HOSTED BY

ELSEVIER

Contents lists available at ScienceDirect

Engineering Science and Technology, an International Journal

journal homepage: www.elsevier.com/locate/jestch

Review

A survey of simulators for P2P overlay networks with a case study of the P2P tree overlay using an event-driven simulator

Shivangi Surati [a,*], Devesh C. Jinwala [a], Sanjay Garg [b]

[a] *Department of Computer Engineering, Sardar Vallabhbhai National Institute of Technology, Surat, Gujarat, India*
[b] *Department of Computer Engineering, Institute of Technology, Nirma University, Ahmedabad, Gujarat, India*

ARTICLE INFO

Article history:
Received 11 July 2016
Revised 15 December 2016
Accepted 30 December 2016
Available online 13 January 2017

Keywords:
Distributed computing
Peer-to-Peer systems
Overlay networks
Peer-to-Peer simulators
Network simulation

ABSTRACT

Implementation of a P2P (Peer-to-Peer) overlay network directly on the realistic network environments is not a feasible initiative as scalability is a major challenge for P2P. The implementation of newly created P2P networks must be analyzed, well tested and evaluated through experiments by researchers and organizations. Various simulators have been developed to analyze and simulate P2P protocols before applying to real applications. However, selection of an appropriate simulator suitable for a given application requires a prior comprehensive survey of P2P simulators. The existing literature survey of P2P simulators has limitations viz. (i) all P2P related simulators have not been included, (ii) all design criteria for comparison and appropriate selection of the simulator may not be considered, (iii) appropriate practical application through the simulation has not been presented to enhance the outcome of a survey. To overcome these limitations, we survey existing simulators with classifications, additional design parameters, limitations and comparison using various criteria. In addition, we discuss about common interface concept that creates a generic application to make the implementation of targeted application portable. Not only that, we present a case study of implementation of BATON (BAlanced Tree Overlay Network) and BATON* using an event-driven model of PeerSim simulator that helps the developers to simulate the tree overlays efficiently.

Contents

* Corresponding author.
E-mail addresses: shivangi.surati@gmail.com (S. Surati), dcjinwala@gmail.com (D.C. Jinwala), gargsv@gmail.com (S. Garg)
Peer review under responsibility of Karabuk University.

http://dx.doi.org/10.1016/j.jestch.2016.12.010
2215-0986/© 2017 Karabuk University. Publishing services by Elsevier B.V.

1. Introduction

Peer-to-Peer network architecture has gained very popularity since last decades due to its high performance characteristics viz. *dynamicity, scalability, multiplicity, efficient content distribution and ability to search effectively*. The researchers and various organizations are gradually accepting the concept of independent peers, accomplishing computing tasks cooperatively in a self-organized shared environment, rather than traditional client–server architectures where client processes are dependent on server. The major issue in building the effective P2P overlay is in tying together these multiple autonomous computers into a cohesive system while maintaining above specified characteristics. The scalability and multidimensional data indexing are two major issues that increase the complexity of designing such system [1,2]. The number of P2P systems and protocols have been implemented that include file-sharing P2P networks viz. Napster[3], Gnutella [4], Bittorent [5]; Distributed Hash Table(DHT)-based P2P overlays viz. CAN [6], Chord [7], Pastry [8], Tapestry [9] or hierarchical tree-structured P2P overlays viz. BATON [10], BATON* [11], VBI-tree [12], SDI-tree [13] etc.

As designing of P2P overlay networks faces so many challenges, they are required to be tested after their development and before applying to the real applications. The analytical approach or mathematical modelling can be used at the initial level during design of P2P topology and corresponding algorithms [14]. However, this approach is used for simple systems that may not explore the P2P model in detail and hence, this solution is not feasible while deploying the complete P2P model into real world scenarios. Another approach for the implementation and testing of P2P systems is the simulators that make use of analytical solutions and its results can be validated by experiments with the actual system. However, configuring the P2P overlay networks on real machines or network without testing is not practicable. It is observed that for P2P research, often the most practical technique is simulation. Various P2P network simulators are available having multiple options for their usage viz. (i) use the simulator readily for implementation of P2P network, (ii) design your own simulator after referring the existing simulators or (iii) extend or modify the modules of the existing simulators based on the requirements of application. Majority of the simulators are studied and surveyed in [15–22]. However, there is a scope for improvement in the existing survey of P2P simulators as follows:

- New simulators are being proposed nowadays with the increasing demand and requirements of the P2P systems. The existing survey [16,17,19–21] may lack in the discussion about the new simulators viz. *ProtoPeer* [23], *PeerfactSim.KOM* [24] and *D-P2P-Sim* [25]. In addition, a survey needs to be comprehensive instead of discussing in brief about the simulators.
- With the development of new P2P simulators, additional properties or criteria are also required to be considered to compare the design and performance of P2P simulators.
- In addition, the new concepts such as *having a portable common interface or framework model* in order to create simulator models of the same application on different simulators [26] or *deploying an initial model of a P2P system iteratively into the intended real P2P system* using RealPeer [27] are required to be discussed with the evolution of P2P simulators.
- Majority of the survey discusses and compares various P2P simulators based on their properties. The existing literature survey [22] considers the implementation of the well-known topologies such as BitTorrent [5], GnuTella [4] or Chord [7] only. However, the tree overlays are efficient data structures for P2P networks as they reduce the search cost in terms of logarithmic to the base m, where m is fanout of the tree. To the best of our knowledge, a detailed case study guiding about an applicability of a simulator for the tree overlays with the experimentation is not represented in the existing literature.

These observations motivated us to reemphasize on the survey of P2P simulators with recent simulators, new perspectives and case study. Thus, our contributions in this paper are as follows:

- We make a comprehensive survey of the P2P simulators that also includes the study of the new P2P simulators viz. *ProtoPeer* [23], *PeerfactSim.KOM* [24] or *D-P2P-Sim* [25].
- We compare and categorize the simulators as per the underlying protocols they implement and their applicability in various fields i.e. *whether the simulator is protocol-specific, generic or domain-specific*. This categorized survey is useful to researchers

and business communities for identifying the P2P simulator appropriate for the required application.

- We also summarize the favorable aspects and limitations of each generic simulator and discuss about the P2P applications and research directions for each of them.
- In addition, we discuss about the generic application interface for the layered simulators that is presented recently to design a common interface for different simulators.
- Lastly and importantly, we present a case study of the implementation of BATON (BAlanced Tree Overlay Network) and BATON* using an event-driven mode of the PeerSim simulator. An event-driven mode of the PeerSim simulator is not well-documented and hence, this case study is useful to the developers opting the use of more realistic event-driven simulation. In addition, PeerSim simulator supports the implementation of the graph overlays efficiently and hence, this case study is a running example for the researchers working on graph or tree overlay networks.

We discuss in detail the properties of various P2P simulators in Section 2. Various P2P network simulators are presented and classified according to their properties in Section 3 followed by the discussion about them in Section 4. Next, we present a case study about the implementation and experimentation of the binary and *m*-ary tree overlay using event-driven mode of the PeerSim simulator in Section 5. Lastly, we discuss the conclusion of the survey of P2P simulators and results of the case study in Section 6.

2. Properties of P2P simulators

The P2P simulators can be compared based on the criteria as follows that helps in selecting the P2P simulator best suited for the implementation and testing of the application [17,19–21]:

2.1. Simulator architecture

The architecture basically specifies the characteristics in design and functioning of the simulator, basic features and the implementation details of the simulator. The different views of analyzing the simulator architecture are as follows:

- **P2P structure:** It indicates whether the P2P simulator *supports structured overlays, unstructured overlays or both*. In structured P2P overlay networks, peers are organized according to predefined topology and the algorithms are designed such that they maintain the topology and properties of the network. They typically use Distributed Hash Tables (DHTs) indexing or Tree based indexing. Unstructured Peer-to-Peer networks do not provide any algorithm for organization or optimization of network connections. It is composed of peers joining the network randomly without any prior knowledge of the topology. The user should be able to specify all relevant simulation parameters in a human readable configuration file. The simulator should also be able to provide dynamic behavior of nodes, node failure as well as malicious behavior of nodes.
- **Simulator mode:** The simulator mode indicates whether the simulator is designed to give the support of *discrete event simulation, cycle-based simulation or both*. In discrete event simulation, the operation of a system is represented as a sequence of events. It uses a scheduler that synchronizes message transfers between nodes with addition of delay if necessary. Each event occurs at an instant in time and marks a change of state in a system. Cycle-based simulation is a time-driven sequential simulation where each protocol's actions are executed by every node sequentially in each cycle. Cycle-based protocols can also be run by the event-based engine, but not vice versa.
- **Underlying network simulation:** P2P simulators take a number of different approaches to simulate the underlying network i.e. the simulation layer. They can be classified as *packet-based or flow-based*. Packet-based simulators simulate the packet as well as links and calculate delay, bandwidth and routing for each packet generated or used by simulation, for example, NS-2 [28]. Other simulators are flow-based that usually do not map the underlying network or even do not take the layout of these networks into account and thus, they abstract away the details below the simulator layer. They work at the application layer and use transport protocols like TCP or UDP as communication channels between inter-connected peers. Typically, packet-based simulators take longer to complete a simulation than flow-based simulators due to calculations for each packet in the simulated network.
- **Underlying protocol simulation:** P2P simulators differ in their applicability to the underlying protocols. They can be classified as *generic simulators, protocol-specific simulators or domain-specific simulators*. Generic simulators can be used for simulation of any P2P application. Protocol-specific simulators are designed to simulate a specific protocol and domain-specific simulators simulate the P2P systems in specific field.
- **The support of the distributed parallel simulation:** The speedup of the task is higher if it is executed using parallel distribution on different machines as compared to its speedup when executed sequentially. Thus, whether the P2P simulator *supports simulations to be run across a number of machines* is an important criteria to consider for the distributed environments. This helps in achieving higher scalability or faster simulation runtime. However, parallel execution requires to check the dependency controls during the distribution on different machines.
- **The support of churn:** The behavior of a peer or node in a dynamic environment is simulated with churn rate i.e. *the rate at which the nodes join or leave the network per unit time*. The property also includes whether the simulation keeps track of nodes joining and leaving the network, what levels of churn be simulated or whether the node failure (temporarily or permanently) is handled or not.

2.2. Usability

Usability is used to measure the learning and ease of using the simulator. For this purpose, the document of the simulator should be comprehensive, clearly defined, extensible and easily understandable. In addition, the experimentation and testing scenarios should be convenient for the end users. The script language or interfaces provided by the simulator documents should be expressive and easy to learn.

2.3. Scalability

The feasible network size, generally in terms of the number of nodes, that can be simulated by the simulator defines the scalability of the simulator. The network size of an application of P2P protocol is relatively large and hence, scalability is a very important and challenging property for verifying the performance of a simulator. Thus, if a simulator provides higher scalability i.e. thousands of nodes or more, then it is useful in conducting ongoing experiments that are difficult to implement on thousands of machines in real-world scenarios. Along with that, efficient use of the available computing resources is also important feature to improve the ability to scale.

2.4. Statistics

Another key aspect of a simulator is the results it produces and how it is stored for further reference. The results need to be expressive and easy to manipulate in order to carry out statistical analysis. Mechanisms should exist that allow for the repeatability of experiments such as saving simulator state so that the reproducibility of results can be verified.

2.5. Interactive visualizer

A visualizer such as a GUI should be available to validate and debug new or existing overlay protocols. The visualizer visualizes both the topology of the underlying network and the overlay topology in a customized way.

As our paper also focuses on the implementation of the P2P protocol through a case study, we consider an additional criteria for different simulators as follows in addition to the criteria as discussed:

2.6. P2P protocols implemented

This property includes the basic protocols implemented by the P2P simulator. The simulator can be selected based on the basic implemented protocols similar to the required application. The implementations of inbuilt overlay protocols can be reused or extended for real network applications. The existing simulators can be compared based on these properties. However, all the simulators do not follow all the discussed properties. We discuss various simulators with their properties in the next section.

3. Various P2P network simulators

We survey various P2P simulators and characterize them based on their applicability to the underlying protocols viz. *generic simulators* that can be used for any P2P application, *protocol-specific simulators* that are designed specifically for an existing protocol or *domain-specific simulators* that cover the P2P systems in specific field.

3.1. Generic simulators

Generic P2P simulator offers many common modules for the protocol simulations, users only need to achieve the core part of the protocol or application. The simulator can easily be extended or replaced with protocols and applications as they are defined with clear hierarchy and modular structure. PeerSim, OverSim, 3LS, PlanetSim are few examples of generic simulators.

3.1.1. PeerfactSim.KOM

PeerfactSim.KOM [24] is a Java-based simulator designed for large-scale P2P applications. An XML-based configuration file is used to begin the simulation that denotes the layers included. (http://peerfact.kom.e-technik.tu-darmstadt.de/de/).

Simulator architecture: PeerfactSim.KOM is a *generic, discrete-event* simulator supporting *both structured and unstructured* overlays. The simulator consists of a *layered architecture* viz. application layer, service layer, overlay layer, transport layer and network layer that tries to cover the diverse aspects of a P2P system. One or more interfaces are used at each layer that offer the functionality to the remaining layers. It considers the network layer for the communication and hence, supports message based *packet-level* transmission in detail. PeerfactSim.KOM provides a *churn generator* based on a mathematical function that takes care of the node join or departure.

Usability: An extensive documentation is available on a website of PeerfactSim.KOM. It provides interfaces at each layer that offer services to the other layers. Based on these interfaces, the simulator provides the concept of default and skeletal implementations.

Statistics: PeerfactSim.KOM provides its own architecture for gathering data of ongoing simulations. It uses logging architecture to trace and debug a simulation and a statistics architecture to grab the important data for on-the-fly statistics or for later post-processing.

Interactive visualizer: The integrated visualization component allows for the visualization of the topology and the exchanged messages of the simulated P2P system. Dealing with the presentation of the topology, the visualization can organize the peers based on the provided coordinates of the network layer or arrange them in a ring-like topology.

P2P protocols implemented: Unstructured overlays: GIA, Gnutella 0.4, Gnutella 0.6, Napster; Structured overlays: CAN, Chord, C-DHT, Kademlia, Pastry, Globase.

3.1.2. D-P2P-Sim

D-P2P-Sim [25] is a novel distributed simulation environment written in Java with GUI for P2P simulations. It evaluates the performance of various protocols by integrating set of tools in a single software solution. D-P2P-Sim+ is an enhancement of D-P2P-Sim to deliver multi-million node simulation support, failure-recovery models simulation capabilities and more statistics (http://students.ceid.upatras.gr/papalukg/).

Simulator architecture: D-P2P-Sim is a *generic, event-driven, multi threading* that applies a pooling technique to achieve thousands/millions of nodes using the event driven approach that makes it more realistic simulator. It has key features as follows: (i) *unbiased:* independent from the protocol implemented mechanism to collect performance data, (ii) *realism:* implemented as close as possible to an application level P2P software, (iii) *distributed:* connects multiple computers in a network or even clusters, (iv) *pluggable and extensible:* provides API that is extensible and based on the plug in mechanism of Java. The architecture of D-P2P-Sim is divided into four modules viz. *the message passing environment, the overlay network, the remote services and the simulator's utilities*. The simulator accepts two types of configuration files in XML format as input i.e. one that defines the simulator's parameters and the second that defines the simulations' parameters. D-P2P-Sim *supports churn* for peer join/leave operations.

Usability: D-P2P-Sim provides pluggable and extensible API that is also available for further development. In addition, it includes a sample dummy implementation with all code and details for the basic design of a P2P protocol with its message transaction mechanism, routing management and architectural outline.

Statistics: An integrated Graphical User Interface and includes graphical statistics functionality. The network containing messages is observed by the *network monitor* and filtered by the *network filter* in order to get extracted useful statistical data that will be further processed by the *overlay monitor*.

Interactive visualizer: Uses GUI buttons to plot the results. Visualizer to visualize the topology is not specified in literature.

P2P Protocols implemented: Chord, BATON*.

3.1.3. ProtoPeer

ProtoPeer [23] is distributed systems prototyping toolkit written in Java that allows for switching between the event-driven simulation and live network deployment without changing any of the application code. Loss and delay modeling is encapsulated in the network interface to easily switch the simulation model to live network. (http://sourceforge.net/projects/protopeer/).

Simulator architecture: ProtoPeer is a *generic, discrete-event driven* simulator supporting *both structured and unstructured* overlays. An application developed using ProtoPeer has its own set of messages and message handler as well as defines timers and timer handler. The peers interact with each other through *message* *passing*. ProtoPeer uses an *event injection mechanism to support churn* i.e. peers arrival and departures as well as peer failures.

Usability: ProtoPeer has an API for building arbitrary message passing systems that supports network I/O, message serialization and message queuing. In addition, it allows users to plug in their own implementation. Applications in ProtoPeer can be modularized into *peerlets* which are reusable, unit-testable and can be composed together to achieve the desired peer functionality. However, documentation is not provided in detail.

Statistics: Through the measurement instrumentations by doing calls to the measurement API in the appropriate places in the application code. The statistics can be computed at various aggregation levels: per peer, per time window and per measurement tag. ProtoPeer computes basic statistics viz. average, sum or variance on the fly as well as they can be logged into a measurement log files to analyze later.

P2P Protocols implemented: Chord.

3.1.4. PeerSim

PeerSim [29–31] is a simulator developed in the project named BISON, generally used to simulate large-scale dynamic P2P network protocol. The network is modeled as a list of nodes; a node has a list of protocols and the simulation has initializers and controls. Initializers are executed before the simulation, while controls are executed during the simulation. They may modify or monitor every component. For example, they may add new nodes or destroy existing ones; or they may act at the level of protocols providing them with external input or modifying their parameters. The components i.e. *protocols and controls* are scheduled to be executed by the simulator engine periodically tunable by the configuration file. These can be applied to monitor and collect statistical data or to simulate network churn in dynamic environments viz. node joins, node failure and node departure.

PeerSim is designed based on Java components that is easy to expand and has a considerable size of the simulated nodes. It is currently active and the simulation code is available at http://peersim.sourceforge.net/.

Simulator architecture: PeerSim is a *generic simulator* that can be used to simulate *both structured and unstructured overlays*. It *supports dynamic scenarios such as churn* and other failure models. PeerSim exclusively focuses on extreme performance simulation at the network overlay level of abstraction. It does not regard any overheads and details of the underlying communication network such as TCP/IP stack or latencies and hence, it supports *flow-based routing*. PeerSim offers two types of simulation engines viz. *cycle-based engine and the event-based engine*. In the cycle-based model, all nodes are chosen at random and each node protocol is invoked in turn at each cycle. For the event-based model, a set of messages or events are scheduled in time and node protocols are invoked according to the time message delivery order. The cycle-based engine could achieve great scalability and performance but lose much facility by neglecting the simulation on the transport layer and the physical presence of concurrent conditions. While the event-based engine supports dynamicity, concurrency and is more realistic but decreases scalability of simulation. PeerSim defines *layered architecture* for the simulation.

Usability: The PeerSim modules are easy to understand, reusable and additional components can be plugged in based on the requirements. Rich documentation is available for the cycle-based engine, while event-based engine has poor documentation even though more realistic. However, the cycle-based implementation can easily be converted to the event-based implementation.

Statistics: The components in the form of observers can be implemented to collect the statistics from various nodes during simulation.

Interactive visualizer: Provides implementation class packages that support well known models such as random graph, lattice and BA-Graph for visualizing simulation.

P2P Protocols implemented: Provides predefined protocols for P2P simulation viz. OverStat, SG-1 and T-Man [16]. Various P2P overlays viz. Pastry, Chord, BitTorrent have also been developed by the researchers using PeerSim.

3.1.5. RealPeer

RealPeer [27] is an open source, platform independent, object-oriented software framework written in Java that supports the modelling and simulation as well as the development of P2P systems. (http://sourceforge.net/projects/realpeer/).

Simulator architecture: RealPeer is *generic, highly modular* simulator that uses *discrete-event* simulation and can be used to simulate *both structured and unstructured overlays*. A P2P system in domain model of RealPeer framework is an aggregate that consists of a set of peers and a physical network. The instances of *concept* classes define whether the represented P2P system is to be used in the role of a model i.e. *the classes encapsulate models of their concepts* or a real system i.e. *the classes encapsulate real occurrences of their concepts*. In addition, a P2P application is divided into four layers viz. P2P core, P2P services, application and user interface. RealPeer uses a *message-passing mechanism* to send and receive messages over the physical network using *MessageProtocol*. Over time, the domain models are refined until they correspond to the intended real P2P system at the end of the development process. It does not support parallel execution but targets to add the functionality in future.

Usability: The RealPeer framework's architecture is extensible such that a developer can combine, freely exchange and reuse elements of the framework. The documentation is available on web.

Statistics: The framework defines a set of *ObserverEvents* that encapsulate different types of simulation data that is consumed by the registered Observer plug-ins. The plug-ins export the data to external analysis tools for reusability in different models.

P2P Protocols implemented: GnuTella (version 4).

3.1.6. OMNeT++

OMNeT++ [32,33] is an extensible, modular and component-based simulator written in C++. It can be extended for real-time simulation, network emulation, database integration, supports alternative languages such as C# or Java and several other functions. It is not strictly a network simulator but also have application domain in the fields of multiprocessors and other distributed systems. Main feature of OMNeT++ is reusability of models. It has been successfully used in other areas like the simulation of IT systems, queuing networks, hardware architectures and business processes as well. OMNeT++ also has a number of contributed models that provide P2P protocols or simulations. Multi-tier topologies are supported by OMNeT++. Java interoperability is provided by the JSimpleModule that is an extension that allows OMNeT++ modules to be written in Java. (https://omnetpp.org/).

Simulator architecture: OMNet++ is a *generic, packet-level simulator* that supports *only structured* overlays. It supports *discrete event* simulation in which modules communicate through message passing. In addition, OMNeT++ has support for parallel distributed simulation execution.

Usability: Basically designs simple modules or components and then groups them into compound modules that can be split into simple modules if require. The extensible modules communicate through messages with parameters and then assembled into larger components using a high programming language NED (NEtwork Description). OMNeT++ has an extensive GUI support and due to its modular architecture, the simulation kernel (and models) can be embedded easily into the applications.

Statistics: It helps the user to visualize the interaction by logging interactions between modules to a file. This log file can be processed after (or even during) the simulation run and can be used to draw interaction diagrams.

Interactive visualizer: The OMNeT++ IDE has a sequence chart diagramming tool which provides a sophisticated view of how the events follow each other. The tool can analyze and display the causes or consequences of an event.

P2P Protocols implemented: P2P swarming simulation, P2P LIVE Video Streaming.

3.1.7. OverSim

OverSim [34,35] is an OMNeT++-based (written in C++) open-source simulation framework for overlay and Peer-to-Peer networks. Three network models for OverSim are implemented viz. Simple, SingleHost and INET. In Simple model, data packets are sent directly from one overlay node to another by using a global routing table that also takes care of packet delays depending on node distance in Euclidean space. Due to the low simulation overhead of the techniques, the Simple model leads to a high level of accuracy and the ability to simulate networks with a large number of nodes.

The SingleHost model reuses overlay protocol implementations without code modifications in real networks like PlanetLab. Here, each OverSim instance only emulates a single host that can be connected to other instances over existing networks like the Internet. The INET underlay model is derived from the INET framework of OMNeT++ that includes simulation models of all network layers from the MAC layer onwards. Since the INET framework is not optimized for large scale networks, the code is profiled and hash table based routing and interface caches are added for faster forwarding of data packets that leads to optimization of overlay topologies and reduction of packet delay. (http://oversim.org/).

Simulator architecture: OverSim is a *generic, packet-level simulator* that supports *both structured and unstructured* overlays. It uses *discrete event simulation* (DES) to simulate the exchange and processing of network messages. In addition, OverSim supports three network models viz. Simple, SingleHost and INET. It supports layered architecture that includes application, overlay and underlay layers. It also *supports churn* using LifeTimeChurn and ParetoChurn models.

Usability: The simulator has an excellent overlay layer's interface to develop and replace overlay layer's protocol. It has a wonderful GUI so that it is convenient to test and debug.The implementation of overlay protocols are reusable.

Statistics: Global observer to collect global statistical data. It collects information of sent, received or forwarded network traffic per node, successful or unsuccessful packet delivery and packet hop count.

Interactive visualizer: Similar to OMNet++.

P2P Protocols implemented: Structured Peer-to-Peer protocols viz. Chord, Kademlia, Pastry, Koorde and Broose; unstructured Peer-to-Peer protocols viz. GIA. More specialized overlays for exchanging event messages in Peer-to-Peer based massively multiplayer online games like VAST or the publish/subscribe-based overlay.

3.1.8. Overlay weaver

Overlay weaver [36,37] is a P2P overlay construction toolkit written in Java that supports routing algorithms for researchers in addition to application developers. For application developers, the toolkit provides a common API for higher-level services such as Distributed Hash Table (DHT) and multicast. Algorithm developers can improve new algorithms and their implementations rapidly by testing them iteratively on the emulator. The routing layer is separated from DHT services and multicasting service. Applications relying on the common API depend on specific transport protocol, database implementation and routing algorithm.

Whenever simulation performed using this, many pitfalls are there such as statistic gathering need a lot of work, documentation need to be written and the lack of scalability might need a fundamental redesign. A newly implemented algorithm can be tested, evaluated and compared on emulator that can host tens of thousands of virtual nodes. It enables large-scale emulation and fair comparison between algorithms. Since simulations have to be run in real-time and there is no statistical output, its use as an overlay network simulator is very limited. (http://overlay-weaver.sourceforge.net/).

Simulator architecture: Overlay Weaver simulator is a *generic message-based* simulator that supports *structured overlays only*. It supports RPC using *discrete event message passing* that uses JVM. The protocols can be tested on a real network by emulating RPC using real TCP/UDP. Various nodes interact with each other by sending and receiving messages. *Distributed simulation is possible* but not much experimented with.

Usability: The API is clean and well designed such that source is quite readable. Overlay Weaver interface consists of a small number of command line tools and the emulator. However, documentation is scattered.

Statistics: All the communication via the network can be reported using a message counter during implementation of the messaging service. The message counter enables the statistics for logging and analyzing the logs to be collected in the execution time.

Interactive visualizer: The visualization of the nodes and communication between them is carried out using a messaging visualizer tool. However, the visualizer imposes a burden on an emulator by visualization in addition to doubling the number of messages. This results in reducing the maximum number of emulated nodes.

P2P Protocols implemented: Multiple routing algorithms based on DHT such as Chord, Kademlia, Koorde, Pastry and Tapestry and a distributed environment emulator.

3.1.9. PlanetSim

PlanetSim [38–40] is a discrete-event overlay network simulator written in Java. It uses software engineering techniques to design and implement new protocols and applications easily. By using the well structured design, a clean API is available for the implementation of overlay algorithms and application services. This makes it possible to simulate an application layer service using a number of different overlay algorithms.

It clearly distinguishes between the creation and validation of overlay algorithms (Chord, Pastry) and the creation and testing of new services on the top of the existing overlays. In addition, PlanetSim provides an Ant algorithm that adds intelligently new links on overloaded paths within the Peer-to-Peer overlay and hence, improves the Peer-to-Peer overlay routing performance. (http://ants.etse.urv.es/planet/planetsim/).

Simulator architecture: PlanetSim is a *generic simulator* supporting *both structured and unstructured* P2P overlay networks. It is a *discrete-event* overlay network simulator supporting *packet-based implementation*. PlanetSim hierarchy can be classified into the *layered architecture* i.e. application layer, overlay layer and the network layer to which the Common API tiers 0, 1 and 2 are mapped. The layers can communicate with each other using upcalls and downcalls from the Common API. It *supports churn model* where nodes can join or leave the network dynamically.

Usability: It has a very good and very clear hierarchy API and has its own good extension interfaces for entities. In addition, an existing entity can easily be replaced to deploy the implementation as per the simulation settings. The architecture implements the Common API (CAPI) so that the development and analysis of overlay algorithms is decoupled from that of applications. The documentation is available on the website of PlanetSim [40].

Statistics: Provides basic statistics such as total simulation time and number of messages employed as part of the simulator and in depth statistic capabilities through aspect-oriented programming (AOP).

Interactive visualizer: The PlanetSim can show the network topology as a GML or Pajek outputs and simulation can be saved to disk for reuse.

P2P Protocols implemented: Implemented overlays viz. Chord, Symphony, SkipNet and generic overlay structures such as SingleLinkedRing and LeafSetRouting. In addition, a variety of services like CAST, DHT, and object middleware.

3.1.10. Dnet

Dnet [41] is a highly modular and efficient simulator developed in ISO C/C++ with two major goals viz. ease of use and efficiency as well as scalability. (http://www.csse.uwa.edu.au/wilkia07/dnet/).

Simulator architecture: Dnet is a *generic, discrete-event driven and message-based* simulator designed for distributed simulation of P2P systems. It uses a uniprocessor sequential event-scheduler as well as two distributed event-schedulers viz. derivative implementations of the Chandy-Misra deadlock-avoidance scheme and Jefferson's Time Warp scheme. Dnet has two distinct halves: *the kernel* is responsible for scheduling events and managing the internals of the simulator and *user-modules* interact with the kernel via an API to specify handlers for events. In addition, Dnet distributed simulation framework uses Message Passing Interface (MPI) to perform inter-simulator communication and hence, *supports parallel execution of events with the increasing speedup* using multiple simulators.

Usability: Dnet uses APIs for handling events and MPI to achieve parallel execution of events on multiple simulators. It has implemented an interface to the Stanford GraphBase file-format for loading underlying network topologies from a file. However, Dnet simulator is not very well documented and the website is also not accessible.

Interactive visualizer:X Create an Internet-like substrate using an Internet-topology generator such as GT-ITM, BRITE or Inet.

3.1.11. 3LS (3 Level Simulator)

3LS [42] is an open-source simulator for overlay networks written in Java and designed to overcome the problems of extensibility and usability.

3LS provides a friendly interface for the development of a new P2P protocol. It still takes messages as the basic simulation object in terms of the simulation granularity. However, the simulator could not simulate the detail of the transfer layer and it could not simulate the routing and the sources of the files P2P networks appropriately.

Simulator architecture: 3LS is a unique P2P simulator with *clock-based simulation engine*. It is a *generic simulator* that supports *only unstructured* overlays. The simulator divides the whole structure into *three layers* viz. network layer, protocol layer and user layer, corresponding to different network topologies, protocols and application simulation. The protocol-level is responsible for simulating the P2P-protocols and applications.

Usability: Input information from the user is fed into the network level through a GUI interface or a file. It is not well-documented for P2P simulation.

Statistics: When simulation is completed, result is stored to a file.

Interactive visualizer: A visualization tool named Aisee.

P2P Protocols implemented: Gnutella 0.4.

3.1.12. Optimal-sim

Optimal-sim [43] is implemented in Java that allows realistic Internet-like topologies to be used in simulation.

Simulator architecture: Optimal-sim is a *generic* simulator that supports *discrete event* mode for simulation to generate overlay network topologies. Peer operations such as peer join, peer leave or peer failure are simulated as events. Thus, Optimal-sim simulates the *churn* model of P2P systems by simulating dynamic topology change and message exchanging of P2P networks.

Usability: It provides API to describe the algorithms in P2P systems.

Statistics: using the results managing tool.

Interactive visualizer: The underlying network topologies are generated by a universal topology generation tool BRITE due to the basic features of BRITE viz. flexibility, extensibility, interoperatibility, portability and user friendly.

P2P Protocols Implemented: An example P2P network simulation .

3.1.13. NS-2

NS-2 [28] is targeted at networking research that provides a rich component library that can run on Linux, UNIX, Windows and other operating system platforms. NS-2 performs simulation using a mixture of C++ and OTCL (object oriented version of TCL). The TCL script defines the nodes and characteristics of communication links, while protocols are implemented in C++. NS-2 was rarely used in p2p simulation. However, it has been improved to efficiently support P2P overlays[44].

Simulator architecture: NS-2 is a *traditional simulator* that is not designed for P2P systems. It is a *discrete event* simulator which is easy to configure and program. NS-2 supports *both structured and unstructured* networks. NS-2 can *run in parallel* with a number of other machines. Support of churn is not clearly defined in NS-2.

Usability: It provides substantial support for simulation of TCP, routing and multicast protocol over wired and wireless network arranged in a structured or unstructured manner. Extensive documentation is available for scripting of NS-2.

Interactive visualizer: NS-2 uses NAM (Network AniMator) to provide visualization.

P2P Protocols implemented: GnuTella.

3.2. Domain-specific simulators

The P2P simulators designed for the *specific domain or field* are being developed nowadays.

3.2.1. P2PRealm

Peer-to-Peer Realm (P2PRealm) [45] is an efficient Peer-to-Peer network written in Java simulator for studying *algorithms based on neural networks*. The simulator is divided into four parts viz. P2P network, P2P algorithms, neural network optimization and input/output interface.

Simulator architecture: P2PRealm simulator is a *field-specified, message passing and training generation* based simulator with the speed of simulation as an essential criteria. It *supports parallel programming* and dynamic network.

Usability: P2PRealm takes various input parameters viz. *resource distribution, query pattern, number of training generations, number of neural networks and the neuron structure of neural networks, optimization method* through a configuration file. It provides output files viz. *topology and neighbor distribution, best and all neural networks of each generation, query routes started from each node of the P2P network*. P2PRealm utilizes Peer-to-Peer Distributed Computing platform (P2PDisCo) to allow the distribution of simulation cases to multiple machines. The documentation of the simulator is not well defined.

Scalability: Medium, 100,000 nodes.

Statistics: Statistics of the query performance of each neural network is recorded when the queries are forwarded in one or more P2P networks.

Interactive visualizer: P2PRealm uses Peer-to-Peer Studio (P2PStudio) visualization tool that provides functionalities to draw network topology and different graphs.

P2P Protocols implemented: The P2PRealm simulator contains implementations of various P2P resource discovery algorithms such as Breadth-First Search, Random Walker, Highest Degree Search and optimal path K-Steiner Tree approximation related to neural networks.

Pros

- Specialized P2P simulator concentrating on speed up of neural networks.
- Supports parallel simulation.

Cons

- Documentation is not well defined.

3.2.2. DHTSim

DHTSim [46] is a structured simulator written in Java and designed to facilitate the teaching of DHT protocols. RPC (Remote Procedure Call) is implemented as discrete event based message passing within the JVM. Identifiers are assigned randomly. It is a basis for teaching the implementation of DHT protocols.

Simulator architecture: DHTSim is a *protocol-specific flow-based* simulator designed with *discrete event based message passing* simulation mode. It *does not support distributed simulation* and does not allow for nodes to fail. DHTSim simulator implements RPC at overlay layer. Churn of bytes can be simulated with two script commands which allow a number of nodes to join over a period of time or a number of randomly selected nodes to leave over a period of time. Thus, it *supports the churn partially*.

Usability: API is fairly straightforward and documentation is limited.

Scalability: Medium, 10,000 nodes.

Statistics: Simulator scenarios are specified using a simple script file.

Pros

- Event simulator specifically for DHTs.
- API is fairly straightforward.

Cons

- Node failure is not simulated.
- Does not include much functionality for extracting statistics.
- No support of distributed simulation.

3.3. Protocol-specific simulators

Protocol-specific P2P simulators are used to evaluate the accuracy and effectiveness of a specific protocol. The example includes GnutellaSim [47] that is developed specifically for a popular GnuTella [4] protocol and Freepastry [48] is developed to implement Pastry [8] protocol.

3.3.1. GnutellaSim

Gnutellasim [47] is a scalable simulator that makes comprehensive evaluation of the Gnutella system. It is written in C++ and NS TCL scripting. GnutellaSim is a *protocol-specific packet-level simulator* that enables the complete evaluation of the Gnutella system with a detailed network model. It is the first system that makes use of an *unstructured overlay* network. The framework is designed to be extensible to incorporate different implementation alternatives for a specific Peer-to-Peer system and is portable to different network simulators. GnutellaSim is divided into the application layer, protocol layer and socket adaptive layer. It *supports churn* to implement dynamic environment.

The architecture features of functional isolation and protocol centralized structure make GnutellaSim easily been extended to achieve other P2P protocol simulation. It can run on the platform of NS-2 and PDNS (for large-scale simulation). The additional features include: receiver advertised window, sender buffer, Socket-like APIs, dynamic connection establishment of TCP, and real payload transfer. GnuTella uses the concept of flooding mechanism to send queries across the overlay with a limited scope and hence, not much scalable as the load on each peer grows linearly with the total number of queries and the system size. Thus, the framework of GnuTella is portable to different network simulators. However, it is designed for a specific protocol with poor scalability.

3.3.2. FreePastry

FreePastry [48] is an open-source implementation of Pastry P2P structured overlay network intended for deployment in the Internet. It works at the transport layer using message-passing mechanism. FreePastry defines a set of basic entities used by the application viz. *Node, Application, EndPoint, Message, RouteMessage, Id and NodeHandle* using Common API (CAPI). The API is developed in Java language. Initially FreePastry is developed that allows users to evaluate Pastry network. It is gradually extended to perform further research and development in P2P substrates as well as a platform for the development of applications. FreePastry uses a scheduler to run multiple nodes within the same JVM. The security of nodes is an issue in Freepastry and its new releases are working on providing security from malicious attacks.

4. Discussion

Based on the survey findings and the list of requirements as discussed, it is observed that generic simulators are preferable as compared to domain-specific or protocol-specific simulators. This is because, they contain pluggable and extensible components and interfaces that can be used to develop majority of the applications. In addition, the criteria viz. *underlying network architecture, scalability and underlying protocol implementation* are the important factors to consider while selecting the simulator. The comparison of various generic simulators is shown in Table 1. Majority of the simulators surveyed suffer due to the limited usability and in particular poor documentation. Dnet, P2PRealm and ProtoPeer have poor documentation. Overlay Weaver provides well documented API and source code, but some documentation is missing in it.

As scalability is one of the major concerns of real P2P systems, the scalability of the P2P simulators is often claimed when applied to the real world P2P applications. NS-2 is a popular network simulator with an extensive documentation, but it is designed for performing simulations at the network layer whereas most P2P research is concerned with the application layer. Its scalability is limited and hence, it is less preferable for simulating P2P networks. Same way, survey shows that RealPeer, OMNeT++, Overlay Weaver, Dnet, 3LS, ProtoPeer, and Optimal-sim are also less scalable and hence, they may not be preferred for the simulation of P2P applications. On the other side, PeerfactSim.KOM, D-P2P-Sim, PeerSim and PlanetSim have very high scalability ($\geqslant$1,00,000 nodes) that fulfills the scalability requirements of the real world P2P applications. OverSim is an overlay network simulation framework for the OMNeT++ simulation environment. OverSim is written in C++ language. The simulators written in Java can easily be extensible due

Table 1
A Summary of various generic simulators.

Simulator	Language	Architecture		Scalability
		P2P structure	Simulator mode	
PeerfactSim.KOM	Java	Structured and unstructured overlays	Discrete event	Very high 10^6 peers for simple overlays 10^5 peers for complex overlays
D-P2P-Sim	Java	-	Discrete event	Extremely high, >400,000 peers
ProtoPeer	Java	Structured and unstructured overlays	Discrete event	Approximately 50,000 peers
PeerSim	Java	Structured and unstructured overlays	Query cycle or discrete event	Very high 10^6 nodes using cycle-based, $2.5 * 10^5$ nodes using event-based
RealPeer	Java	Structured and unstructured overlays	Discrete event	20,000 nodes
OMNeT++	C++	Structured overlays	Discrete event	Low 1000 nodes
OverSim	C++	Structured and unstructured overlays	Discrete event	Medium 100,000 nodes
Overlay weaver	Java	Structured overlays	Discrete event	Low 4000 nodes
PlanetSim	Java	Structured and unstructured overlays	Discrete event	Medium 100,000 nodes
Dnet	C/C++	–	Discrete event	10,000 nodes
3LS	Java	Unstructured overlays	Clock-based simulation engine (cycle)	Low 1000 nodes
Optimal-sim	Java	–	Discrete event	10,000 nodes
NS-2	C++ and Object-oriented version of TCL	Structured and unstructured overlays	Discrete event	Low 5000 nodes

to rich set of APIs and plenty of built-in functions as compared to C ++. Depending on the requirements of the P2P overlay network, it is easy to add new features in the existing simulation code written in Java.

The summary of the favorable aspects and limitations of each generic simulator is given in Table 2. It can be observed that PeerSim, PeerfactSim.KOM and PlanetSim can be considered as better simulators for simulation of various P2P applications. They are written in Java, rich in documentation and highly scalable. In PeerSim, only cycle based simulator is well documented, however, more realistic event driven simulation is also provided to meet real world application scenarios. The components to gather the statistical data are helpful in analysis of P2P overlay. In PlanetSim, design and API are thoroughly documented, detailed tutorials are also available, but it has no mechanism to gather statistics, it is done through the available visualizer. PeerfactSim.KOM is highly scalable, has rich documentation and it has its own architecture for statistics gathering.

Moreover, given the current state of simulation used in P2P, a common and generic application interface for P2P simulator has been developed that meets the requirements of P2P researchers.

Towards a common interface: Various simulators differ in simulation architecture, underlying protocol simulation and implementation constructs as well as usability of the simulation. They provide various abstraction levels for different underlay models and the applications can be evaluated at different granularity. The researchers in [26] claim that the existing simulators suffer from a lack of interoperability and portability making the comparison of research results extremely difficult. To overcome this problem, they propose a generic application interface for discrete-event P2P overlay network simulators. The application is implemented only once using the concept of portability. After that, it can be executed on various simulators as well as in a real network environment, thereby enabling a diverse and extensive evaluation.

The basic requirements from various simulators viz. *node management, messages, schedulers, network topology information, statistical and analytical functionality* are combined to design such a generic application interface. Thus, a lot of future implementation overhead can be avoided using this common interface, as overlays and P2P applications that are implemented once can be reused with other simulators.

Lastly, we discuss about the P2P protocols implemented, P2P applications and further research directions of each generic simulator in Table 3. Based on the implementation of various P2P protocols, it is observed that the popular P2P overlays viz. *CAN, Chord, Pastry, Kademlia, GnuTella, BitTorrent* have already been implemented and well tested by various P2P simulators. PeerfactSim.KOM, PeerSim, PlanetSim and OverSim provide simulation of the different types of existing and popular P2P protocols (structured as well as unstructured). These simulators are preferable for the researchers and P2P developers to develop new P2P applications. In addition, we observe that majority of the simulators viz. *PeerfactSim.KOM, ProtoPeer, PeerSim, OverSim, Overlay Weaver, PlanetSim* provide functionality to simulate the DHT-based overlay networks. However, the existing literature may lack in the simulation of another efficient type of the structured overlays i.e. *tree based overlay networks*. Hence, we present a case study of the implementation of the binary and multiway tree overlay network in the next section. To the best of our knowledge, even both the efficient simulators PeerfactSim.KOM and PlanetSim may lack in providing interfaces or components for the implementation of the tree overlay networks. They are required to be extended to implement protocols based on the tree overlay as further research work. However, PeerSim provides a rich set of components and pluggable interfaces for the implementation of the graph/tree overlay protocols apart from well known DHT based protocols. Hence, it can be preferred for the simulation of the graph and tree overlays.

In the following section, we present a case study of the implementation of BATON and BATON* tree overlay networks using PeerSim simulator as it provides various interfaces to implement graph or tree overlays. The main objectives of the case study are as follows:

- To present use of event-driven mode of PeerSim simulator in detail.

Table 2
The favorable aspects and limitations of various generic simulators.

Simulator	Favorable aspects	Limitations
PeerfactSim. KOM	• Offers existing implementations for a variety of P2P protocols • Consists of a modular architecture and flexible tools to configure simulation setups and to create scenarios • Provides logging and statistics architecture for debugging and collection of simulation states • Provides an add-on visualization component for understanding of the simulation of P2P-networks • Supports large scale P2P systems	• May be complex to understand as multiple layers with many interfaces are provided
D-P2P-Sim	• Supports distributed environment • Powerful integrated GUI for statistic data observation, collection and analysis • Implemented as close as possible to an application level P2P software • Extremely high scalability • Enhanced as D-P2P-Sim + to include multi-million node simulation support, failure-recovery models simulation capabilities and more statistics	• No visualizer to visualize the topology
ProtoPeer	• Easy switching between the simulation and live network deployment without changing a line of code • Defines peerlets for re usability of code and unit testing	• Documentation is not provided in detail • No information is provided for the interactive visualizer
PeerSim	• Very high scalability • Cycle based and Event based simulation model • Support some well known models • Supports dynamic network	• Do not have details of underlying communication network • No support of distributed simulation • Only cycle based engine is documented
RealPeer	• Extensible and reusable elements in the framework • Follows the new concept of simulation-based development of P2P systems	• No interactive visualizer provided • Scalability is lower • No support of distributed simulation
OMNeT++	• Vast GUI support • Support for parallel distributed simulation execution • Reusability of simulation models • Multitier topologies are supported • Has a powerful GUI execution environment	• Low scalability • Less protocols implemented for P2P systems
OverSim	• Good GUI interface for validation and debugging new or existing overlay protocols • Highly scalable • Supports IPv4 and IPv6 as well as UDP and TCP [35]	• Overlay protocol has to provide at least a key-based routing interface (KBR) to the application
Overlay weaver	• Capabilities for distributed simulation • API is clean and well designed so that source is quite readable • Provides multiple routing algorithms	• Statistics gathering need a lot of work • Visualizer reduces the performance of the emulator • Documentation is quite scattered
PlanetSim	• Output network topology graph in GML and pajek formats • Provides high quality software, layered design and the entities that can easily be replaced • Available as P2P overlay, latency-aware overlay, bandwidth-aware overlay, multi-overlay simulation as well as Ant algorithms for suitability of researchers • API is clean and well designed so that source is quite readable • The framework is extensible at all levels • Complete transparency to services running either against the simulator or the network	• Possible to visualize the overlay topology at the end of a simulation run, but there is no interactive GUI • A very simplified underlying network layer without consideration of bandwidth and latency costs, hence, its difficult to simulate heterogeneous access networks and terminal mobility
Dnet	• Supports parallel execution using multiple simulators	• Not very well documented • P2P protocol implementation is not given • Statistics gathering is not well defined
3LS	–	• Supports only unstructured overlays • No support of dynamic network • Very low scalability
Optimal-sim	• Simulates churn model of nodes • Supports dynamic network • Provides BRITE interactive visualizer tool	• No support of distributed simulation • Scalability is not good
NS-2	• Supports visualization using NAM (Network AniMator) • Runs parallel with other machines • Provides extensive documentation	• Not a specific P2P simulator as works at network layer • Very low scalability

- To guide about creation of the P2P tree overlays using PeerSim simulator as it supports the implementation of the graph overlays efficiently.

5. Case study: implementation of BATON and BATON* overlay using event-driven mode of PeerSim

The event-driven mode of PeerSim simulator [29] is based on more complex but more realistic approach with the support for concurrency. We first present brief introduction about the overlay structure of BATON and BATON* in this section for better understanding of implementation of tree overlays using PeerSim.

5.1. The structure of BATON and BATON*

BATON (BAlanced Tree Overlay Network) [10] is based on a balanced binary tree structure where each peer (node) has a level and a number that identifies the node in the tree. Each node maintains links to the parent node, two children nodes, adjacent nodes and selected neighbor nodes as shown in Fig. 1. Two sideways routing

Table 3
The P2P Applications and research directions of various generic simulators.

Simulator	P2P applications/protocols implemented	P2P research directions
PeerfactSim. KOM	Unstructured overlays: GIA, Gnutella 0.4, Gnutella 0.6, Napster; Structured overlays: CAN, Chord, C-DHT, Kademlia, Pastry, Globase	Suitable for various unstructured P2P overlays and the structured P2P overlays based on DHT. However, the overlay network simulation can be extended to support the implementation of P2P protocols based on tree topology as further research work
D-P2P-Sim	Chord, BATON*	Suitable for the structured overlays based on both, DHT and tree. However, more P2P protocols are required to be implemented as further research work
ProtoPeer	Chord	Implementing a network model using the MaxMin bandwidth allocation, which would allow the simulation of bandwidth-limited applications such as BitTorrent
PeerSim	Provides predefined protocols: OverStat, SG-1 and T-Man. Protocols implemented: Pastry, Chord, BitTorrent	Provides rich set of interfaces competent to implement the structured DHT as well as tree overlay networks. The documentation and utilization of more realistic event driven mode can be enhanced in future
RealPeer	GnuTella (version 4)	Can be extended for parallel and distributed simulation to overcome the scalability limits and to implement various structured and unstructured scalable P2P systems
OMNeT++	P2P swarming simulation, P2P LIVE Video Streaming	Scalability is required to be enhanced and more P2P systems are required to be simulated
OverSim	Unstructured Peer-to-Peer protocol: GIA; Structured Peer-to-Peer protocols:Chord, Kademlia, Pastry, Koorde and Broose	Suitable for the applications based on a key-based routing interface (KBR), for example, the structured P2P overlays based on DHT
Overlay weaver	Chord, Kademlia, Koorde, Pastry, Tapestry	Suitable for the services based on DHT or multicast applications. More work can be done to enhance the simulator for the other services in future
PlanetSim	Chord, Symphony, SkipNet, SingleLinkedRing and LeafSetRouting	Provides modifiable configuration files to create customized Networks (CircularNetwork, RandomNetwork). Suitable for the generic overlay and structured overlay based on DHT. However, overlay simulation can be enhanced to implement protocols based on the tree overlay as further research work
Dnet	Not specified	P2P protocols are required to be tested
3LS	Gnutella 0.4	Need to enhance the scalability of the simulator to implement P2P overlay networks as further research work
Optimal-sim	An example P2P network simulation	The testing of more structured and unstructured P2P protocols are required as research direction
NS-2	GnuTella	The research extensions are required for implementation of more structured and unstructured P2P protocols and applications

tables viz. *a left routing table* and *a right routing table* contains links to nodes at the same level with numbers that are less (respectively greater) than the number of the source node by a power of 2. A node *n* accepts a new node as a child node only if its both the routing tables are full, otherwise, the join request is forwarded either to the parent node or to an equilevel node from the routing table or to one of the adjacent node. The cost of node join in terms of the number of routing hops and the cost of updating routing tables both are bound to log_2N.

BATON* [11] is an extension of BATON that reduces the cost of node and search algorithms from log_2N to log_mN (m >2) by enlarging the fanout of the tree from binary to m-ary. Each node in BATON* maintains the links to m children nodes, to the parent node, to the adjacent nodes and to the selected neighbor nodes as shown in Fig. 2. The neighbor routing tables (left and right) maintain links to the selected neighbor nodes at the same level having a distance equal to $d * m^i$, where $d = 1, 2,\ldots,m-1$ and $i \geqslant 0$, from the node itself. The maximum size of a routing table of a node at level l is $m * l$. A node can only accept a new joining node as a child if it has full neighbor routing tables but does not have m children. Otherwise, it has to forward the join request to either its parent, its lower level adjacent node or a neighbor node that does not have enough children. In BATON*, the newly inserted node can appear anywhere in the adjacency order. The cost of node join in terms of the number of routing hops in BATON* is bound to log_mN. However, the cost of updating routing tables is bound to $O(m \cdot log_mN)$.

5.2. Event-driven mode of PeerSim simulator

The PeerSim simulator is enriched with the interfaces supporting graph overlay network as discussed. Hence, we select the

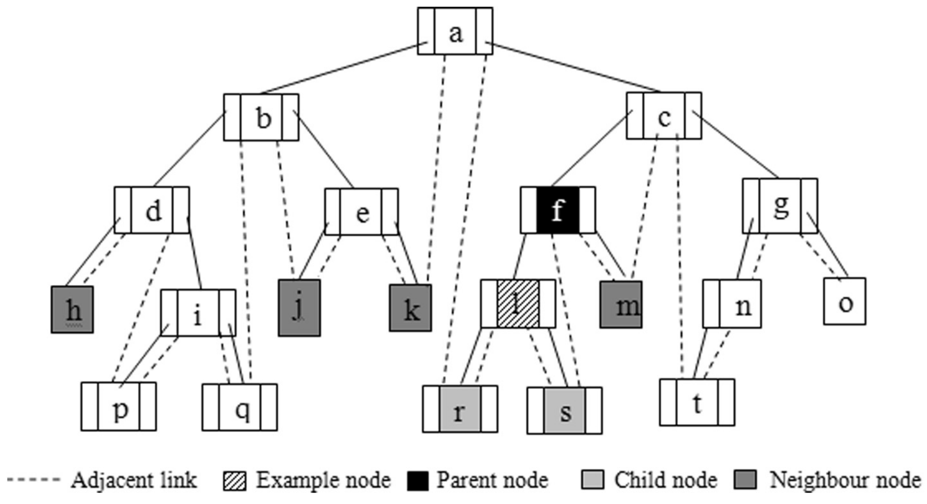


Fig. 1. BATON Structure, fanout $m = 2$.

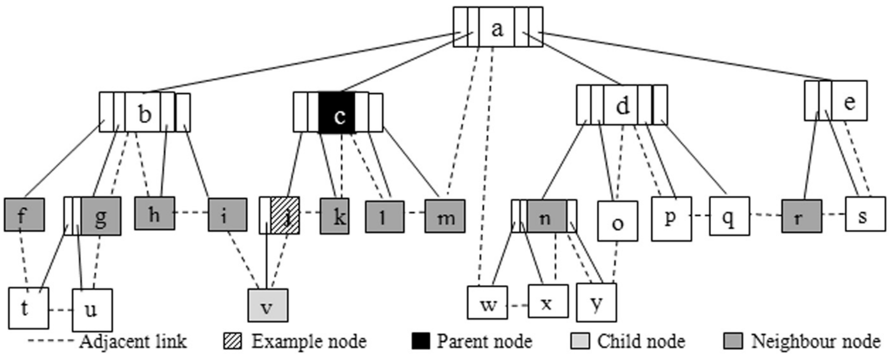


Fig. 2. BATON* Structure, fanout $m = 4$.

PeerSim simulator to simulate the BATON and BATON* tree overlay networks that are subset of the graph structure.

The PeerSim simulator has four fundamental interfaces viz. *node, protocol, linkable and control*. The *node* interface provides access to the protocols it holds and to a fixed ID of the node. *Protocol* provides a prototype that combines different pluggable components and defines an operation to be performed at each cycle, for example, CDProtocol for cycle-driven mode or EDProtocol for event-driven mode. *Linkable* is an interface used to manage node's view. Linkable is typically implemented by the protocols and provides a service to the other protocols to access a set of neighbor nodes. *Controls* are used to define operations that require global network knowledge and management. Initializers, dynamics and observers are various types of controls. The P2P network is composed of nodes.

In event-driven mode, different protocols are scheduled through events for the execution instead of scheduling them with cycles. Events or messages are sent to the different protocols by the control components or by the protocols themselves through the transport layer. The protocols handle these messages and respond to them accordingly. Event-based engine is based on the class EDSimulator from the *peersim.edsim* package. The first step in simulation is to read the configuration file, given as a command-line parameter. The configuration contains all the simulation parameters concerning all the objects involved in the experiment. Then, the simulator sets up the network initializing the nodes in the network and the protocols in them. Each node has the same kinds of protocols; that is, instances of protocols form an array in the network, with one instance in each node. At this point, initialization is performed that sets up the initial states of each protocol. The initialization phase is carried out by the control objects that are scheduled to run only at the beginning of each experiment.

After initialization, the event-driven simulation schedules the execution of all the specified components i.e. *protocols and controls* in the event queue. This scheduling is defined by the {@link Scheduler} parameters of each control component. The first event is then taken from the event queue. If the event wraps a control, the control is executed, otherwise the event is delivered to the destination protocol, that must implement {@link EDProtocol}. This is iterated while the current time is less than the parameter *simulation.end* specified in the configuration file or the event queue becomes empty. The order given in the configuration is respected in case when more control events fall at the same time point. If more non-control events fall at the same time point, they are processed in a random order. The engine also provides the interface to add events to the queue.

5.3. Simulation of BATON and BATON*

In majority of the literature survey, the case study considering the practical scenario that reflects the usage of a simulator is not presented. In this section, we present a case study of simulation of BATON and BATON* using an event-driven mode of the PeerSim simulator.

5.3.1. Network setup for simulation

The design of the network along with the protocols and controls using PeerSim simulator is shown in Fig. 3. Each node follows three protocols viz. *IdleProtocol, transport* (both defined in PeerSim) and *BATONProtocol* (user defined). The BATON overlay network is initialized with a single routing node at level-0 and *m* children data nodes that cover the entire data space. The set of protocols and controls are specified during network setup that works as follows (Fig. 4):

1. *Protocol IdleProtocol* is an inbuilt protocol implementing *linkable interface* that provides a service to the other protocols to access a set of neighbor nodes. The instances of the same linkable protocol class over the nodes define an overlay network.
2. *Protocol UniformRandomTransport* is an inbuilt protocol that creates a transport protocol that is necessary to send messages between nodes. The network traffic operations viz. node join or leave operations are generated using this protocol.
3. *Protocol BATONProtocol* is a user-defined core protocol that simulates the behavior of the BATON overlay application in every node. It uses *IdleProtocol linkable interface* to create the links amongst the nodes and messages will be exchanged by the transport protocol defined above. The *Routing Table* and *BATONMessage* classes are the supportive classes to create the left and right routing tables of a node and to define various types of messages respectively.
4. *Initializer Control WireRegRootedTree* is an inbuilt initializer that implements the *linkable interface IdleProtocol* to create the children node links from the parent node in a hierarchical manner, beginning from the root node. Thus, it creates the multiway tree overlay network specified by the value of the fanout *m* of the tree (>2).
5. *Initializer Control AssignNodeIDs* is a user-defined control initializer that initializes the existing nodes in the network (in this case only 1 node-root node) by giving them unique node Id, level and number according to the fanout *m* and link to the parent node.
6. *Initializer Control BuildNodeStates* is a user-defined control initializer that initializes adjacent links and left and right routing tables for each node present in the initial state. The entries in the routing tables are null if the specific equilevel neighbor node is not present.
7. *Dynamic Control DynamicNetwork* is an inbuilt control routine to create the dynamic network by adding/removing the number of nodes in the overlay network. We have added one node at specific interval (DNET_STEP) to simulate the churn behavior

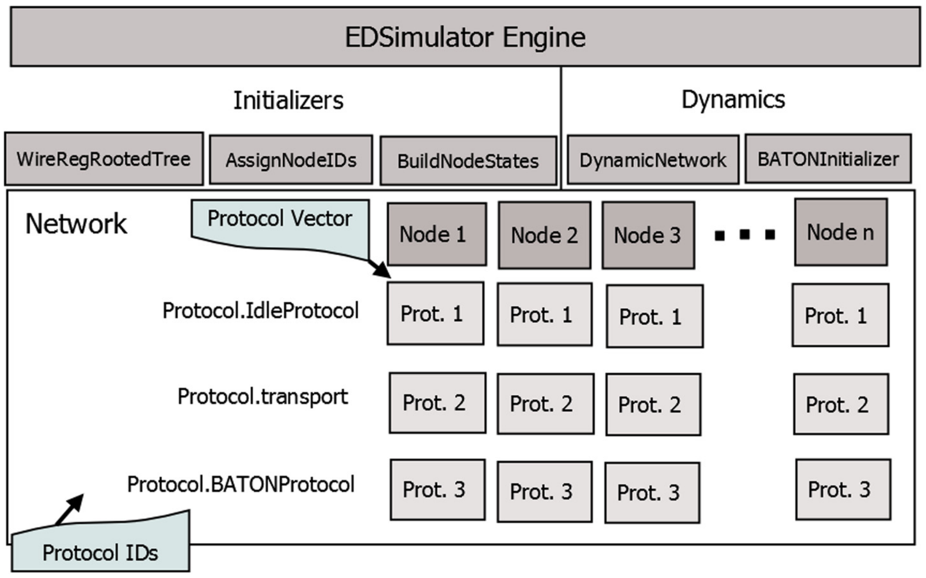


Fig. 3. The BATON and BATON* network for simulation using PeerSim.

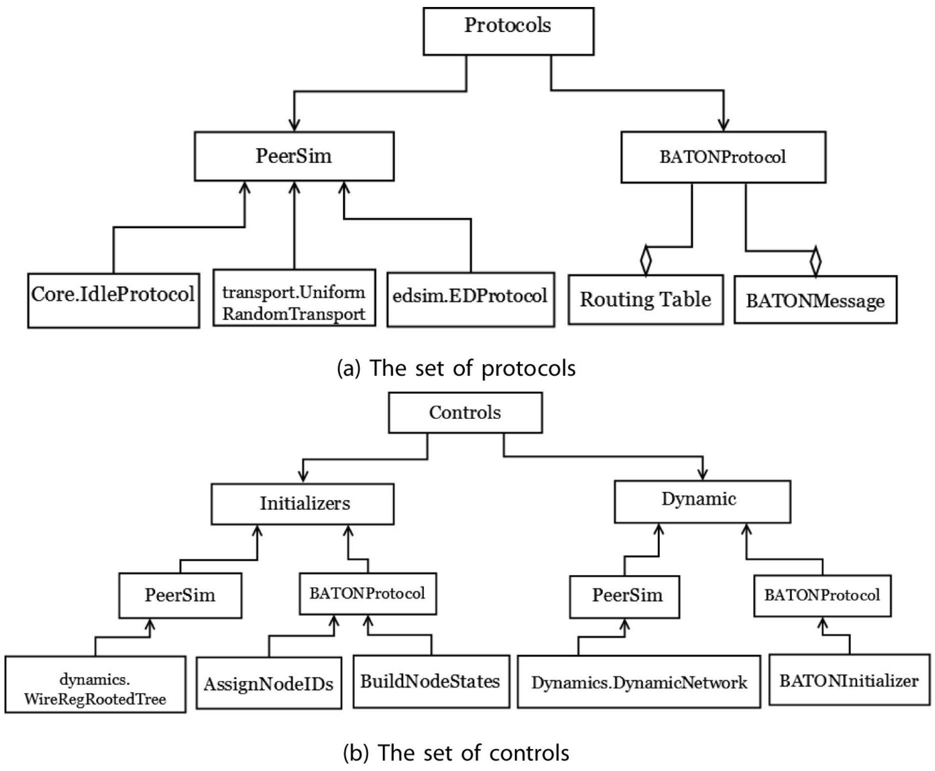


Fig. 4. The set of protocols and controls for implementation of BATON and BATON*.

of the network. It also specifies the minimum and maximum number of nodes in the network. It also uses BATONInitializer control.

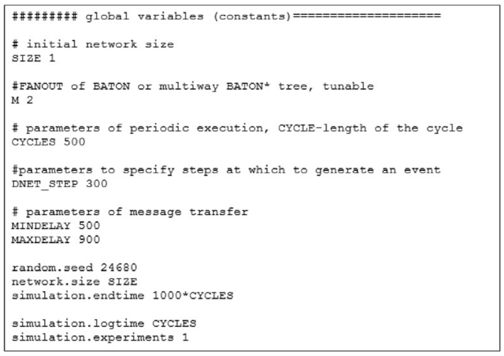
8. *Initializer Control BATONInitializer* is a user-defined control routine that initializes the node by assigning unique node Id to it. In addition, it randomly selects a node that is already part of the BATON that helps the new node to join the network. At the end, it adds this new node join request in a simulator queue to be taken care by the BATONProtocol.

The input parameters and these protocols for the simulation are given using configuration file as shown in next section.

5.3.2. Input configuration file

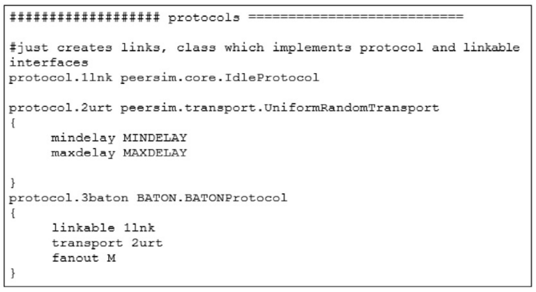
The configuration file is created to tune the input parameters viz. *fanout m, initial size of the network, the step at which to generate the node join request* (for dynamic node join) etc. For the implementation of BATON and BATON*, the value of fanout m is varied i.e. $m = 2$ *for BATON and* $m > 2$ *for BATON** using this configuration file. Thus, the same configuration file can be used to implement BATON or BATON* by tuning the parameter m.

Global parameters: The part of the configuration file containing global parameters for the simulation of BATON and BATON* is shown in Fig. 5a. First few parameters viz. SIZE, M, CYCLES, DNET_STEP, MINDELAY, MAXDELAY define the tunable values



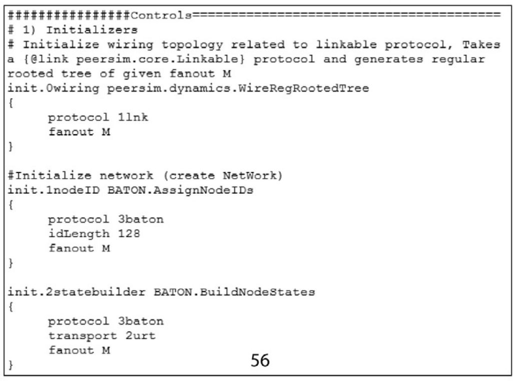
```
######### global variables (constants)====================

# initial network size
SIZE 1

#FANOUT of BATON or multiway BATON* tree, tunable
M 2

# parameters of periodic execution, CYCLE-length of the cycle
CYCLES 500

#parameters to specify steps at which to generate an event
DNET_STEP 300

# parameters of message transfer
MINDELAY 500
MAXDELAY 900

random.seed 24680
network.size SIZE
simulation.endtime 1000*CYCLES

simulation.logtime CYCLES
simulation.experiments 1
```

(a) The global parameters initialization in configuration file



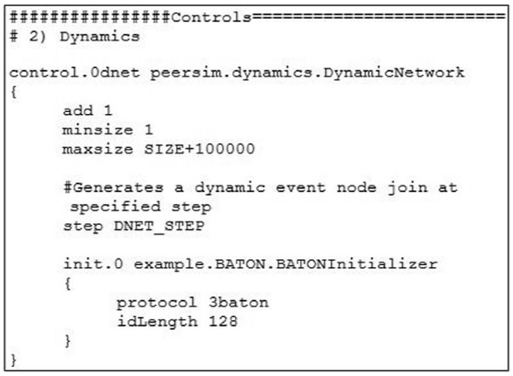
```
#################### protocols ===========================

#just creates links, class which implements protocol and linkable
interfaces
protocol.1lnk peersim.core.IdleProtocol

protocol.2urt peersim.transport.UniformRandomTransport
{
      mindelay MINDELAY
      maxdelay MAXDELAY

}
protocol.3baton BATON.BATONProtocol
{
      linkable 1lnk
      transport 2urt
      fanout M
}
```

(b) The inbuilt and user-defined protocols in configuration file



```
################Controls========================================
# 1) Initializers
# Initialize wiring topology related to linkable protocol, Takes
a {@link peersim.core.Linkable} protocol and generates regular
rooted tree of given fanout M
init.0wiring peersim.dynamics.WireRegRootedTree
{
      protocol 1lnk
      fanout M
}

#Initialize network (create NetWork)
init.1nodeID BATON.AssignNodeIDs
{
      protocol 3baton
      idLength 128
      fanout M
}

init.2statebuilder BATON.BuildNodeStates
{
      protocol 3baton
      transport 2urt
      fanout M
}                                    56
```

(c) The inbuilt and user-defined initializer controls in configuration file



```
################Controls=========================
# 2) Dynamics

control.0dnet peersim.dynamics.DynamicNetwork
{
      add 1
      minsize 1
      maxsize SIZE+100000

      #Generates a dynamic event node join at
       specified step
      step DNET_STEP

      init.0 example.BATON.BATONInitializer
      {
            protocol 3baton
            idLength 128
      }
}
```

(d) The inbuilt and user-defined dynamic controls in configuration file

Fig. 5. The complete configuration file for the implementation of BATON and BATON*.

similar to macros. By tuning these values, the values of the remaining parameters can be changed in the complete configuration file. Next, the value of *random.seed* is used as a seed value while assigning node Ids to the nodes in the network. The parameter *network.size* indicates initial number of nodes in the network. In our simulation, the network is initialized with one node, however, it can be set to the required number of initial nodes in the application. The parameter *simulation.endtime* is key parameter in an event-driven mode that indicates when to stop the simulation. It is initialized with zero at startup time and it is incremented by the message delays. Simulation stops when the event queue is empty i.e. nothing left to do or if all the events in the queue are scheduled for a time later than the specified end time. The simulator prints indication messages regarding the progress of time on standard output at the frequency by the parameter *simulation.logtime*.

Protocols: In protocols, we configure the protocols for the simulation of the BATON and BATON* viz. BATONProtocol that is our defined protocol, IdleProtocol that is the overlay network and UniformRandomTransport specifying the transport layer (Fig. 5b). The BATONProtocol uses IdleProtocol and UniformRandomTransport protocols as well as fanout parameter. The BATONProtocol handles the events such as node join and node leave and counts the number of routing hops when join request is forwarded to the downward nodes. It overrides the *processEvent* method of EDProtocol interface prototyped as follows:

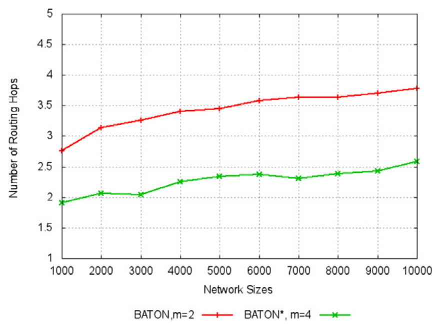
public void processEvent(Node node, int pid, Object event);

where node is the source node that handles the event, pid is the protocol id i.e. protocol id of BATONProtocol and event is an event such as node join. The new node joins the network as a child node of an existing node using the wiring topology of IdleProtocol interface as discussed in the initializer controls. In addition, BATONProtocol creates the links and routing tables of the newly joined node as well as sends the update messages to update the routing tables of the existing neighbor nodes. The other two inbuilt protocols viz. IdleProtocol and UniformRandomTransport provide the interfaces to create the tree topology and to send the messages using transport layer respectively.

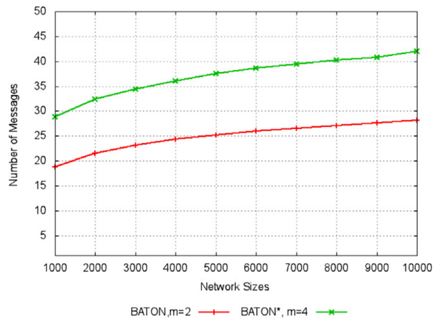
Initializer Controls: The first initializer control is *WireRegRootedTree* in the package peersim.dynamics that takes an IdleProtocol linkable protocol and adds connections between the nodes in the initial network to create the regular rooted tree of BATON and BATON* with *m* number of children nodes. The children nodes of a node are known as its neighbor nodes in IdleProtocol linkable interface that defines the methods such as (i) add a neighbor node i.e. a new node is added as a child node of an existing node when it joins the network, (ii) to count the number of neighbor nodes (child nodes) or (iii) to get a neighbor node (child node) at position *i*. The next user-defined initializer control *AssignNodeIds* assigns the node Ids to the nodes in the network. The last user-defined initializer control *BuildNodeStates* constructs the adjacent links and routing tables of the initial nodes in the network. Thus, these three initializer controls create an initial rooted tree and build initial states of each node in the network (Fig. 5c).

Dynamic Controls: The dynamic control used to actually simulate the churn behavior of the dynamic BATON and BATON* networks. We use *DynamicNetwork* control defined in peersim.dynamics package to change the size of networks by adding and removing nodes (Fig. 5d). The *add* parameter takes the number of nodes to be added in the network (positive value) or to be removed from the network (negative value). The *minsize* and *maxsize* parameters specify the minimum and maximum permissible size of a network in terms of the number of nodes. A node join or leave event occurs at the steps specified by DNET_STEP parameter. The add node event generation gives rise to the control *BATONInitializer* that finds an existing node in the network as a seed node and sends a join request to add a new node in the network. It adds an event i.e. node join in the event queue using *add* method of EDSimulator prototyped as follows:

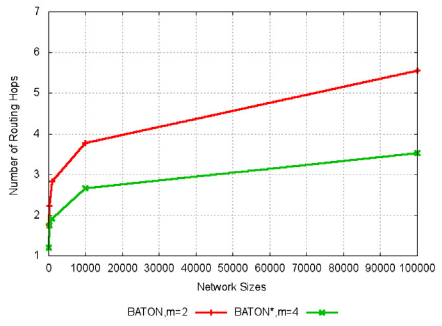
public static void add(long delay, Object event, Node node, int pid);



(a) The number of routing hops for the node join with various network sizes



(b) The cost of updating routing tables for various network sizes



(c) The scalability of the node join

Fig. 6. The routing performance of BATON and BATON*.

where delay is the number of time units (non-negative) before the event is scheduled. Parameter event is the object associated to the event, parameter node is the node associated to the event and parameter pid is the identifier of the protocol to which the event will be delivered i.e. protocol id of a BATONProtocol.

5.3.3. Experimental study

For the experimentation, we use the PeerSim simulator for the implementation and evaluation of the BATON ($m = 2$) [10] and BATON* ($m = 4$) [11], in event-driven simulation mode. We use 2.53 GHz processor, 4 GB RAM, JDK 1.7 (Java Development Kit) to develop our application and NetBeans IDE (Integrated Development Environment) as a programming interface. In our experimentations, we implemented the node join algorithm for number of nodes from 1000 to 10,000 and the average number of routing hops required for each network size for both the BATON and BATON* is calculated. We select the fanout values 2 and 4 for BATON and BATON* respectively. For each network size, the number of messages to update the routing tables after node join is also calculated. In addition, we also vary the number of nodes from 10 to 100,000 to check the scalability and logarithmic to the base m behavior of the node join for both BATON and BATON*.

1. The performance of the node join: In Fig. 6a, we depict a graph showing the average number of routing hops required to find the position for the new node to join with various sizes of network ranging from 1000 to 10000. It is observed that the number of routing hops for the BATON* with fanout $m = 4$ is bound to O($log_m N$) that is lesser than the routing hops of BATON that is bound to O($log_2 N$) for various network sizes. Thus, BATON* gains an advantage over binary trees due to the higher fanout value that reduces the height of the tree and hence, improves the routing performance of node join.

2. The cost of updating routing tables: Next, the average number of update messages required to be sent by the newly joined node to update the routing tables is shown in Fig. 6b with the increasing network size from 1000 to 10000. It is observed that the cost of updating routing tables in BATON* is more as compared to BATON. The number of horizontal links i.e. links to the neighbor nodes increases at each level as the fanout of the multiway tree increases. Hence, the number of update messages to update the routing tables of the newly joined node as well as routing tables of the neighbor nodes also increases in the bound of O($m \cdot log_m N$).

3. The scalability: Lastly, we show the scalability of the BMMI-tree overlay network through the graph in Fig. 6c. We observe that both the trees i.e. BATON and BATON* are scalable with the increasing network size from 10 to 100,000. In addition, as we show in the same graph, the routing performance of the node join algorithm follows the logarithmic nature with respect to the increased network size.

Thus, the simulation results of BATON and BATON* using PeerSim simulator match with the analysis given in the literature. The number of routing hops for the node join decreases with the increased fanout, but at the cost of updating routing tables.

6. Conclusion

After the comprehensive survey and study of various P2P simulators, we conclude that PeerSim, PlanetSim and PeerfactSim.KOM simulators are preferable as compared to the other simulators. They are written in Java, rich in documentation and highly scalable. In addition, we select the PeerSim simulator for the implementation of BATON and BATON* tree overlays using the realistic event-driven mode of PeerSim simulator. The experimental results of BATON and BATON* using event-driven mode of PeerSim simulator match with the analysis given in the literature. As we observed from our experimental results, the number of routing hops for node join algorithm decreases with the increased fanout value (BATON*). However, the cost of updating routing tables is more in BATON* than that of BATON for various network sizes. In addition, the event-driven mode of PeerSim simulation is highly scalable i.e. *up to 100,000 nodes* for tree structured overlay as we derived from the experimental study.

In future, more generic simulators can be developed to increase the scalability and portability of the existing simulators. The same simulation of the P2P overlay protocol can be executed with different simulators as well as on the real network without affecting the code as future research direction.

References

[1] E.K. Lua, J. Crowcroft, M. Pias, R. Sharma, S. Lim, A survey and comparison of peer-to-peer overlay network schemes, IEEE Commun. Surv. Tutorials 7 (2) (2005) 72–93.
[2] C. Zhang, W. Xiao, D. Tang, J. Tang, P2P-based multidimensional indexing methods: a survey, J. Syst. Softw. 84 (12) (2011) 2348–2362.
[3] Napster, http://napster.co.uk/, (accessed: 2016-09-15).
[4] Gnutella, http://rfc-gnutella.sourceforge.net/, (accessed: 2016-10-13).
[5] Incentives build robustness in bittorrent, http://bittorrent.org/bittorrentecon.pdf, (accessed: 2016-10-01).

[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, SIGCOMM Comput. Commun. Rev. 31 (2001) 161–172.
[7] I. Stoica, R. Morris, D.L. Nowell, D.R. Karger, F.M. Kaashoek, F. Dabek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, IEEE/ACM Trans. Networking 11 (1) (2003) 17–32.
[8] A.I.T. Rowstron, P. Druschel, Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems, Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Springer-Verlag, London, UK, 2001, pp. 329–350.
[9] B.Y. Zhao, J.D. Kubiatowicz, A.D. Joseph, Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing, UC Berkeley, 2001. Tech. rep.
[10] H.V. Jagadish, B.C. Ooi, Q.H. Vu, BATON: a balanced tree structure for peer-to-peer networks, VLDB '05: Proceedings of the 31st international conference on Very Large Databases, VLDB Endowment, 2005, pp. 661–672.
[11] H.V. Jagadish, B.C. Ooi, K.-L. Tan, Q.H. Vu, R. Zhang, Speeding up search in peer-to-peer networks with a multi-way tree structure, SIGMOD '06: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, ACM, 2006, pp. 1–12.
[12] H. Jagadish, B.C. Ooi, Q.H. Vu, R. Zhang, A. Zhou, VBI-tree: a peer-to-peer framework for supporting multi-dimensional indexing schemes, ICDE '06: Proceedings of the 22nd International Conference on Data Engineering, 2006, pp. 34–44.
[13] R. Zhang, W. Qian, A. Zhou, M. Zhou, An efficient peer-to-peer indexing tree structure for multidimensional data, Future Gener. Comput. Syst. 25 (1) (2009) 77–88.
[14] A.M. Law, W.D. Kelton, Simulation Modeling and Analysis, third ed., McGraw Hill Higher Education, 2000.
[15] J.B. Harris, A Scalable & Extensible Peer-to-peer Network Simulator, Ottawa-Carleton Institute for Computer Science, Ottawa, Ontario, 2005 (Master's thesis).
[16] S. Naicken, A. Basu, B. Livingston, S. Rodhetbhai, A survey of peer-to-peer network simulators, in: PGNet '06: Proceedings of The Seventh Annual Postgraduate Symposium, Liverpool, UK, 2006.
[17] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman, D. Chalmers, The state of peer-to-peer simulators and simulations, SIGCOMM Comput. Commun. Rev. 37 (2007) 95–98.
[18] M. Baker, R. Lakhoo, Peer-to-Peer Simulators, ACET, 2007. Tech. rep.
[19] R. Bhardwaj, V. Dixit, A.K. Upadhyay, An overview on tools for peer to peer network simulation, Int. J. Comput. Appl. 1 (1) (2010) 70–76. published By Foundation of Computer Science.
[20] H. Bi, T. Guo, B. Qu, Peer-to-peer simulators architecture and design methodology, in: CASE '11 : an International Conference on Control, Automation and Systems Engineering, 2011, pp. 1–4.
[21] H. Xu, S. Ping Wang, R. Chuan Wang, P. Tan, A survey of peer-to-peer simulators and simulation technology, J. Convergence Inf. Technol. 6 (5) (2011) 260–272.
[22] K. Eger, T. Hoßfeld, A. Binzenhöfer, G. Kunzmann, Efficient simulation of large-scale p2p networks: packet-level vs. flow-level simulations, in: UPGRADE '07: Proceedings of the second workshop on Use of P2P, GRID and agents for the development of content networks, ACM, New York, NY, USA, 2007, pp. 9–16.
[23] W. Galuba, K. Aberer, Z. Despotovic, W. Kellerer, Protopeer: A p2p toolkit bridging the gap between simulation and live deployement, in: Simutools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 2009, pp. 60:1–60:9.
[24] D. Stingl, C. Gross, J. Ruckert, L. Nobach, A. Kovacevic, R. Steinmetz, Peerfactsim.kom: A simulation framework for peer-to-peer systems, in: HPCS '11: Proceedings of the International Conference on High Performance Computing and Simulation, 2011, pp. 577–584.
[25] S. Sioutas, G. Papaloukopoulos, E. Sakkopoulos, K. Tsichlas, Y. Manolopoulos, A novel distributed p2p simulator architecture: D-p2p-sim, in: CIKM '09: Proceedings of the 18th ACM Conference on Information and Knowledge Management, ACM, New York, NY, USA, 2009, pp. 2069–2070.
[26] C. Gross, M. Lehn, D. Stingl, A. Kovacevic, A. Buchmann, R. Steinmetz, Towards a common interface for overlay network simulators, in: ICPADS '10: IEEE 16th International Conference on Parallel and Distributed Systems, 2010, pp. 59–66.
[27] D. Hildebrandt, L. Bischofs, W. Hasselbring, Realpeer – a framework for simulation-based development of peer-to-peer systems, in: PDP '07: Proceedings of the 15th EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing, 2007, pp. 490–497.
[28] The network simulator-ns-2, http://www.isi.edu/nsnam/ns/, 2011 (accessed: 2016-10-05).
[29] PeerSim p2p simulator, http://peersim.sourceforge.net/, 2005 (accessed: 2016-09-30).
[30] A. Montresor, M. Jelasity, Peersim: A scalable p2p simulator, in: P2P '09: Proceedings of the IEEE Ninth International Conference on Peer-to-Peer Computing, 2009, pp. 99–100.
[31] I. Kazmi, S. Bukhari, Peersim: An efficient and scalable testbed for heterogeneous cluster-based p2p network protocols, in: UKSim'11: Proceedings of the 13th International Conference on Computer Modeling and Simulation, 2011, pp. 420–425.
[32] OMNeT++, http://www.omnetpp.org/, (accessed: 2016-09-15).
[33] A. Varga, Omnet++, Modeling and Tools for Network Simulation, Springer, Berlin Heidelberg, 2010, pp. 35–59.
[34] I. Baumgart, B. Heep, S. Krause, OverSim: A flexible overlay network simulation framework, in: IEEE Global Internet Symposium, 2007, pp. 79–84.
[35] OverSim: The overlay simulation framework, http://www.oversim.org/, (accessed: 2016-10-05).
[36] Overlay Weaver: An overlay construction toolkit, http://overlayweaver.sourceforge.net/, (accessed: 2016-10-17).
[37] K. Shudo, Y. Tanaka, S. Sekiguchi, Overlay weaver: an overlay construction toolkit, Comput. Commun. 31 (2) (2008) 402–412. special Issue: Foundation of Peer-to-Peer Computing.
[38] P. García, C. Pairot, R. Mondéjar, J. Pujol, H. Tejedor, R. Rallo, PlanetSim: a new overlay network simulation framework, in: T. Gschwind, C. Mascolo (Eds.), Software Engineering and Middleware, Lecture Notes in Computer Science, Vol. 3437, Springer, Berlin/Heidelberg, 2005, pp. 123–136.
[39] J.P. Ahulló, P.G. López, PlanetSim: an extensible framework for overlay network and services simulations, in: Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 2008, pp. 45:1–45:1.
[40] Planetsim, https://github.com/jpahullo/planetsim, 2005, (accessed: 2016-04-14).
[41] A. Wilkins, Distributed simulation of peer-to-peer networks, Tech. rep., The University of Western Australia, 2005. http://undergraduate.csse.uwa.edu.au/year4/Current/Students/Files/2005/AndrewWilkins/CorrectedDissertation.pdf.
[42] N. Ting, R. Deters, 3LS – a peer-to-peer network simulator, in: P2P '03 : Third International Conference on Peer-to-Peer Computing, 2003, pp. 212–213.
[43] H. Wan, N. Ishikawa, Design and implementation of a simulator for peer-to-peer networks: Optimal-sim, in: PACRIM'05: Proceedings of IEEE Pacific Rim Conference on Communications, Computers and signal Processing, 2005, pp. 105–108.
[44] W. Kun, D. Han, Y. Zhang, S. Lu, D. Chen, L. Xie, Ndp2psim: A ns2-based platform for peer-to-peer network simulations, Parallel and Distributed Processing and Applications – ISPA 2005 Workshops, vol. 3759 of Lecture Notes in Computer Science, Springer, Berlin Heidelberg, 2005, pp. 520–529.
[45] N. Kotilainen, M. Vapa, T. Keltanen, A. Auvinen, J. Vuori, P2prealm – peer-to-peer network simulator, in: Proceedings of the 11th International Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks, 2006, pp. 93–99.
[46] DHTSim, http://www.informatics.sussex.ac.uk/users/ianw/teach/dist-sys/dht-sim-0.3, 2006 (accessed: 2016-09-07).
[47] Packet-level peer-to-peer simulation framework and gnutellasim, http://www.cc.gatech.edu/computing/compass/gnutella/, 2003 (accessed: 2016-10-10).
[48] Freepastry, http://freepastry.org/, (accessed: 2016-08-20).