CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
*Concurrency Computat.: Pract. Exper.* (2015)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3716

# Systematic evaluation of peer-to-peer systems using PeerfactSim.KOM

Matthias Feldotto[1] and Kalman Graffi[2,*,†]

[1]*Heinz Nixdorf Institute & Department of Computer Science, University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany*
[2]*Technology of Social Networks, University of Düsseldorf, Universitätsstr. 1, 40225 Düsseldorf, Germany*

SUMMARY

Comparative evaluations of peer-to-peer protocols through simulations are a viable approach to judge the performance and costs of the individual protocols in large-scale networks. In order to support this work, we present the peer-to-peer system simulator PeerfactSim.KOM, which we extended over the last years. PeerfactSim.KOM comes with an extensive layer model to support various facets and protocols of peer-to-peer networking. In this article, we describe PeerfactSim.KOM and show how it can be used for detailed measurements of large-scale peer-to-peer networks. We enhanced PeerfactSim.KOM with a fine-grained analyzer concept, with exhaustive automated measurements and gnuplot generators as well as a coordination control to evaluate sets of experiment setups in parallel. Thus, by configuring all experiments and protocols only once and starting the simulator, all desired measurements are performed, analyzed, evaluated, and combined, resulting in a holistic environment for the comparative evaluation of peer-to-peer systems. An immediate comparison of different configurations and overlays under different aspects is possible directly after the execution without any manual post-processing. Copyright © 2015 John Wiley & Sons, Ltd.

Received 7 May 2015; Revised 7 September 2015; Accepted 30 September 2015

KEY WORDS: P2P; simulation; distributed systems

## 1. INTRODUCTION

Peer-to-peer (p2p) networks emerged in the early 2000s and shaped the view on large-scale overlay networks. Besides the popular file-sharing applications, further use cases also emerged, such as real-time communication [1] or online social networks [2–6]. The general strength of p2p overlays is the harness of the overlay network members that creates an efficient, cost-free infrastructure to host distributed applications. Also in the future, the trend towards decentralized overlays will continue. However, several challenges arise such as reliability, availability, robustness, and security. In order to address these challenges, several p2p protocols have been proposed.

In order to evaluate p2p protocols in large-scale networks with 1,000 nodes and more, simulators offer a reasonable trade-off between testbed-based evaluation and analysis. Using simulators like PeerfactSim.KOM [7], PeerSim [8], or OverSim [9], researchers can investigate the behavior of p2p protocols under a wide range of parameter configurations, user behavior models, and churn models. For a p2p simulator, besides main p2p overlays, it is essential to also support several layers above and below. Minimal requirements are an application layer generating a proper workload as well as a network layer resembling validated network properties such as delay, loss, and bandwidth. Further layers for services such as monitoring, replication or distributed data structures, or transport layer

*Correspondence to: Kalman Graffi, Technology of Social Networks, University of Düsseldorf, Universitätsstr. 1, 40225 Düsseldorf, Germany.
†E-mail: graffi@hhu.de

Copyright © 2015 John Wiley & Sons, Ltd.

protocols such as the User Datagram Protocol (UDP) or the Transmission Control Protocol (TCP) are also favorable.

Besides the functional requirements on a p2p simulator, its usage in the context of detailed evaluation defines notable challenges. The extensive evaluation of new p2p protocols requires parameter studies to be performed with several random seeds. This gives insights into the behavior of the protocol and shows interdependencies of the chosen parameters. Thus, optimal settings for given expected scenarios can be derived. Choosing a large number of seeds (at minimum 10) per parameter configuration allows to identify the common behavior of the protocol and to reduce the impact of individual outliers. Additionally, the results for a single protocol have to be compared with the results of previously existing protocols in order to estimate quality improvements. Ideally, this comparison should be performed for all monitored metrics, over all seeds and for all protocols.

In this paper, we present our extensions for the discrete and event-based p2p simulator *PeerfactSim.KOM* [7, 10–13] – available at www.peerfact.org. These extensions comprise the following features:

- Seed count: Now the number of experiments per setup can be defined initially. The corresponding simulations are carried out in parallel, the results of the individual runs are aggregated, and overall statistical results for the experiment setup are calculated.
- Setup of experiments: Our extension allows to define *all* experiment configurations and *all* parameter variations in addition to the number of seeds initially. For each setup and seed results are created, and all configuration results are combined in comparative plots.
- Monitoring: We integrated a general set of analyzers at all layers, especially at the overlay layer, which grasps all actions during the simulation and creates statistics on the performance and costs of the overlay. For each metric an exhaustive statistical analysis is created consisting of an average, median, standard deviations, and minimum and maximum values.
- Automated output: Besides the generation of result files, corresponding gnuplot [14] files are also created and executed, thus printing all results at the end of the simulations.
- Functional extensions: Next to a churn model, the simulator now comes with an isolation model to study networks where the connectivity of the underlying network is highly dynamic. Furthermore, malicious overlay implementations for Chord and Pastry allow the examination of adversary behavioral in the network.
- Modular configuration files: The configuration can be distributed over different XML files and the single components can be reused in different scenarios. Especially for complex scenarios, this is a big advantage and the simulator comes with default configurations for the most popular components.
- Complete evaluation sets: We run complex sets of comparative evaluations to study different properties in different scenarios. Altogether, more than thousands of output files and more than 2 GB of data were produced from three different configurations. We show a small subset of the results in this article.

The discrete and event-based p2p simulator *PeerfactSim.KOM* now comes with our extensions. It allows to define all protocols to evaluate the range of their parameter settings as well as the number of seeds per setup initially. All simulations are started in one point and are run and calculated in parallel afterwards. The simulator merges all results in the end and generates suitable gnuplot scripts and graphs for all result combinations. Thus, the simulator is very user-friendly, powerful, and fully suited for large-scale, multi-parameter comparative evaluations of p2p protocols. As a second contribution of this article, we apply the approach for systematic comparative evaluation at the use case of simulating p2p overlays for file-sharing applications and structured p2p overlays for lookup resolving. We show both the benefits of the systematic evaluation approach as well as the comparative evaluation results for a set of popular p2p overlays.

In the following, the paper is structured as follows. In Section 2, we review existing p2p and overlay simulators, pointing out the usefulness of the presented evaluation enhancements. In Section 3, we present the architecture and capabilities of PeerfactSim.KOM and give an overview of the different layers and the existing implementations, especially the overlays and their API. Section 4

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

describes our enhancements for the evaluation process in detail, while Section 5 presents the application of the simulator and enhancements applied to the comparative evaluation of popular p2p overlays. In Section 6, we discuss the performance of PeerfactSim.KOM in comparison with two other p2p simulators. Section 7 gives a conclusion and sketches our future work on the simulator.

## 2. RELATED WORK

The evaluation of novel p2p protocols requires the creation of large-scale networks in order to observe the emerging effects of these protocols. Especially, questions with regard to scalability, efficiency, and validity can be reasonably answered only if the considered p2p network is large enough. We identify three main evaluation branches for novel p2p protocols: analysis, testbeds, and simulators. By using methods from theoretical analysis, algorithms are analyzed in a simplified mathematical model. On the one hand, theoretical analysis allows to identify asymptotic performance and cost scales; on the other hand, analytical results highly depend on the assumptions made and, further, do not provide insights in cost and performance parameters of average cases. For example, the analysis of the p2p overlay Chord [15] shows that it provides routing in $O(\log(N))$ hops with $N$ being the number of nodes in the network. As the analysis only works on a simplified model, not all environment aspects like delays and churn are considered. Therefore, no insights about the actual expected lookup time are gained. For such detailed information, the network status and capabilities have to be considered. The most direct approach to do this is to use prototypical implementations of the novel p2p protocols within testbeds that offer an isolated and controlled environment with real components and connections. Implementing the p2p protocol and operating it in a controlled (testbed) or uncontrolled (Internet) environment allows to gain insights in accurate performance and cost characteristics of the p2p protocol. Challenges arise in the creation of sufficiently large networks, as even the largest international testbed, the Planet Lab [16], consists only of 1,000 to 2,000 nodes, a fairly small number of nodes in comparison with actual p2p networks. Thus, in order to combine the scalability of analytic modeling as well as the precision of testbed evaluations, simulations have been proposed. They emulate the behavior of the p2p protocols within a controlled, observable environment and make use of realistic network, workload, and churn models. The several peers and connections are no longer real components like in the testbed, but they are simulated by a single application. Network simulators like ns2 [17], ns-3 [18], or OmNet++ [19] aim at the simulation of common routing protocols at the network and data link layer. Because of the increased complexity in these layers, for example because of medium access, the size of the simulated networks is limited to a few hundred nodes in typical scenarios.

Several simulators for p2p networks have been proposed in the last decade that aim at simulating application layer overlays and extending the scalability of the considered networks to sizes beyond 10,000 nodes. Naicken gives an overview of popular p2p simulators in [20]. Discrete event-based simulators are mainly used by the community, while round-based simulators are ignored because of their lack of precision. Besides the technical issues, creating and maintaining a p2p simulator is a challenging task. OverSim [9], DEUS [21], and PeerfactSim.KOM are still actively maintained, while PeerSim [8] and Overlay Weaver [22] are still available but are not actively maintained. OverSim is a flexible p2p simulation framework, written in C/C++, and built on top of the discrete event-based simulation system OMNeT++ [19]. DEUS [21] is an interesting universal simulator that allows to simulate various protocols, also p2p protocols. PeerfactSim.KOM is written in a Java-based simulation framework, which aims at the support of the simulation of multiple layers in a p2p system.

PeerSim [8] is a Java-based p2p simulator, which is mainly designed to support extreme scalability and dynamism. PeerSim has been widely used; however, its latest version dates back to 2009. In order to provide high performance with more than 1,000,000 simulated nodes, the simulator uses simplified assumptions about the communication protocol stack for example by ignoring details of the transport layer. Furthermore, PeerSim supports two kinds of simulation models: cycle-based and event-based models. Using the foremost model, nodes perform specific actions in a sequential order in each periodic cycle. Concurrency between the simulated nodes is not simulated with the

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

cycle-based model. The event-based model is more realistic because actions are invoked according to scheduled events. The limitations in the communication protocol stack for scalability reasons cause the layers simulated below the overlay layer to be inaccurate and inflexible for user adaptations. Unfortunately, the simulator only supports the removal of existing nodes or the addition of new nodes. Appropriate models to simulate churn are not given by default but could be implemented easily. Although a Java-class documentation for PeerSim packages is available, there are no tutorials concerning it. In general, the tutorials available for PeerSim do not cover the simulators possibilities in detail, because they mainly focus on the cycle-based model.

ProtoPeer was introduced in [23], promising to allow both the simulation of p2p protocols in large-scale networks as well as the practical usage of the same code in a testbed. Unfortunately, it is also not continued anymore. ProtoPeer [23] is a message-level, event-based p2p simulator, written in Java, which allows to reuse simulated code for real network applications without changing it. For this reason, p2p applications have to be developed by using the ProtoPeer API, which mainly consists of timer and network operations. For real network deployment, simulated timers and transport layer protocols can be exchanged by real timers or networking protocols. This allows the user to provide customized timer and network implementations so that even network models for mobile ad hoc networks might be used as layer below the overlay layer. The authors of ProtoPeer define the simulator's scalability limit at approximately 50,000 nodes by investigating simulations of lookups in Chord [15]. The project website of ProtoPeer, given in [23], has not been accessible during our research and will probably not be reachable again.

ProtoPeer [24] and a set of further p2p simulators, namely, P2PSim [25], PlanetSim [26], GnutellaSim [27], GPS [28], and Query-cycle Simulator [29] have been used in the last decade, but have been abandoned. Overlay Weaver [22] only supports structured overlays up to approximately 4,000 nodes and provides a common API for higher-level services, for example distributed hash tables (DHTs).Although the website of the Overlay Weaver project is partially kept up to date, the documentation of the simulator and its API are incomplete; moreover, the latter is not available. To summarize, only OverSim and PeerfactSim.KOM are, to our best knowledge, still maintained.

We present in this paper an extension to PeerfactSim.KOM that allows to systematically and comparatively evaluate p2p protocols on various layers. The evaluation of layered p2p systems, consisting of user, application, overlay, network, and even potential further layers, is the focus of PeerfactSim.KOM. Using PeerfactSim.KOM, we let the user define the complete simulation setup beforehand: protocols, parameters, and the number of random seeds. With one single setup, all possible parameter and protocol combinations are simulated, and all possible comparative simulation results are created and plotted. Thus, we improve the usability of researching p2p algorithms.

## 3. SIMULATOR ARCHITECTURE

The simulator is a discrete event-based simulator written in Java that focuses on the evaluation of multi-layered p2p systems. In this section, we concentrate on the different functional layers, especially the overlays. Later, in Section 4 we concentrate on the simulation engine, including its configuration and monitoring components.

### *3.1. The different layers*

A detailed overview of the corresponding layers is given in Figure 1. Starting from the lowest layers, the churn models defined in the simulator describe the online and offline behavior of the nodes. Typically, a churn model on the basis of measurements based on [30] is used to reflect the online times of the nodes. In a long-term study, the peer behavior in the KAD network, a DHT based on Kademlia [31], which is part of eDonkey, a p2p file- sharing system with several million simultaneous users, was examined and modeled as a Weibull distribution [32]. But also further models are implemented and available, for example, a static model or an exponential distribution of the events. In addition to churn models, a similar concept for isolation models is included. The isolation ability gives the possibility to simulate an environment in which a part of the network is isolated for some times. One typical scenario is that continents or countries block the communication to the rest of the

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

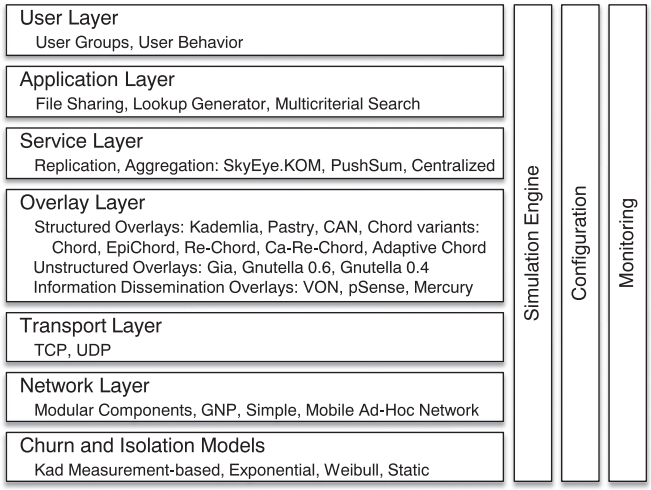



Figure 1. Layered architecture of PeerfactSim.KOM.

world, and thus, the Internet graph is not connected anymore. Another scenario is to simulate opportunistic networks in which communication islands emerge and the connectivity of the underlying network is highly dynamic.

The network layer is in charge of modeling delay, bandwidth, jitter, and loss between two nodes in the simulation. Besides realistic network models based on measurements in the Internet, such as the modular network layer (MOD) or the Global Network Positioning (GNP) network layer, simple and specific network types, for example mobile ad hoc networks, can also be used. The transport layer implements the characteristics of UDP and TCP flows. Thus, it allows to use congestion avoidance and reliable message flows.

The overlay layer operates on top of the network and transport layer and implements structured, unstructured, and information dissemination overlays. A more detailed overview of the overlays is given in Section 3.2 as this is the most important layer of the simulator. All overlays are fit under specific interfaces so that overlays of the same category can be comparatively tested.

The application layer models the workload for a specific application type. For example, the file-sharing application defines the document popularity and the users' query patterns. Besides using the overlays to find corresponding nodes offering desired files, it also initiates a file download. The search application defines a set of items to be searched with a customizable keyword distribution and uses the overlays to index and search the items. Between the overlay and the application layer, the service layer is located. It offers a set of protocols that are considered as enhancements for the overlays, but not as essential. In fact, both the replication as well as the information aggregation protocols abstract from the specific overlay, they use only their generalized interfaces. On top of the applications, a user layer is integrated. It provides the possibility to simulate different user behavior, for example, to simulate different user groups for the applications.

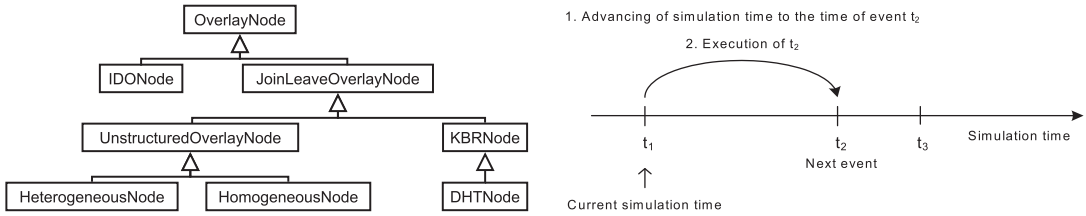
Besides the different layers, the simulator consists of three components for handling the simulation: the simulation engine is responsible for the whole execution, the configuration component enables the user to define different experiments, and the monitoring component produces output for the further analysis. These three components are responsible for the evaluation process, we deal with them in Section 4.

### *3.2. API for overlays and applications*

In this subsection, we focus on the API between the different layers. This enables the exchange of single components without changing the other layers, which is essential for systematic evaluations. Only if a simulator provides this functionality, it is possible to do real comparative evaluations of single aspects. In the evaluations of this article, we focus on the replacement of overlays; therefore, we present a common API for them that is used by the different applications. However, all other layers have similar concepts to replace single components there, for example the network layer.

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe




(a) Hierarchy of overlay interfaces (b) Example of the scheduling queue

Figure 2. Simulator components.

In the overlay layer, we find different interfaces, arranged in a hierarchy (cf. Figure 2(a)). On top, we find the `OverlayNode`, which provides basic operations and properties like its unique identifier. On the next level, we find on the one hand the `JoinLeaveOverlayNode` that provides basic join and leave operations for nodes in a p2p network; on the other hand, we have an `IDONode` that has more special operations for information dissemination networks (IDO) and also more special join/leave operations. For more details on IDOs, please refer to [33]. Below the `JoinLeaveOverlayNode`, we find basic interfaces for structured and unstructured overlay nodes. The `UnstructuredOverlayNode` is further divided into an interface for a `HeterogeneousOverlayNode` and a `HomogeneousOverlayNode`. They both define methods for publishing data to and querying data from the network. On the structured side, we have an interface for a `KBRNode`, which is based on the common API for structured p2p overlays [34] where a common key-based routing (KBR) API is proposed. It provides all methods that a structured overlay node should offer, such as routing and lookups. As a last basic interface, we have the `DHTNode`, which extends the `KBRNode` and is designed for distributed hash tables (DHTs) [15, 31, 35, 36] and therefore offers store and lookup operations.

For all presented interfaces, the most popular overlays are implemented. For unstructured overlays, Gnutella 0.4 [37], Gnutella 0.6 [38], and Gia [39] are implemented as representatives. For the structured variant, Kademlia [31], Pastry [35], Content addressable Network (CAN) [36], and multiple Chord [15] alternatives (EpiChord [40], ReChord [41], Ca-ReChord [42], and Adaptive Chord [43]) are ready to use. Furthermore, the simulator comes with two centralized overlays for comparison: one with a centralized index and one with a centralized storage. Also a Chord and a Pastry implementation with malicious nodes are contained to investigate adversary behavior in the network. There, a subset of nodes forwards messages to the wrong destination or even does not forward them at all to disrupt the proper operation of the network. Basic implementations for information dissemination overlays like VON [44], pSense [45], and Mercury [46] complete the list of available overlays.

The hierarchy of the overlay interfaces offers the possibility to use the API that is needed by the application. If, for example, we only need the join and leave of nodes, we can use an API on a higher level than in a case where we need lookups. To see the use of these APIs, we look at two sample applications in more detail. First of all, there are different lookup generators; they are either based on the DHT or the KBR interface. Their main use is to test and evaluate different overlays and their properties (e.g., the hop count). They need join, leave, and lookup operations and support further ones like periodic lookups with different random distributions. It enables the user to obtain detailed insights in the different properties of p2p overlays in a systemic environment.

A more complex and realistic application is a file-sharing application. It supports different APIs by using individual overlay handlers; next to the DHT and KBR interface, it also has support for unstructured overlays. This application models a realistic file-sharing behavior in a network. Files are chosen from defined file sets, in which file popularieties are modeled, for example, with a Zipf distribution [47]. They can be published by all or only a subset of all nodes. The search for documents can also be managed by the application. While the searching for the correct locations of files is handled by different overlays (through the described interfaces), the upload and download itself is managed by an independent content distribution where different implementations are also possible.

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

With this complex and modifiable combination of components, nearly each file-sharing scenario can be simulated.

Both applications enable us to evaluate the properties of the underlying overlay networks. Which application is the best in each case depends on the concrete goals. Some examples of the use of these applications are shown in Section 5. Further applications exist in the simulator and can also be added. For example, a complex index-based search for items is included in the simulator or applications for special scenarios, for example for information dissemination or aggregation.

## 4. SIMULATION CONTROL AND EVALUATION PROCESS

After presenting the functional layers of the simulator and especially the API for the overlays (cf. Section 3), we now look at the non-functional components, the simulation engine and the configuration and monitoring components. PeerfactSim.KOM is a flexible and mature event-based simulator for p2p systems written in Java. The events follow a timeline that assures sequential processing. Events are part of so-called operations, which are generated either through the entities in the simulation or through an external actions file. One example of such an operation is a lookup in a DHT. Operations help to trace actions on every layer, allowing the protocols to store local state information easily and to react on operation timeouts. Operations on various layers are decoupled allowing for the combined simulation of various protocols in parallel (Section 4.1).

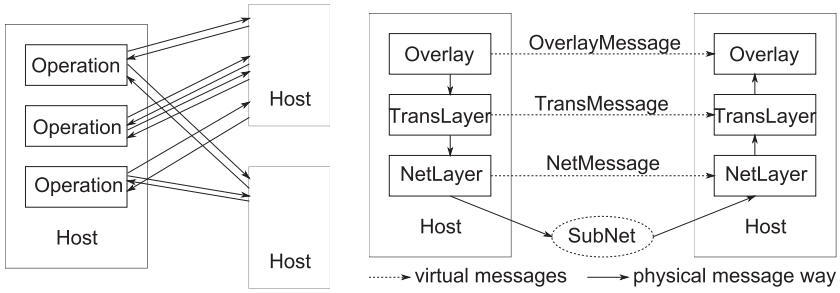
The workflow and simulation control within the simulator is easily configurable with a configuration and an action file. An XML-based configuration file is used to specify that implementations and configurations on which layers are to be used. The configuration file further specifies an action file, which contains the operations that are to be started by specific peers at specific time intervals. These actions refer to commands at the specific layers and can be setup for single of periodic events in given time frames. Once the configuration is completed, the user may start an optional GUI to choose the configuration file and to observe the simulation status (Section 4.2). During the simulation, the monitoring is responsible for tracking the information necessary to generate the simulation statistics and visualizations in the end (Section 4.3).

### *4.1. Simulation engine*

We elaborate the details of the simulation for a deeper understanding. PeerfactSim.KOM is an event-based simulator. Everything that happens during the execution is ordered into a queue of events and handled one after the other. If new events were created during execution, they would be arranged into the correct position of the queue (cf. Figure 2(b)). The management and execution of the simulation and the queue is handled by the simulation engine in its own scheduler. A `SimulationEvent` consists of the event data, a defined event handler, and the simulation time when it should be executed. Event scheduling is carried out by keeping an ordered list of future events waiting to happen. The event queue has to ensure that events can be fetched in the correct order. They are delivered to the defined `SimulationEventHandler` that can be an arbitrary object of the simulator implementing the `eventOccured()` method. New events can be added to the schedule by the configuration through an actions file (cf. Section 4.2) or by other operations within a simulation. There are several events that can be placed into the queue: on the one hand, events required for the actual simulation of the protocols and, on the other hand, administrative events. In the first group, operation executions and message-received events can be found as the most important events. The second group contains different tasks like starting and stopping the simulator or the monitor and also events for the underlying subnet and the churn model. By handling everything in the simulator as an event, the correct execution can be guaranteed. Every action in an overlay network or an application is simulated in PeerfactSim.KOM as an operation. In this object, which is saved at the executing host, data are held from beginning until the end of a process (cf. Figure 3(a)). Examples for operations in an overlay are the join process as `JoinOperation` and the lookup process as `LookupOperation`. Maintaining the state of each operation in the initiating host allows to trace the status, success, and failure of individual operations and thus to initiate further events like retransmissions in case of failures or next protocol steps upon the success of an operation. In all cases, the

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe




(a) The operations concept (b) The messages concept

Figure 3. Operations and messages during the simulation.

status of operations is also reported to the monitoring analyzers, which themselves keep track of relevant metrics for the simulations.

Also messages exist in the simulator to exchange information between the same layer on different hosts (cf. Figure 3(b)). Every message implements the `Message` interface. The important message for our purpose is the `OverlayMessage`. It can be sent by using the transport layer. While processing and sending the message, it will be included in a `TransMessage` and in a `NetMessage`. Similarly, the message will arrive at the other host in the transport layer. For processing, each node has a listener implemented that processes the incoming messages. In addition to the operations and messages, there are callbacks. Because the simulation is event based, there is no way back to the caller of an action. Therefore, the concept of callbacks exists on each level.

### *4.2. Configuration*

In this section, we describe the configuration of comparative evaluations with PeerfactSim.KOM. We introduce the user-side handling with configuring the evaluation and obtaining the results. Each layer offers a set of protocols, which themselves can be configured in various ways. In order to configure an experiment setup, an XML-based configuration file is used. In this, all layers and protocols to be used in the simulator can be activated and combined in the setup of peer sets. Each peer set can range from one peer to thousands of peers; individual protocol-wise actions can be defined in a separate action file. Besides the activation of the individual protocols, their configuration is also defined. In addition to the static configuration in the XML and the actions file, parameters can be used during the simulation for dynamic changes of the properties.

To obtain a better understanding of this elementary configuration, we present a short example configuration (cf. Listing 1). In the default section, different variables and their default values are declared. They can be used in the configuration and are configured while starting the simulation. Then the main class of the simulator is defined. By also defining the core element of the simulator, PeerfactSim.KOM can be seen as a framework to implement multiple simulators. Afterwards, the different layers are defined: a network layer, a transport layer, an overlay, and an application with different parameters (like the distribution of the lookups). Also the monitor class is defined together with an oracle that allows the analyzers to access the whole configuration. The configuration file allows to define specific implementations for the possible layers and to build custom host setups with this, for example, 5,000 nodes with the configuration (NetLayer: GNP, DHT: Chord, Application: File Sharing) and 1,000 nodes with another configuration (NetLayer: GNP, DHT: MaliciousChord, Application: DHTLookupGenerator). Thus, it allows to have nodes with different protocols in the same network.

At the end, the XML include mechanism is used to include two further configurations. This basic XML functionality can be used to build the configuration by using different modules that can be reused in several configurations. The simulator comes with include files for nearly all implemented components of the simulator. Here, we have on the one hand a host builder that defines the composition and locality of the hosts (cf. Listing 2) based on the previously deifined layers. On the other hand, we have the definition of the scenario that describes the simulation schedule (cf. Listing 3).

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration xmlns:xi="http://www.w3.org/2001/XInclude">
  <Default>
    <Variable name="seed" value="0" />
    <Variable name="size" value="1000" />
    <Variable name="startTime" value="0m" />
    <Variable name="finishTime" value="300m" />
  </Default>
  <SimulatorCore class="org.peerfact.impl.simengine.Simulator"
    static="getInstance" seed="$seed" finishAt="$finishTime" />
  <NetLayer class="org.peerfact.impl.network.simple.SimpleNetFactory">
    <LatencyModel class="org.peerfact.impl.network.simple. SimpleStaticLatencyModel" />
  </NetLayer>
  <TransLayer class="org.peerfact.impl.transport.DefaultTransLayerFactory" />
  <Overlay class="org.peerfact.impl.overlay.dht.chord.chord. components.ChordNodeFactory"
      port="123">
  </Overlay>
  <Application
    class="org.peerfact.impl.application.dhtlookupgenerator. DHTLookupGeneratorFactory">
    <Distribution class="org.peerfact.impl.util.stats.distributions. UniformDistribution"
        min="30" max="120" />
  </Application>
  <Monitor class="org.peerfact.impl.common.DefaultMonitor" start="$startTime" stop="$finishTime">
    <Analyzer class="org.peerfact.impl.analyzer.DefaultNetAnalyzer" />
  </Monitor>
  <Oracle class="org.peerfact.impl.util.oracle.GlobalOracle" />
  <xi:include href="includes/HostBuilder.xml" />
  <xi:include href="includes/Scenario.xml" />
</Configuration>
```

Listing 1: A sample configuration file for the simulator

```
<?xml version="1.0" encoding="utf-8"?>
<HostBuilder class="org.peerfact.impl.scenario.DefaultHostBuilder" experimentSize="$size">
  <Group id="all" groupID="World" size="$size">
    <NetLayer />
    <TransLayer />
    <Overlay />
    <Application />
  </Group>
</HostBuilder>
```

Listing 2: A sample include file for the host builder

```
<?xml version="1.0" encoding="utf-8"?>
<Scenario class="org.peerfact.impl.scenario.CSVScenarioFactory"
  actionsFile="config/includes/actions/actions.dat"
  componentClass="org.peerfact.impl.application.dhtlookupgenerator. DHTLookupGenerator"
  additionalClasses="org.peerfact.api.overlay.JoinLeaveOverlayNode">
</Scenario>
```

Listing 3: A sample include file for the scenario

```
all 1s-120m JoinLeaveOverlayNode:join callback
all 120m startLookups
```

Listing 4: A sample actions file for the DHT lookup generator

In most cases, also here, an additional actions file is used to define the actions; it defines that in the first 120 min, all hosts join the network. Afterwards, all hosts start random lookups with the defined generator.

The simulator comes with different runners: there exist different variants of a `GUIRunner` with visual output and a `SimulatorRunner` for use in a console to start simulations. A single simulation can be started with them by using a configuration file and variable values as parameters. Starting the simulator by using a GUI is the most practical way for a first success. For this purpose, two start scripts come with the simulator (`runGui.bat` and `runGui.sh`). Alternatively, the main class `GUIRunner` can be directly used. A configuration file to use in the simulation, a seed for initializing the random number generator, and the values of the variables depending on the configuration file

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

can be set in a graphical user interface. With the button 'Start Simulation,' the simulation will begin and the simulation progress can be seen in a new window. After the simulation finishes, a visualization window opens if it is configured (cf. Section 4.3). Here, the simulated network topology, message transfers, and node details with different properties and measures are visualized. The visualization can be saved for later use. Furthermore, the generated output of the configured analyzers is saved in the `outputs` directory (Section 4.3).

For use on machines without a graphical user interface, for example dedicated simulation servers, the simulator can be started in a console mode. To start a simulation, the scripts `run.bat` and `run.sh` can be executed with the relative configuration file path as the first parameter. Further parameters can be used for simulation parameters. They follow separated with blanks and have the format `<name>=<value>`. Also here, the main class `SimulatorRunner` can be executed directly. The same simulations are conducted as when using the GUI version.

As the third variant, a complete set of simulations and configurations can be started together with one call of the `BatchRunner`. This enables us to run easily comparative evaluations of p2p protocols. It is set on top of the `SimulatorRunner` and reuses it for the single simulations. In contrast, here we allow to give the variables multiple values. Together with a configuration file and a number of seeds, the runner creates the combination of all variable combinations together with the number of seeds. The `BatchRunner` allows two different modes: on the one hand, a script can be generated that includes commands to start all needed simulations and also the merging process. This script can be used on a cluster, on separate machines, or the single commands can be exported to different formats. So it is very flexible, independent of the underlying system architecture for big evaluations. On the other hand, the `BatchRunner` offers the possibility for the whole simulation in one call. Instead of writing a script with commands, all simulations are executed directly by the runner. Through a further parameter, the number of parallel executions in independent processes can be defined. If a simulation fails, the runner automatically restarts this single failed simulation. In the end, it automatically calls the merging process.

To start a batch simulation, the scripts `runBatch.bat` and `runBatch.sh` or the main class `BatchRunner` can be used. The first parameter is the number of parallel executions for the second mode. If it is set to 0, only the execution scripts are created. With a second parameter, the number of seeds used for each configuration is defined. It should be at least 10 for acceptable statistical results. The third parameter is the relative configuration file path as in the console runner before. Further parameters can be used for simulation parameters. In contrast to the console mode, the parameters can have multiple value. Therefore, they have to be specified in the following format: `<name>=<value1>,<value2>,<value3>`.

Once the simulation is set up and started, analyzers are hooked in the code to grasp relevant events, for example the sending or receiving of a message or the initiation of a lookup. These events are interpreted and analyzed in dedicated monitoring packages, which exist for every protocol category (Section 4.3). As for structured overlays, for example, the set of interesting metrics is the same, their relevant information is analyzed in common classes that the individual protocols inherit from. Thus, for all protocols of the same category, objective and comparative evaluations are performed. The analysis of the protocol behavior is printed in comma-separated value files, and corresponding gnuplot files are generated with every simulation. Thus, the results of the simulation are directly presentable after the simulation ends.

The merging process completes the automatic evaluation. After all simulations have finished, it merges the output data and creates the single run, multiple-seed run, and comparative graphics. At the end, a hierarchy is created in the output folder that contains all important files in three different layers: (1) comparative evaluation: plot scripts and generated graphics for comparison with all parameter combinations; (2) multiple-seed evaluation: data files, plot scripts, and generated graphics for one parameter combination with all merged seeds; (3) single evaluation: data files, plot scripts, and generated graphics for one configuration with one seed.

By having the data files, the gnuplot scripts and the graphics in the folders, the user can modify them easily for different purposes, for example for publishing in a paper. The resulting data files and also the plots depend on the used analyzers, which we will discuss in the next section.

Copyright © 2015 John Wiley & Sons, Ltd.

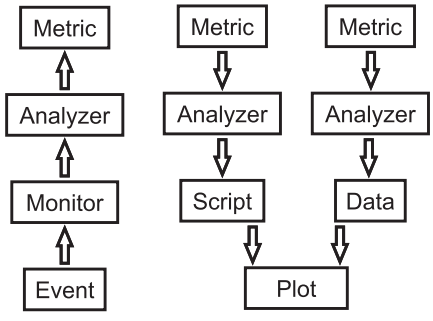
DOI: 10.1002/cpe

### *4.3. Monitoring*

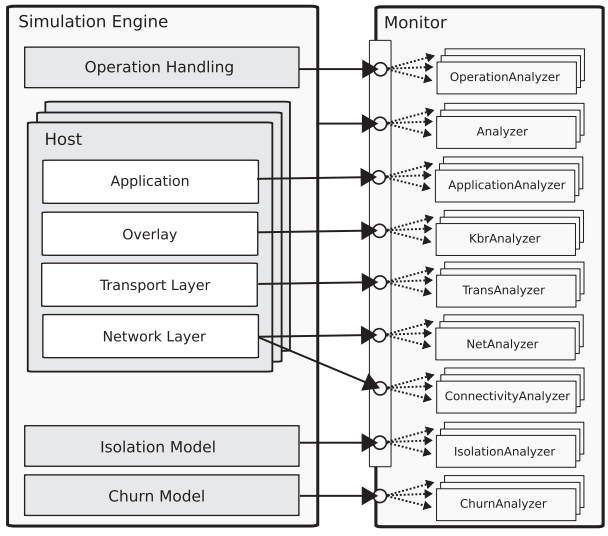
After the simulation ends, the simulation results are saved in the outputs folder. To have an automatic process in this part of the evaluation, an intelligent monitoring process was developed. During the simulation (cf. Figure 4(a)), different events (e.g., the sending of a message) are called in the monitor of the simulation. This monitor forwards the event to all registered analyzers of the event type. A `DefaultMonitor` is integrated in the simulator that serves the common analyzers presented in this section. It can be easily extended for special evaluation purposes. By using the XML configuration, (cf. Section 4.2) different analyzers for the same events can be included. The analyzers evaluate the event and save the needed information in different metrics. The metrics store the overall values and, additionally, the values grouped by the peers. The metrics are responsible for the aggregation of the collected data. Different metrics are included, for example a `CounterMetric` that only counts events, a `SumMetric` to sum up over values or a `StatisticMetric` that calculates different statistical values (like average, median, deviation, and quantils). After the simulation or during output steps (cf. Figure 4(b)), the data of the metrics are passed to data files through the analyzers. Furthermore, the analyzers produce gnuplot scripts fitting exactly to the metrics. In the end, if the gnuplot binary folder is defined in the path variable of the operating system, the plots are produced with the gnuplot scripts and data files. As output, three different types of results are generated (for the data files as well as for the scripts and graphics): (1) output of all peers ordered by the time; (2) output grouped by the single peers; and (3) output grouped by the single peers and sorted by their values.

Each analyzer produces these three types of data and script files once and, depending on the included metrics, different plot outputs. All plots are produced as Portable Network Graphics (PNG) file and as monochrome PDF for the direct include in publications. By having the data files, the gnuplot scripts and the graphics in the folders, the user can modify them easily for different purposes, for example for publishing in a paper with individual formatting. The resulting data files and plots depend on the used analyzers.

The main reason for using a simulator is to generate statistics on the reviewed aspects. Therefore, PeerfactSim.KOM has the possibility to collect data and calculate results. In analogy to the layered structure of the simulator, there exist different analyzer types (cf. Figure 4(c)). For the overall monitoring of the simulation, an analyzer can be defined, which calculates statistics independent of the current simulation events (e.g., it counts the available hosts at different times). The operation handling, especially the start and end of the operations, can be observed by dif-




(a) During simulation (b) After simulation




(c) Overview about the analyzers

Figure 4. The monitoring process.

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

ferent `OperationAnalyzers`. The overlay, transport, and network layer can be monitored by a `DHTOverlayAnalyzer` or a `KBROverlayAnalyzer` or an `UnstructuredOverlayAnalyzer` (message forwarding and query monitoring), `TransAnalyzer` (sending and receiving of messages), `NetAnalyzer` (sending, receiving, and dropping of messages), or `ConnectivityAnalyzer` (online, offline, and isolation events of hosts). As last types, the `ChurnAnalyzer` monitors the modeled churning and the `IsolationAnalyzer` the modeled isolation behavior.

We have implemented and included default implementations that evaluate the most interesting aspects of p2p systems in the simulator. Further, more specific analyzers can be added (e.g., the file-sharing analyzer for the presented application). To use analyzers in the simulation, the monitor part with analyzers must be defined in the configuration. The `DefaultMonitor` is the basis of the whole monitoring and analysis process, every configured analyzer is registered to it. The monitoring is started and stopped by specific events in the simulation, their time is configured by the start and end attributes in the setup. The different types of analyzers are represented as interfaces in the implementation. All analyzers inherit from the analyzer that only provides the start and stop handling. These two are called by the monitor at the beginning and end of the monitoring. The analyzer can do its own initialization (e.g., preparing of files as in the file-based analyzers). Most of the analyzers themselves create events in the simulation queue to do different jobs periodically (e.g., write statistics to a file). Some analyzers only use these periodic tasks to generate data with help of an oracle that has general knowledge about all hosts.

All analyzer implementations that inherit from the `AbstractFileMetricAnalyzer` use the presented metric concept for collecting data. The analyzer itself only adds data to the different used metrics, for example a `CounterMetric` or a `StatisticMetric`. The base analyzer is responsible for the analysis and output of the statistics. A special analyzer exists in the `VisAnalyzer`. It implements four interfaces (`ConnectivityAnalyzer`, `NetAnalyzer`, `TransAnalyzer`, and `OperationAnalyzer`) and collects the data like all other analyzers. The difference is the usage of the data. Instead of generating statistics in files, all data is held in memory and presented in the visualization window after the simulation has finished. It gives a first overview of the behavior of the configured scenario. From here, it can be saved as a visualization file and also be used in a presentation to show the course of events in the network. Because of the architecture of the monitoring component, each analyzer can be easily extended by adding a further metric to measure another interesting property. All outputs are also automatically extended. Next to the new values in the data files, we directly receive new plot scripts and the created graphs.

To summarize, we defined and implemented a complete automatic evaluation process for PeerfactSim.KOM. By only defining a configuration file and the varying variables, the simulator produces a very complex and exhaustive output with all interesting measurements in the end that can be directly reused. An example subset of this output is presented in the next section.

## 5. SAMPLE EVALUATIONS

In this section, we look at some sample evaluations and use the output of the simulations to compare different overlays. Next to presenting the functionality and use of this simulator, we gain different interesting insights about the implemented overlays. We start with the comparison of unstructured and structured overlays in a typical file-sharing application in a simple scenario (cf. Section 5.1). Then, we continue with the simple network to compare different properties of the structured overlays (cf. Section 5.2). In the next step, we investigate different Chord variants to compare them in a more complex and realistic network model with integrated churn (cf. Section 5.3). At the end, we look at the used time and memory consumption for the simulations in PeerfactSim.KOM.

Some parameters of the simulation stay the same during all simulations, they are presented first (cf. Table I). All configurations are run 20 times with different seed values to obtain robust statistical results. By using 20 seeds and merging the results directly, which is carried out by the simulator itself, we do not have to worry about individual outliers. Instead, we directly see the common behavior of the simulated protocol and its properties and can compare the tested configurations with each

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

Table I. Basic configuration for the evaluation scenarios.

| Parameter | Value |
|---|---|
| Number of seeds | 20 |
| Number of nodes | 1000 |
| Simulated time | 300 min |

Table II. Configuration for the first evaluation scenario.

| Parameter | Value |
|---|---|
| Network layer | Simple (static delay, no packet loss) |
| Churn model | None |
| Isolation model | None |
| Overlays | Gnutella06, Gia, Chord, EpiChord, ReChord, CaReChord, Pastry, CAN |
| Application | File Sharing |
| File set distribution | Zipf (100 files, reorder every 10 min) |

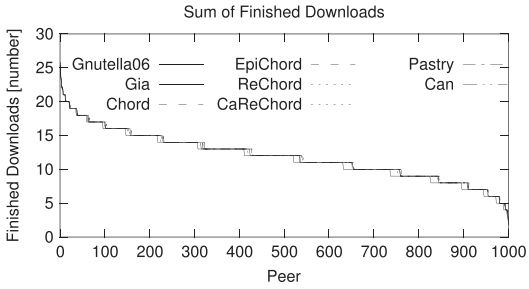



Figure 5. File-sharing application: successful.

other. As network size, 1000 nodes were chosen. It is large enough to see all effects and dependencies in the network and not too large to obtain an acceptable run time for the simulations. The experiments are simulated for 300 min. As we will see later in the single evaluations, this time allows first to stabilize the network and then to simulate enough load to obtain reasonable results.

The comparison with different other simulators is left out here, as we want to investigate on the comparative aspects of different overlays, which is not directly supported by the other simulators. The general comparison of PeerfactSim.KOM, PeerSim [8], and ProtoPeer [23], which was presented in [48], is discussed in Section 2.

### *5.1. Unstructured and structured overlays in comparison*

To compare the overlays of different types, structured as well as unstructured ones, we first use a very simple scenario (cf. Table II). The network layer is based on a simple network model with static delay and no packet loss. There is no churn and no isolation model activated in this scenario. As stated, the network consists of 1,000 nodes; as an application, we use the presented file-sharing application. The overlays simulated are Gnutella06 [38] and Gia [39] as unstructured variants and Chord [15], EpiChord [40], ReChord [41], CaReChord [42], Pastry [35], and CAN [36] as structured DHT-based overlays. Each simulation consists of three phases, starting with a joining phase of 120 min in which all nodes try to join the network uniformly distributed. A publishing phase of 60 min and a lookup phase of 120 min follow. The nodes publish and lookup files from a file set based on a Zipf distribution [47] with 100 files and a reorder interval of 10 min.

After running the simulations, we obtain different comparative plots and also two plots that we want to investigate now: the successful downloads with the different overlays (cf. Figure 5) and the used bandwidths by the different overlays (cf. Figure 6). The successful downloads obtained by

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

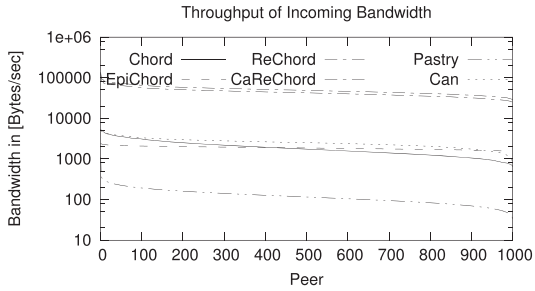



Figure 6. Used bandwidth of different overlays.

Table III. Configuration for the second evaluation scenario.

| Parameter | Value |
|---|---|
| Network layer | Simple (static delay, no packet loss) |
| Churn model | None |
| Isolation model | None |
| Overlays | Chord, EpiChord, ReChord, CaReChord, Pastry, CAN |
| Application | DHTLookupGenerator |
| Lookup distribution | Uniform (between 30 and 120 s) |

each used overlay are very similar, which shows us that all overlays are working correctly and can be used for a file-sharing scenario. The unstructured overlays find as many files as the structured variants. The more interesting point is the bandwidth consumption. Here, we have large differences between the different overlays, which can be seen very well in Figure 6. We find much higher values in the structured variants, especially in the Chord variants. This is reasoned by the fact that these overlays invest much more work in holding the correct typology. It can be seen very well in the case of ReChord and CaReChord that these two overlays also have a self-stabilizing process that consumes further bandwidth.

### *5.2. Distributed hash tables-based overlays in detail*

In the next step, we want to investigate the behavior of all the DHT-based overlays in more detail (cf. Table III). The network layer as well as the churn and the isolation model remain as in the simple scenario. We use the presented lookup generator that periodically starts random lookups with the DHT interface (cf. Section 3.2). Thereby, we reach a stable simulation scenario. Each simulation consists of two phases. First is a joining phase of 120 min, in which time all nodes try to join the network uniformly distributed. A lookup phase of 180 min follows, in which each node looks up random values in uniform distributed intervals between 30 and 120 s. The other properties stay the same. By using the introduced configuration process, only the altering variables have to be changed, the rest of the configuration does not have to be modified or created new. This enables us to run various evaluations in different complex scenarios without spending much time in configuring and preparing the simulations.

The hop count measures the main performance of the overlays (cf. Figure 7). The fewer hops are required, the faster a lookup can be finished. Additionally, the hop count should be nearly equal for all peers. This fact can be seen in the measurements of all plots, only the hop count in CAN deviates for the different peers. With hop count values between 20 and 40 hops, CAN is also the worst tested overlay, which is based on its topology in a two-dimensional coordinate system. It follows Chord, ReChord, and CaReChord with a stable hop count at around six hops. As the three overlays use exactly the same routing process, they also have the same hop count. Pastry has a hop count between three and six hops in this scenario. EpiChord can reach average values between two and three, as it uses a cache and often only needs one hop for a lookup. All given results also reflect the theoretical bounds, but allow much more detailed comparison of the overlays.

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

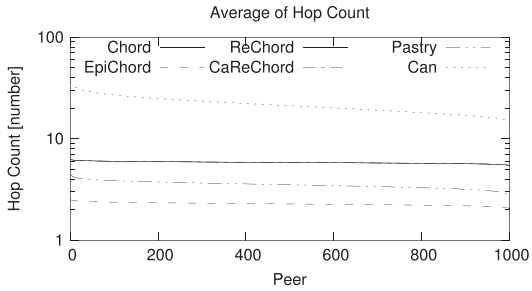



Figure 7. Hop count of different overlays in a simplified model.

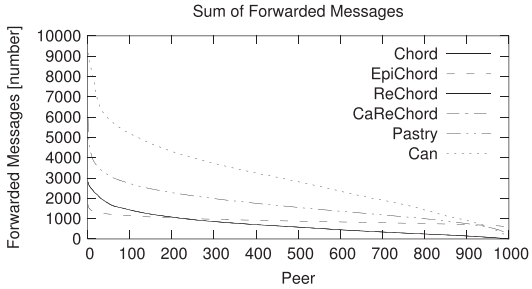



Figure 8. Forwarded lookups by the nodes in different overlays in a simplified model.

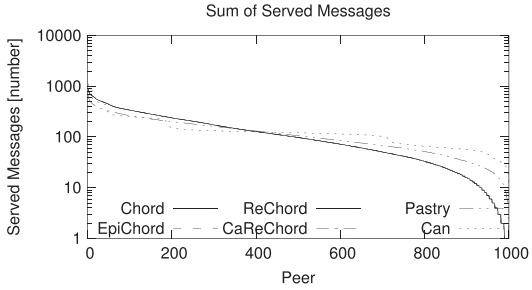



Figure 9. Served lookups by different nodes in different overlays in a simplified model.

By looking at the forwarded and served lookups, we can measure the fairness of the overlay. All peers should have a similar amount of load; the forwarded messages indicate the load in the routing process (cf. Figure 8). While Chord and its variants distribute the routing load fairly among the peers, CAN and Pastry have an imbalance. The served messages indicate how fair the data is distributed among the peers and how many lookups have to be served by them (cf. Figure 9). We see a similar distribution in all overlays. There are always some nodes that have to serve more messages and others that have to serve nearly no messages. This is explained by the random distribution of the lookup targets.

At last, we present the used bandwidths of the nodes (cf. Figure 10). Next to the routing load, we also measure the load to manage the topology. The results are similar as in the first simulation with all overlays. The Chord variants and especially the self-stabilizing ReChord and CaReChord consume clearly more bandwidth than Pastry and CAN. This is due to the use of virtual nodes, each having a set of contacts. With the help of the presented plots, we can evaluate the overlays in all interesting properties. Next to the efficiency, the costs of the routing also have been considered.

### *5.3. Different chord variants under churn*

Last, we investigate the four different Chord variants in a more realistic scenario (cf. Table IV). The network layer is now replaced by a modular network layer. This offers a realistic scenario

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

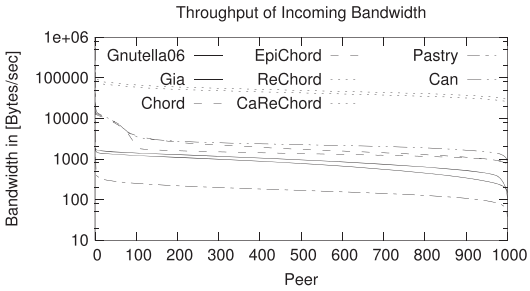



Figure 10. Used bandwidth of different overlays in a simplified model.

Table IV. Configuration for the third scenario.

| Parameter | Value |
|---|---|
| Network layer | Modular |
| Churn model | KAD |
| Isolation model | None |
| Overlays | Chord, EpiChord, ReChord, CaReChord |
| Application | DHTLookupGenerator |
| Lookup distribution | Uniform (between 30 and 120 s) |

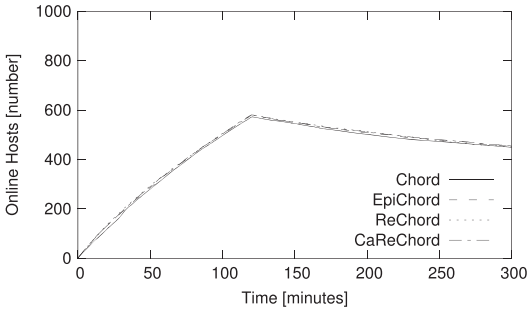



Figure 11. Online hosts.

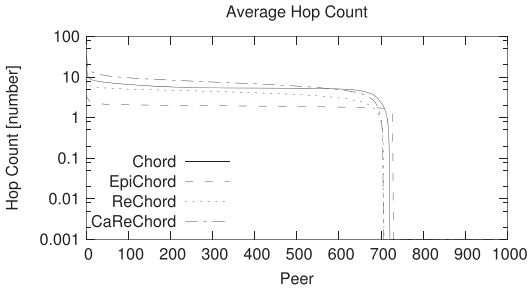



Figure 12. Average hop count of different chord variants with the modular network layer and churn.

with measurement-based delay, packet loss, jitter, and bandwidth. Furthermore, we activate churn in this scenario. The churn is based on a KAD model, a Weibull distribution [32] based on original measurements by Steiner *et al*. [30] in the eDonkey network. Herewith, dynamic systems with node and data failures can be evaluated. The number of nodes in this scenario with churn is depicted in Figure 11. The simulator allows the configuration of nearly each possible scenario, but because of space limitations, we can only show a subset of evaluations.

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

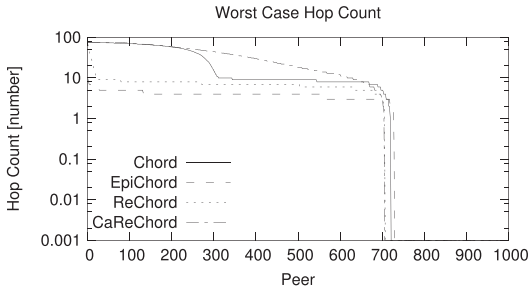



Figure 13. Worst case hop count of chord variants with the modular network layer and churn.

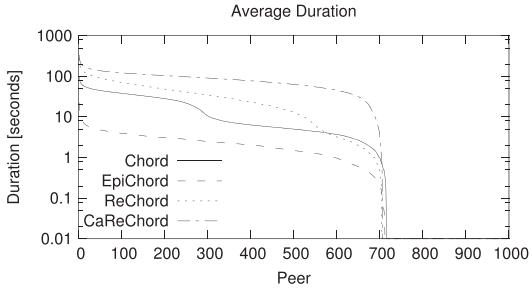



Figure 14. Average lookup of chord variants with the modular network layer and churn.

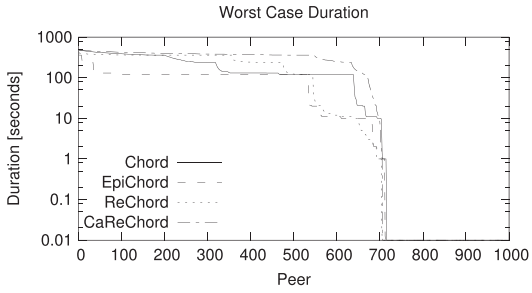



Figure 15. Worst case lookup duration of chord variants with the modular network layer and churn.

To see the performance of the Chord adaptions, we look at the hop count in average (cf. Figure 12) and in the worst case (cf. Figure 13). In addition, we look at the lookup times in average (cf. Figure 14) and in the worst case (cf. Figure 15). Furthermore, we look at the exact bandwidth consumption (cf. Figure 16) and the number of messages (cf. Figure 17) to see the costs of these overlays. While the hop count is a good indicator for the performance of the routing process, the lookup duration directly gives us performance values in a realistic scenario. The hop counts of the four Chord overlays (cf. Figure 12) are comparable with the values from the previous measurements. They are a little higher, which is reasoned by missing nodes because of churn and retransmissions because of packet loss. While looking at the worst-case values for the hop count (cf. Figure 13), we see significant high values in the default Chord overlay and the CaReChord implementation. Both overlays have very long routes in individual cases. EpiChord and ReChord do not have such big problems with the unstable network and the worst-case hop count is only slightly larger than the average value. The more interesting aspect is the lookup duration (cf. Figure 14). The values reflect the same trend as the number of hops. But here, we see the influence of the self-stabilizing overhead. ReChord and CaReChord have more work to do, and therefore, the lookup duration is longer. In the worst cases (cf. Figure 15), all overlays need more time for the lookups (about ten times compared with the average value) with the same tendency (self-stabilizing overlays need more time).

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

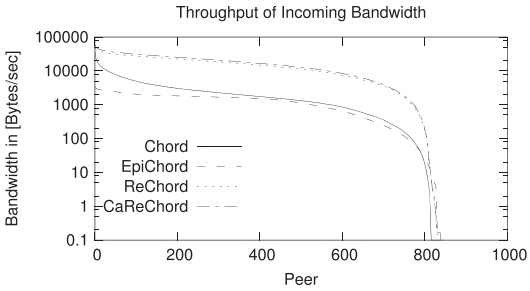



Figure 16. Bandwidth of different chord variants with the modular network layer and churn.

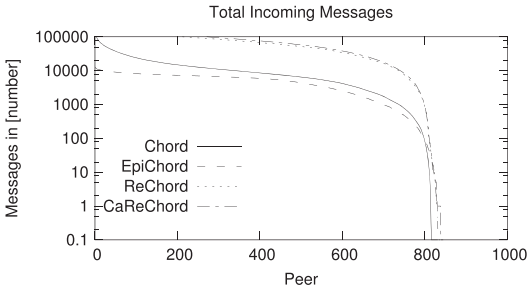



Figure 17. Bandwidth of different chord variants with the modular network layer and churn.

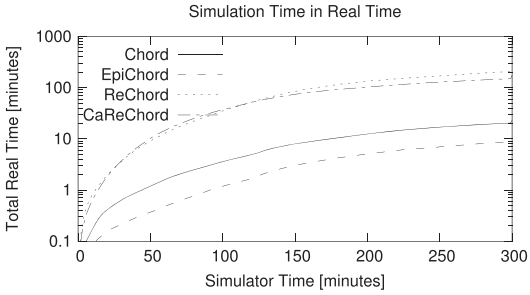



Figure 18. Time needed by the simulations.

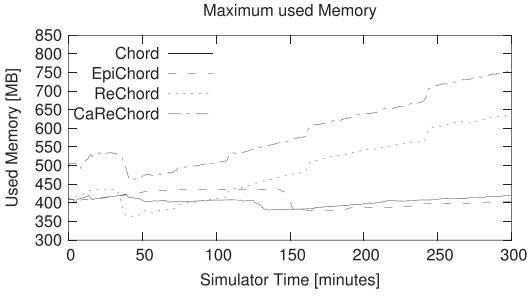



Figure 19. Memory used by the simulations.

To see the reasons, we also look at the bandwidth consumption (cf. Figure 16). It is much higher for the two self-stabilizing networks in comparison with Chord and EpiChord. The same result is reflected by the number of messages (cf. Figure 17) used by the four chord variants.

As a final evaluation, we present the performance of the simulator itself. Therefore, we consider the last experiment and look on the one hand at the simulation duration (cf. Figure 18) and, on the other hand, at the used memory (cf. Figure 19). We see a coherence to the bandwidth consumption.

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

If an overlay uses more messages, the used memory and the total duration of the simulation also increases. This effect is caused by the properties of the event-based simulation engine. The processing of every message requires time and space. In pretests, the needed memory and time can be measured and then the real simulations can be planned and started in the needed environments. For example, time and memory can be allocated in compute centers for an efficient parallel execution. Especially for large sets of evaluations, this is an important advantage. In [48], it is already shown that the time and memory consumption of PeerfactSim.KOM is linear in the number of nodes and that the simulator has no problems with networks of up to 100,000 nodes.

We have seen different comparative plots for different selected metrics. All interesting properties of the overlays can be compared directly. The properties of the simulation enable us to conduct a complete comparative evaluation of p2p systems in all scenarios. Because of the large set of integrated events, analyzers, and metrics in the simulator, almost all important properties of all layers are analyzed, and we can choose the interesting outputs for further processing. We can decide after the runs what we want to compare and we do not have to restart simulations if we want to investigate other properties. Using many seeds for all configurations, we obtain representative results of the evaluations, which let us make reliable statements about the protocols and their properties.

While in the previous discussions, we presented a look on the features and characteristics of PeerfactSim.KOM; next, we discuss its performance in comparison with other simulators that aim at the simulation of the same kind of networks.

## 6. COMPARISON WITH OTHER SIMULATORS

Simulators aim to reproduce the reality and to estimate the behavior of large-scale networks that would otherwise be difficult to setup and to evaluate. A comparison of simulators can be either based on the results and the quality of the estimations or on their performance in terms of simulation time and memory consumption. While a comparison in terms of simulation time and memory consumption is achievable, a comparison in terms of equality of simulation results is hard to achieve because of the impact of the models and implementation approaches of the various simulators.

Considering the consistency of the simulation results of, for example, a single overlay protocol tested on various simulators is difficult to balance. As most simulators come with a specific layering in terms of simulation core, network model, and overlay protocol, the quality of the models and implementations take influence on the obtained results of for example the simulated overlay protocol. The specific network layer, for example, defines the delay of single hops in the overlay and thus the overall lookup latency. The overlay protocol itself allows typically a set of configuration options, for example, for Chord, the number of fingers, fix finger intervals, or stabilizing intervals. The selection of these parameters takes influence on the hop count and success ratio of the lookup procedure. In [49], Baumgart and Heep analyzed the impact of various parameter settings in Chord and other DHTs on the lookup latency and identified large differences in the resulting quality. For example, modifying the fix finger interval from 30 to 480 s increases, the average lookup delays from 320 to 640 ms in a network with 10,000 nodes. Parameter changes within one single overlay induce a big influence on the results, further influences coming from the characteristics of the network layer, the transmission layer, and the simulation core will further take influence on the results.

Thus, for a comparison of the simulator characteristics, it is necessary to simulate exactly the same code, that is, overlay protocol. With the same code to simulate on the overlay layer in combination with a simple network model, a comparison of the lower layers, especially the simulator core, is possible. Exactly, this evaluation of PeerfactSim.KOM has been given by our co-authors of PeerfactSim.KOM in [48]. In this paper, our co-authors of previous PeerfactSim.KOM papers [7, 11] have created a common interface for overlay network simulators, which allows to match simulated applications and overlay protocols to this interface. This interface was implemented for PeerfactSim.KOM, PeerSim [8], and ProtoPeer [23]. The three simulators have been comparatively evaluated by implementing a KBR [34] compliant overlay that was run through an decorator pattern on all three simulators using this Common Simulator Interface.

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

PeerSim [8] provides both a cycle-based and an event-based simulation approach, allowing it for the researcher to choose between high-scaling simulation with low network details and concurrent execution of events and mid-scaling simulations with the option of high-detail network models and strict event scheduling. PeerSim is rather popular in the literature and has been chosen therefore as a reference. ProtoPeer [23] on the other hand aims at allowing both the simulation of overlay protocols as well as their live network deployment. The underlay network component can be easily exchanged between a simulated and real network. PeerfactSim.KOM provides with nearly 90,000 lines of code much more functionality than both other simulators with approximately 7,500 lines of code each, this is due to the integration of various protocols on various layers within PeerfactSim.KOM, while the other simulators mainly focus on the core and support the usage of 'external' modules. The interface implementation is lightweight on all simulators with 225–237 lines of code.

As a basis of comparison and as the selected protocol to run all all three simulators, a simple random walk protocol was applied. This random walk protocol uses the Common Simulator Interface, which is the core of the paper [48]. This interface is implemented in all three simulators and allows to run the random walk (source code) on the three simulators and thus to compare the performance of these simulators. In the simulations described in [48], periodically every node initiates a random walk that spreads the information on this node to the routing tables of the receiving/forwarding nodes. In the comparison of the protocol's performance, the simulation time as well as the memory consumption were investigated. The performance of the random walk protocol resulted in the same behavior on all simulators. The simulations were run 5 times each and scaled from 10, 100, 1000, 10,000 to 100,000 nodes. During this scaling, the memory consumption, Figure 20, and simulation time, see Figure 21, was recorded for all simulators. In addition, a pure implementation of the random walk was tested on PeerfactSim.KOM to capture the overhead of the Common Simulator Interface. With scaling the number of nodes, the memory consumption and the simulation runtime

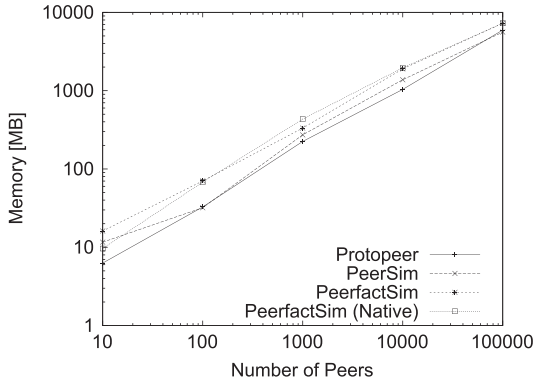



Figure 20. Memory used by PeerSim, ProtoPeer, and PeerfactSim.KOM. Source: [48].

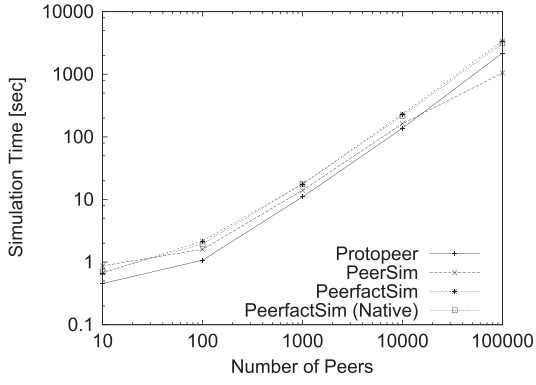



Figure 21. Time needed by PeerSim, ProtoPeer, and PeerfactSim.KOM. Source: [48].

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

of all simulators were behaving accordingly. Although PeerfactSim.KOM consumes more memory and needs more time to finish the simulation in comparison with PeerSim and ProtoPeer, the difference is small. This overhead mainly comes from the layering in PeerfactSim.KOM that aims to support not only overlay protocols but also further higher and lower layer protocols, such as monitoring [50] or low-layer message prioritization [51]. These additional layers, even if using dummies, still consume computing time and add to the memory consumption. Nevertheless, the simulators behave equally and show consistent results.

While this comparison in [48] introduces an approach to compare the performance of simulators, eventually, various factors take influence on the memory and time consumption. In specific, the main influence comes from the number of events in the simulator that need to be processed, the quality and number of network models as well as implementation characteristics. Thus, a simulation with a detailed network model, with a sophisticated workload generator in the application and details on how the stabilizing mechanisms, churn, and message processing is implemented, takes a great influence on the performance of the simulator. With the evaluation in [48], an impression of the simulator performances is given.

## 7. CONCLUSIONS

In this article, we presented a concept for the systematic evaluation of p2p systems with the p2p systems simulator PeerfactSim.KOM. The simulator provides different layers that are all well specified by an application interface, and therefore, an exchange of single components is supported directly by the architecture. The simulator already comes with a large range of different implementations in each layer and a complete monitoring component that has analyzers for the most interesting properties of p2p systems. Our approach for systematic comparative evaluation incorporates the holistic setup of all overlays to be tested, all parameters to vary during the simulations, and the overall number of random seeds to use per experiment through a simple XML file. By configuring all experiment setups to be simulated, the user of the simulator can initiate all simulation runs in one instance. The single runs can be distributed on different machines to allow a parallel and significantly faster execution. Once all simulations are run, the results are merged on the level of individual simulation runs, across several simulation runs of one setup with several seeds, as well as across parameter or overlay variations. The results present an exhaustive comparison of the configured overlays and protocols in full details, both in comma-separated value files as well as automatically generated and plotted gnuplot graphs. The evaluation shows a comparison of both structured and unstructured p2p overlays at the use case of file-sharing as well as lookup resolving for structured p2p overlays. The proposed and integrated extensions for PeerfactSim.KOM enable the user to classify and extensively compare different p2p overlays and also their parameters configurations in different scenarios.

In the future, we plan to extend the support for simulated layers to more functional layers in parallel and the support of opportunistic, mobile networking in the networking layer. Further, we aim the investigation of multi-layer interdependencies and the effect of and solution to malicious behavior in various layers. We develop new mechanisms handling these issues and evaluate them with the possibilities of the simulator.

ACKNOWLEDGEMENTS

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre ‘On-the-fly Computing’ (SFB 901) and the DFG Research Grant ‘OverlayMeter’ (GR 4498/1-1).

REFERENCES

1. Skype. *Skype – Peer-to-peer internet telephony*, 2004. (Available from: http://www.skype.com) [Accessed on 1 December 2015].
2. Cutillo LA, Molva R, Strufe T. Safebook: Feasibility of transitive cooperation for privacy on a decentralized social network. In *IEEE WOWMOM ’09: Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, Kos Island, Greece, 2009; 1–6.

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

3. Graffi K, Groß C, Mukherjee P, Kovacevic A, Steinmetz R. LifeSocial.KOM: A P2P-based platform for secure online social networks. In *IEEE P2P '10: Proceedings of the International Conference on Peer-to-peer Computing*: Delft, The Netherlands, 2010; 1–2.
4. Graffi K, Groß C, Stingl D, Hartung D, Kovacevic A, Steinmetz R. LifeSocial.KOM: a secure and P2P-based solution for online social networks. In *IEEE CCNC '11: Proceedings of the IEEE Consumer Communications and Networking Conference*: Las Vegas, NV, USA, 2011; 554–558.
5. Graffi K, Mukherjee P, Menges B, Hartung D, Kovacevic A, Steinmetz R. Practical security in P2P-based social networks. In *IEEE LCN '09: Proceedings of the Annual Conference on Local Computer Networks*, 2009.
6. Graffi K, Podrajanski S, Mukherjee P, Kovacevic A, Steinmetz R. A distributed platform for multimedia communities. In *IEEE ISM '08: Proceedings of the International Symposium on Multimedia*, 2008.
7. Kovacevic A, Kaune S, Heckel H, Mink A, Graffi K, Heckmann O, Steinmetz R. PeerfactSim.KOM – a simulator for large-scale peer-to-peer networks. *Technical Report Tr-2006-06*, Technische Universität: Darmstadt, Germany, 2006.
8. Montresor A, Jelasity M. PeerSim: A scalable P2P simulator. In *IEEE P2P'09: Proceedings of the International Conference on Peer-to-peer Computing*: Seattle, Washington, USA, 2009; 99–100.
9. Baumgart I, Heep B, Krause S. OverSim: A scalable and flexible overlay framework for simulation and real network applications. In *IEEE P2P '09: Proceedings of the International Conference on Peer-to-peer Computing*: Seattle, Washington, USA, 2009; 87–88.
10. Graffi K. PeerfactSim.KOM: A P2P System simulator – experiences and lessons learned. In *IEEE P2P '11: Proceedings of the International Conference on Peer-to-peer Computing*, Kyoto, Japan, 2011; 154–155.
11. Kovacevic A, Kaune S, Mukherjee P, Liebau N, Steinmetz R. Benchmarking platform for peer-to-peer systems. *IT – Information Technology* 2007; **49**(5):312–319.
12. Stingl D, Groß C, Rückert J, Nobach L, Kovacevic A, Steinmetz R. PeerfactSim.KOM: A simulation framework for peer-to-peer systems. In *IEEE HPCS'11: Proceedings of the International Conference on High Performance Computing and Simulation*, Istanbul, Turkey, 2011; 577–584.
13. Feldotto M, Graffi K. In *IEEE HPCS'13: Proceedings of the International Conference on High Performance Computing and Simulation*. IEEE: Los Alamitos, CA, USA, 2013; 99–106.
14. Williams T, Kelley C. *gnuplot 4.6: An Interactive Plotting Program*, 2012. (Available from: http://www.gnuplot.info) [Accessed on 1 December 2015].
15. Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM: New York, NY, USA, 2001; 149–160. [Accessed on 1 December 2015].
16. PlanetLab. *An Open Platform for Developing, Deploying, and Accessing Planetary-Scale Services,* Princeton, NJ, USA. (Available from: https://www.planet-lab.org/.) [Accessed on 1 December 2015].
17. UC Berkeley and LBL and USC/ISI, and Xerox PARC. *The Network Simulator - ns-2*. Los Angeles, CA, USA, 2014. (Available from: http://www.isi.edu/nsnam/ns/) [Accessed on 1 December 2015].
18. Renard K, Peri C, Clarke J. A performance and scalability evaluation of the ns-3 distributed scheduler. In *Proceedings on the Simutools*. ACM: Brussels, Belgium, 2012; 378–382.
19. Varga A, Hornig R. An overview of the OMNeT++ simulation environment. In *ICST Simutools'08: Proceedings of the International Conference on Simulation Tools and Techniques for Communications*. ICST: Brussels, Belgium, 2008; 60.
20. Naicken S, Livingston B, Basu A, Rodhetbhai S, Wakeman I, Chalmers D. The state of peer-to-peer simulators and simulations. *Computer Communication Review* 2007; **37**(2):95–98.
21. Amoretti M, Agosti M, Zanichelli F. DEUS: A discrete event universal simulator. In *Simutools'09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques for Communications, Networks and Systems*: Rome, Italy, 2009; 58.
22. Shudo K, Tanaka Y, Sekiguchi S. Overlay weaver: an overlay construction toolkit. *Computer Communications* 2008; **31**(2):402–412.
23. Galuba W, Aberer K, Despotovic Z, Kellerer W. ProtoPeer: From simulation to live deployment in one step. In *IEEE P2P'08: Proceedings of the International Conference on Peer-to-peer Computing*: Aachen, Germany, 2008; 191–192.
24. Galuba W, Aberer K, Despotovic Z, Kellerer W. Protopeer: a P2P toolkit bridging the gap between simulation and live deployment. In *Simutools'09: Proceedings of the International Conference on Simulation Tools and Techniques for Communications, Networks and Systems*. ICST: Brussels, Belgium, 2009; 60.
25. Gil TM, Kaashoek F, Li J, Morris R, Stribling J. *P2Psim: A Simulator for Peer-to-peer Protocols*, 2005. (Available from: http://pdos.csail.mit.edu/p2psim/) [Accessed on 1 December 2015].
26. López PG, Pairot C, Mondéjar R, Ahulló JP, Tejedor H, Rallo R. PlanetSim: a new overlay network simulation framework. In *Sem'04: Proceedings of the International Workshop on Software Engineering and Middleware*. Springer: Berlin, Germany, 2004; 123–136.
27. He Q, Ammar MH, Riley GF, Raj H, Fujimoto R. Mapping peer behavior to packet-level details: a framework for packet-level simulation of peer-to-peer systems. In *IEEE MASCOTS'03: Proceedings of the International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. IEEE: Los Alamitos, CA, USA, 2003; 71–78.


Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

28. Yang W, Abu-Ghazaleh NB. GPS: a general peer-to-peer simulator and its use for modeling BitTorrent. In *IEEE MASCOTS'05: Proceedings of the International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. IEEE: Los Alamitos, CA, USA, 2005; 425–434.
29. Schlosser M, Condie T, Kamvar S. Simulating a file-sharing P2P network. *Technical Report* , Stanford InfoLab: Stanford, CA, USA.
30. Steiner M, En-Najjary T, Biersack E. Long term study of peer behavior in the KAD DHT. *IEEE/ACM Transactions on Networking* 2009; **17**:1371–1384.
31. Maymounkov P, Mazières D. Kademlia: A peer-to-peer information system based on the XOR metric. In *IPTPS '02: Proceedings of the International Workshop on Peer-to-peer Systems*, vol. 2429, LNCS. Springer: Berlin, Germany, 2002.
32. Weibull W. Wide applicability. *Journal of Applied Mechanics* 1951; **103**:293–297.
33. Gross C, Lehn M, Münker C, Buchmann AP, Steinmetz R. Towards a comparative performance evaluation of overlays for networked virtual environments. In *Ieee p2p'11: Proceedings of the International Conference on Peer-to-peer Computing*: Kyoto, Japan, 2011; 34–43.
34. Dabek F, Zhao BY, Druschel P, Kubiatowicz J, Stoica I. Towards a common API for structured peer-to-peer overlays. In *Iptps '03: Proceedings of the International Workshop on Peer-to-peer Systems*, vol. 2735, LNCS. Springer: Berlin, Germany, 2003; 33–44.
35. Rowstron AIT, Druschel P. Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM Middleware '01: Proceedings of the International Conference on Distributed Systems Platforms*, vol. 2218, LNCS. Springer: Berlin, Germany, 2001; 329–350.
36. Ratnasamy S, Francis P, Handley M, Karp R, Schenker S. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, Vol. 31. ACM: New York, NY, USA, 2001; 161–172.
37. Clip2 - the gnutella developer forum. *Gnutella*, 2002. (Available from: http://rfc-gnutella.sourceforge.net/developer/stable/index.html) [Accessed on 1 December 2015].
38. Klingberg T, Manfredi R. *Gnutella 0.6*, 2002. (Available from: http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html) [Accessed on 1 December 2015].
39. Chawathe Y, Ratnasamy S, Breslau L, Lanham N, Shenker S. Making gnutella-like P2P systems scalable. In *Sigcomm '03: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM: New York, NY, USA, 2003; 407–418.
40. Leong B, Liskov B, Demaine ED. EpiChord: parallelizing the chord lookup algorithm with reactive routing state management. *Computer Communications* 2006; **29**(9):1243–1259.
41. Kniesburges S, Koutsopoulos A, Scheideler C. Re-chord: a self-stabilizing chord overlay network. In *ACM SPAA '11: Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures*: San Jose, CA, USA, 2011; 235–244.
42. Benter M, Divband M, Kniesburges S, Koutsopoulos A, Graffi K. Ca-Re-Chord: A churn resistant self-stabilizing chord overlay network. In *NetSys'13: Proceedings of the Networked Systems Annual Technical Conference*: Stuttgart, Germany, 2013; 27–34.
43. Wette P, Graffi K. Adding capacity-aware storage indirection to homogeneous distributed hash tables. In *Netsys'13: Proceedings of the Conference on Networked Systems Annual Technical Conference*, Stuttgart, Germany, 2013; 35–42.
44. Hu S-Y, Chen J-F, Chen T-H. VON: a scalable peer-to-peer network for virtual environments. *IEEE Network* 2006; **20**(4):22–31.
45. Schmieg A, Stieler M, Jeckel S, Kabus P, Kemme B, Buchmann A. pSense – maintaining a dynamic localized peer-to-peer structure for position based multicast in games. In *IEEE P2P '08: Proceedings of the International Conference on Peer-to-peer Computing*: Aachen, Germany, 2008; 247–256.
46. Bharambe AR, Agrawal M, Seshan S. Mercury: supporting scalable multi-attribute range queries. *ACM SIGCOMM Computer Communication Review* 2004; **34**(4):353–366.
47. Zipf GK. *Human behaviour and the principle of least effort: an introduction to human ecology*. Addison-Wesley: Oxford, England, 1949.
48. Groß C, Lehn M, Stingl D, Kovacevic A, Buchmann A, Steinmetz R. Towards a common interface for overlay network simulators. In *IEEE ICPADS '10: Proceedings of IEEE International Conference on Parallel and Distributed Systems*: Shanghai, China, 2010; 59–66.
49. Baumgart I, Heep B. Fast but economical: a simulative comparison of structured peer-to-peer systems. In *NGI' 12: Proceedings on the IEEE EURO-NF Conference on Next Generation Internet*: Karlskrona, Sweden, 2012; 87–94.
50. Graffi K, Stingl D, Rueckert J, Kovacevic A, Steinmetz R. Monitoring and management of structured peer-to-peer systems. In *IEEE P2P '09: Proceedings of the International Conference on Peer-to-peer Computing*: Seattle, Washington, USA, 2009; 311–320.
51. Graffi K, Pussep K, Kaune S, Kovacevic A, Liebau N, Steinmetz R. Overlay bandwidth management: scheduling and active queue management of overlay flows. In *IEEE LCN '07: Proceedings of the Annual Conference on Local Computer Networks*, 2007.


Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe