1109/MASCOTS.2005.31>.