A Survey of Peer-to-Peer Network Simulators

Stephen Naicken, Anirban Basu, Barnaby Livingston and Sethalat Rodhetbhai
Network Lab
Department of Informatics
University of Sussex
{stephenn, a.basu, bjl20, s.rodhetbhai}@sussex.ac.uk

***Abstract*— As scientists, our research must be tested and evaluated in order for it to be shown to be valid. Part of this process includes providing reproducible results so that peers are able to confirm any findings. The techniques used to achieve this include analytical solutions, simulators and experiments with the actual system, however in the area of Peer-to-Peer (P2P) computing, this gives rise to a number of challenges.**

In this paper, we focus on P2P simulators by surveying a number of P2P simulators with respect to important criteria when simulating a P2P system. Simulators are compared, usage experiences are given and limitations are discussed. The survey shows that while there are a plethora of simulators, there are many issues such as poor scalability, little or no documentation, steep learning curves, and system limitations that makes using these simulators a convoluted process.

I. Introduction

The computer science community, as well as all other scientific communities, expects researchers to evaluate and test research developments. Reproducible results must be provided so that peers are able to confirm any findings. To achieve this, analytical solutions, simulations and experiments are the methods that are used [1], however, in the area of Peer-to-Peer (P2P), they give rise to a number of challenges.

The Analytical approach is one where a mathematical model of the system is examined. It can provide an analytical solution, but only if the model is simple [1]. While this approach is being used in P2P research, as it requires a simple model, many of the complexities of real-world P2P systems have to be ignored [2]. One example of the successful use of an analytical approach is [3] where the authors provide a theoretical analysis of BitTorrent [4] under concurrent node joins and failures.

Should applying analytical methods prove too complex, an alternative is to run experiments with the actual system. P2P systems can consist of a large number of nodes [5] and any experiment on even a relatively small scale of a few thousand nodes would be impractical, or even impossible. Should it be feasible to acquire the resources necessary to run such an experiment, there are other impracticalities that must be considered. Applying changes to an experiment such as modification of the protocol running at each node and the topology of the network would be difficult and certainly time consuming if the experiment were performed on a large scale. Introducing malicious nodes into a network for an experiment focused on security issues such as denial of service attacks would be another issue to consider [6]. The overlay network testbed, Planetlab [7], addresses many of the issues with large scale experiments. It consists of 664 nodes over 322 sites that are subject to real-world network conditions. This makes it a particular useful facility to perform large scale experiments of overlay protocols and for the validation of results obtained from simulators.

Simulators are not without their disadvantages and these will be discussed in due course, nevertheless, they do address the impracticalities of mathematical analysis and experimentation. At this point, it is important to emphasise that simulations are not seen as strictly independent from the analytical approach and experimentation. If possible analytical approaches should be utilised and proved using a simulator, similarly simulation results should if possible be validated by experiments with the actual system. It is clear however, that for P2P research often the most practical technique is simulation and it is therefore important that the P2P research community are made aware of the simulation tools available.

The rest of the paper is organised as follows. The criteria used to analyse P2P simulators is given in section II-A, followed by the evaluation methodology and review of P2P simulators with respect to the given criteria in sections II-B and II-C respectively. Section II-D provides a discussion of some of the issues encountered while evaluating the chosen simulators with respect to the evaluation methodology. In Section III, future work is discussed and finally conclusions are given in section IV.

II. Peer-to-Peer Simulators

A. Criteria

The simulators will be assessed according to a number of criteria grouped under the headings Simulator Architecture, Usability, Scalability, Statistics, Underlying Network Simulation and System Limitations.

1) Simulator Architecture: These are criteria relating to the design and functioning of the simulator, what features it includes and how they are implemented. The criteria include: whether it supports structured or unstructured overlay simulation, or both; whether it simulates each individual packet or abstracts away the underlying network partly or completely; whether the simulator works on discrete event simulation basis, that is whether it uses a scheduler which synchronises message transfer between nodes, adding delay as necessary; how it implements remote procedure calls; how the identifiers are chosen; whether it allows for distributed simulation, i.e.

can simulations be run across a number of machines to allow greater scaling or faster simulation runtime?

These criteria also include aspects of how node behaviour is simulated, for example: whether churn can be specified (nodes joining and leaving the DHT); what levels of churn be simulated; whether nodes can be made to temporarily or permanently fail (i.e. not leave cleanly); whether both iterative and recursive routing can be simulated.

2) Usability: These criteria have to do with how easy the simulator is to learn and use, they include: whether the simulator has a clean API which allows protocol code to be easily understood, altered and ported to and from other simulators; how experiment scenarios are created, if there is a script language, how easy it is to learn and how expressive it is; what documentation exists and how easy it is to follow.

3) Scalability: P2P protocols are generally designed to be scalable and are often designed to solve problems of scalability. So one of the most important tests a simulator can provide is how the protocol scales to thousands of nodes or more, especially as this would be a very hard if not impossible experiment to attempt using a real network.

4) Statistics: Another key aspect of a simulator is the results it produces. The results need to be expressive and easy to manipulate in order to carry out statistical analysis and to produce graphs. The initial state of the simulator should be able to be saved so that the simulation can be repeated to ensure that the results are reproducible.

5) Underlying Network Simulation: P2P simulators take a number of different approaches to simulating the underlying network, from simulating every packet to completely abstracting away everything below the overlay layer. The criteria used to assess this aspect of the simulators include: which properties of the network layer can be simulated; whether cross-traffic (other traffic on the network unrelated to the overlay) can be simulated; whether it can simulate differences in link latency; how realistic the underlying topology is.

6) System Limitations: How well does the simulator make use of the computer resources available to it? If it is inefficient in its use of resources then this will reduce its ability to scale.

B. Evaluation Methodology

For each simulator, the latest version was downloaded and compiled from source. Some simulators are packaged with implementations of either Chord-TON [8], Chord-SIGCOMM [9] or Chord-PNS [10], but for those without an implementation, an attempt was made to implement the Chord-TON protocol. If any simulation scenario files were packaged with the simulator, these were executed and results compared to any expected results to ensure that the simulators were functioning correctly. After this, the intention was to recreate the experiments from the Chord-TON paper and compare the results. With some simulators, we attempted to test their scalability by running experiments to find the maximum number of nodes that could join the network and interact successfully. In addition to our usage experiences, we made use of documentation, research papers and source code to evaluate the simulators with respect to our criteria given above.

C. Simulator Review

1) DHTSim: DHTSim [11] is a discrete event simulator for structured overlays, specifically DHTs. It is intended as a basis for teaching the implementation of DHT protocols, and as such it does not include much functionality for extracting statistics. RPC is implemented as discrete event based message passing within the JVM. Identifiers are assigned randomly. It does not support distributed simulation, nor does it allow for nodes to fail.

Simulator scenarios are specified using a simple script file. Churn 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.

Since it is designed as a teaching aid, the API is fairly straightforward. However the documentation is limited.

2) P2PSim: P2PSim [12] is a discrete event packet level simulator that can simulate structured overlays only. It contains implementations of six candidate protocols: Chord, Accordion [13], Koorde [14], Kelips [15], Tapestry [16] and Kademlia [17]. P2PSim has three implementations of Chord: the basic Chord with successor list and no finger tables; ChordFinger maintaining $(b-1) \times \log_b n$ immediate fingers; and the proximity-aware ChordFingerPNS [10], which keeps $(b-1) \times \log_b n$ PNS fingers. Choice of a PNS finger by a node is made by picking the node closest to itself from the corresponding immediate fingers successor list.

Event scripts can be setup to simulate churn but neither the churn nor the node failure statistics are exhaustive. P2PSim can simulate node failures and both iterative and recursive lookups are supported. Node IDs are generated by consistent 160-bit SHA-1 hashing. Distributed simulation, cross traffic and massive fluctuations of bandwidth are not supported. The C++ API documentation is poor, but implementation of other protocols can be built by extending certain base classes. Custom event generators can also be implemented by extending a base class. The P2PSim code suggests support for a wide range of underlying network topologies such as end-to-end time graph, G2 graph, GT-ITM, random graph and Euclidean graph, which is the most commonly used. P2PSim developers have tested its scalability with a 3,000-node Euclidean ConstantFailureModel topology. Work has also been done to simulate a 1,700-node Internet topology with the King data set [18].

3) Overlay Weaver: Overlay Weaver [19] is intended to be a toolkit for easy development and testing of P2P protocols. It provides functionality for simulating structured overlays only and does not provide any simulation of the underlying network. It is packaged with implementations of Chord, Kademlia, Pastry [20], Tapestry and Koorde. RPC can either be emulated using real TCP/UDP so the protocol can be tested on a real network, or using discrete event message passing within the JVM. Distributed simulation is possible, however

it is barely documented, so it was not experimented with. The code includes methods for identifier assignment based on SHA-1 consistent hashing or random generation. Both iterative and recursive routing can be simulated.

The interface is quite straightforward, consisting of a small number of command line tools and the emulator itself. However, the documentation is quite sparse and does not cover many of the simulators functions. Some of the tools are not documented at all so the source needed to be analysed to find out how to use them. However the API is clean and well designed so the source is quite readable.

Scenarios are defined in a simple script file which can be either manually created or generated by a tool. The script allows for a number of nodes running a particular protocol to be created and then put, get, suspend, resume and status display events can be scheduled. The simulator also allows for interactive control of nodes via a network socket.

The script generating tool provides a function for generating definitions for simulating churn. Churn is simulated by creating a specified number of nodes which then join and leave randomly for a specified period of time. So the levels of churn that can be simulated depend on the maximum number of nodes that can be simulated and how many of those nodes are needed for the main simulation.

The documentation says that Overlay Weaver has been seen to scale to 4,000 nodes. Its distributed simulation capabilities will allow it to scale further, allowing it to overcome hardware and software system limitation when used on a single machine. Since Overlay Weaver is intended as a tool to aid the initial design of P2P protocols, the simulator aspect of it is secondary. There is a graphical real-time visualiser which helps in understanding the protocols operation. There is also an undocumented tool which gathers some statistics on the number of messages passed, but detailed data for statistical analysis requires significant modifcation of the source code.

4) PlanetSim: PlanetSim [21] is a discrete-event overlay network simulator, written in Java. It supports both structured and unstructured overlays, and is packaged with Chord-SIGCOMM and Symphony [22] implementations. There are currently no mechanisms for gathering statistics, however it is possible to output the network topology graph in GML [23] and Pajek [24] formats.

The architecture implements the Common API (CAPI) presented in [25] so that the development and analysis of overlay algorithms is decoupled from that of applications. The CAPI tiers 0, 1 and 2 are mapped to the network, overlay and application layers of the simulator's architecture respectively. The layers can communicate with each other using upcalls and downcalls from the CAPI. By using this 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.

An overlay can run on a number of different underlying networks. Only the simple underlying networks, RandomNetwork and CircularNetwork are provided. These ignore latency costs, but it is possible to use PeerSim [26] or Brite [27] network information if more realistic underlying networks are required.

The authors intend to provide a TCP/IP wrapper that allows the simulator to be used for experiments. The same code used for simulation can then be used for experiments with nodes communicating using TCP or UDP. Other future work includes the implementation of statistics gathering mechanisms.

A research paper on PlanetSim has been published [28] and there is extensive documentation of the simulator at the PlanetSim website.

5) PeerSim: PeerSim [26] is an event-based P2P simulator written in Java, partly developed in the BISON project [29] and released under the GPL open source licence. It is designed specifically for epidemic protocols with very high scalability and support for dynamicity. It can be used to simulate both structured and unstructured overlays. Since it is exclusively focused 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, latencies, etc. Its extendable and pluggable component characteristics allow almost all predefined entities in PeerSim to be customised or replaced with user-defined entities. For flexible configuration, it uses a plain ASCII file composed of key-value pairs.

Two models of simulation are supported by PeerSim: cycle-based and event-based models. These models are achieved by its two different simulation engines: a cycle-based engine and an event-based engine respectively. 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 later model can support concurrency.

In terms of scalability, avoiding the overhead required to simulate low-level communication, PeerSim can simulate millions of network nodes. The developer claims that it can achieve a network consisting of 10^6 nodes in the cycle-based model.

Some components 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, such as node joins, node failure and node departure.

In the simulation, the network is represented by a collection of nodes which can hold one or more protocols. The communication between node protocols is based on object method calls. PeerSim does not support distributed simulation. Node identifications are generated incrementally as integers, but they can also be customised by user-defined mechanisms. The PeerSim developer website provides some tutorials and API documentation. However, the tutorials and examples are focused on the cycle-based model only and there is no in-depth discussion on the event-based model. PeerSim provides implementation class packages that support some well known models such as random graph, lattice and BA-Graph. It also has a class package that performs statistical computations.

PeerSim also provides predefined protocols for P2P simulation, such as OverStat [30], [31], SG-1 [32] and T-Man [33].

6) GPS: GPS [34] is a message level discrete event simulator with a built-in protocol implementation of BitTorrent. It allows for simulation of both structured and unstructured overlays. While it is a message level simulator, it also partially models the underlying network topology using the GT-ITM model of considering a Transit-Stub topology [35]. It does not model each packet, but provides a number of different flow level models. Distributed simulation is not implemented. The identifiers are generated incrementally and RPC is implemented as asynchronous message transfer in the discrete event scheduler.

GPS comes with a tool which provides a method for automatic generation of events in the context of the implemented BitTorrent protocol. Generated events are triggered at regular intervals. The author says that churn can be simulated using the event scripts. The API is poorly documented and support for extending the simulator to support protocols other than BitTorrent is limited.

The author has tested up to 512 BitTorrent peers with 1 BT tracker and a 1,054 nodes graph. Within the underlying Transit-Stub topology, three connections are considered: between transit nodes, between transit nodes and the stub and within the stub. Bandwidth and delay can be set by connection group. It can also be read from a bandwidth and delay matrix per individual connection. While cross traffic cannot be simulated, available bandwidth calculations can be estimated based on congestion using the following equation:

$$BW = \frac{MSS \times C}{RTT \times \sqrt{p}}$$

where BW is the available bandwidth, MSS is the maximum segment size used by TCP, C is a constant depending on whether immediate or delayed ACKs are chosen and p is the estimated packet loss rate on the congestion.

7) Neurogrid: Neurogrid [36], [37] is a P2P search protocol project that includes a single-threaded discrete event simulator [2], originally designed for comparing the Neurogrid protocol, Freenet [38] and Gnutella [39] protocols. The simulator works on the overlay layer level and can simulate either structured or unstructured protocols. It is packaged with implementations of Gnutella, Freenet and the Neurogrid protocols. It is a single-threaded discrete event simulator and it does not simulate the underlying network. Identifiers are generated incrementally. It does not support churn simulation. The author claims that 300,000 nodes have been simulated on a 32bit machine with at most 4GB RAM. It includes a flexible mechanism for statistics gathering which allows detailed data to be extracted. Simulation scenarios are specified in a simple parameters file. The file does not appear to include the ability to schedule events at specified times. Node failure is not currently implemented, but it may be easy to modify the simulator to implement the this behaviour. Documentation is a little disorganised in wiki form, but quite abundant.

8) Query-Cycle Simulator: The Query-Cycle Simulator [6] is a P2P file sharing network simulator that uses the Query-Cycle model. In this model, peers, both good and malicious, form an unstructured P2P network. The simulation process comprises of a number of query-cycles. Each cycle consists of one or more peers conducting a query to which other peers respond with matching results. The querying peers then choose a result and download a file from the corresponding peer. The cycle completes when all querying peers have downloaded a file. When the network contains malicious peers, the Eigentrust algorithm [40] is used and the cycle for a given peer only completes when a correct (e.g. uncorrupted) file is downloaded. As well as being used to demonstrate Eigentrust, it has been used to demonstrate the adaptive topologies algorithm [41].

Real-world behaviours of P2P systems have been considered in the simulator model. Content-distribution and peer behaviours are modelled using empirical studies, for example the P2P network uses the power-law topology, as the Gnutella network has been shown to have this property.

The simulator makes use of a graphical user interface from which the user can specify parameters for the network characteristics, the content distribution and the peer behaviours. Once these are committed and the simulation is in a running state, the properties can not be modified, but it is possible to halt, restart and save the simulation for later execution. A visualiser shows the state of the network as cycles are completed. Statistics, including the number of downloads and uploads, can be collected at each node.

9) Narses: Narses [42] is a scalable, discrete event, flow-based application-level network simulator. It allows for modelling of the network with different levels of accuracy and speed to efficiently simulate large distributed applications. Narses can simulate individual clocks with independent clock skews. Node IDs are represented as integers, with each new node assigned an incremented value.

There are four underlying network topology models varying from the least accurate to the most accurate and the slowest: Naive, NaiveTop, FairTopo and SafeFairTopo. Naive models a star topology where all nodes are connected to a hub and each connection to the hub has a value for bandwidth and latency associated to it. Contention between messages is not considered, making this model fast but inaccurate. NaiveTopo is similar to Naive except that it is not constrained to the star topology only. FairTopo takes into account the effects of competing TCP flows but assumes no bottleneck at the network topology core. It is more accurate and slower than the Naive models. SafeFairTopo is similar to FairTopo, but it dynamically checks that no link at the network core becomes a bottleneck. The transport layer in Narses contains two approximations of a socket interface, transport and reliable message transport. Use of a flow-based model makes Narses a significantly more efficient (at the cost of accuracy) network simulator than packet level simulators.

Narses has been tested with an example 600-node FairTopo model. Distributed simulation jobs can be run with Narses and data is exchanged using Java RMI. Narses does not contain

implementations of any overlay protocols.

D. Discussion

An attempt was made at implementing Chord-TON for Narses, but this was a difficult task because of the poorly documented API and no examples of a similar overlay implementation for the simulator. Difficulties also occurred when implementing Chord-TON using GPS. Although, fig. 1 of the GPS research paper [34] shows that the architecture is suitable for implementing key-based routing algorithms such as Chord, the architecture and interfaces are highly coupled with the BitTorrent algorithm. In private communication with the author, he confirmed that he has not attempted an implementation of Chord and is currently working on using GPS with Grid protocols. The Query-Cycle simulator proved unsuitable for an implementation of Chord, and Neurogrid was disregarded due to the lack of support for churn.

When performing a number of simple experiments using Overlay Weaver and its Chord implementation, some irregularities where found. Stabilisation of a 3-node chord network with one node forced to fail resulted in the stabilisation process taking an excessive length of time. Upon analysing the source code, a number of errors were found in the LinearWalker class, which is used by the Chord implementation. Successor Lists were not reconciled correctly and occasionally predecessor pointers were incorrect. The code to reconcile successor lists was fixed and to fix the predecessor bug, code was added to periodically check the state of predecessor nodes.

Overlay Weaver was tested using two machines, oak, a 64-bit Intel Pentium 4 dual-core 2.8GHz with 1GB of physical memory and 2GB swap, and jabba, a 32-bit dual Intel Xeon 1.8 GHz 1GB physical and 2GB swap, both using a 32-bit Java Virtual Machine on Linux. When testing the author's scalability claims, the dual Xeon could simulate a maximum of 2,700 nodes and the dual core P4, a maximum of 1350 nodes. When these limits were hit, the JVM reported that it could not create any more native threads. This proved to be a software limitation as discussed in [43]. The number of threads available to an application is limited by either the kernel or glibc. The OS scheduler allows non-root users to access only half of the value set in /proc/sys/kernel/threads-max. The threads-max value was set to 32,750 on jabba and 16,384 on oak. Setting thread-max on oak to that of jabba made no differences to the number of nodes that could be simulated. The glibc library can also be a limiting factor to the number of threads that can be run, but no attempt has been made at modification of the /usr/include/bits/local_lim.h header file and subsequent recompilation of glibc to test this.

Further analysis showed that each Overlay Weaver node uses 6 threads. Implementation of a network on a large scale may encounter operating system limits on the maximum number of threads that can be assigned to the JVM process.

For P2PSim, the example simulation experiments provided in the P2PSim package were run with a 3,000-node Euclidean ConstantFailureModel topology on the ChordFingerPNS implementation. ChordFingerPNS keeps $(b-1) \times \log_b n$ finger entries where b determines the base of the fingers and n is the number of nodes. Simulations were run using values 2, 4, 8, 16 and 32 for b and recursive lookups were used. Simulation experiments took a very long time to run on victory, a computer running Linux with two hyper-threaded 64-bit 3GHz Intel Xeon processors and 8GB RAM.

Packet-level simulators are inherently processor-intensive because at least two events (one to model the time taken to send the packet and one to schedule its arrival at the receiving node) get scheduled each time a node forwards a packet. Packet-level simulations are also expensive in terms of memory because the number of packets that simultaneously traverse a link increases linearly with the link's bandwidth-delay product.

P2PSim's simulation statistics lack detailed information about churn and node failures. It is not very easy to change the code to output such statistics.

Even though simulation in PeerSim has low overheads by abstracting away the lower network layers, the communication between node protocols in a simulation based on object method calls is not pragmatic in a real network. To implement an overlay algorithm in PeerSim, the algorithm may have to be adjusted to comply with the simulator's interface. The event-based model of PeerSim is more realistic than the cycle-based model because it supports concurrency, although it has less efficiency.

Unfortunately, the official PeerSim developer website provides examples and tutorials for the cycle-based model only although there are good API documents. Therefore, because of the lack of comprehensive documentation of the event-based model, it is still not straightforward to use this model for such simulation. This is despite the author's claim that it is not complicated to migrate cycle-based simulations to the event-based simulator.

It can also be noticed that most of component libraries provided in the source code of PeerSim were mainly designed to suit graph-like overlay protocols. Thus some modification of existing components and/or coding of additional components have to be taken into account for implementation of other protocols. Since the design of PeerSim aims to support modular programming based on objects (building blocks), these blocks can be substituted by another component implementing the same interface. However, in several cases, these pre-defined interfaces or modules may contain some declarations (properties and methods) that are neither relevant nor suitable to the implementation of a given protocol. It may be possible to ignore certain declarations in the interface, however if they prove to be too strict given the user's requirements, it may be necessary to reimplement the interface in question.

As PeerSim is configured using a plain text file, scheduling and parameter values for each of the components can be adjusted flexibly. But there are some limits on the pre-defined scheduler component such as it only supports simple scheduling. Therefore, some code modification may be required for enhanced scheduling. Although PeerSim offers components for common statistics, additional coding for user-defined data

collection is still necessary.

Experiments were carried on the scalability of Neurogrid. It was found that when simulating the Neurogrid protocol the running time increases linearly with the number of nodes. On a dual Xeon 1.8GHz computer running Linux, 0.18 second was required per node. So for the author's claim of 300,000 nodes, this computer would need 15 hours running time. However, since Neurogrid does not include an implementation of Chord, these results cannot be used to compare its scalability to the other simulators.

III. Future Work

The Chord-TON protocol will be implemented on all simulators where feasible. The Chord-TON simulation experiments will then be recreated with the aim of allowing for further comparisons of the simulators, with particular focus on the mechanisms for gathering statistics and analysing the obtained results. An investigation into the reproducibility of results across a number of simulators will then also be possible. Other protocols including unstructured ones shall also be considered for implementation and analysis.

IV. Conclusions

One of the disadvantages of simulators discussed by Law and Kelton [1], is the large amount of computing time that is required. This indeed was the case for a number of simulators that were evaluated. For example, in informal experiments using DHTSim, an experiment with 1,000 nodes joining a Chord network took in the region of several minutes, however increasing the number of nodes to 5,000 resulted in significantly more computing time being used. On the other hand, our experiments using another discrete-event based simulator, Neurogrid, gives an estimated running time of 15 hours for a 300,000 node unstructured network. Basic profiling of both these simulators showed high CPU usage, but low memory usage. However the same can not be said of Overlay Weaver, although in all our experiments we hit the thread limitations imposed by either the Linux kernel or glibc before physical memory limits were reached. For P2P simulations, scalability is an important property, but it is limited by the choice of simulator architecture, the hardware resources available and any operating system limitations. Further investigations are required to gain an accurate assessment of the scalability of the simulators that have been reviewed.

Many users of the P2P simulators will be interested in testing and evaluating overlay algorithms purely with respect to correctness and with little or no interest in the underlying network layer. Others may wish to test algorithms not only for correctness, but also to see how they behave with an underlying network layer of varying properties. This was the primary reason why Narses was included in our discussion. Although not an overlay network simulator (the authors position it more as an alternative to NS-2 [44]), it does claim to offer better scalability than NS-2 due to its flow-based rather than packet-based architecture. We believe that this may make it more suitable for overlay network simulations that require some consideration of the network layer. As the network layer is tunable, it would be possible to test the overlay with a simple network layer that will not impose greatly on the overlay algorithm, or one that is more complex and may have more of an effect on behaviour at the overlay level. PlanetSim also appears to offer some similar functionality in that users can modify the network layer properties such as topologies and latencies as required.

The usability of some simulators proved to be a serious issue. Due to poor documentation, implementing Chord on some of the simulators was a challenge. The only successful implementation was made on PeerSim after a number of modifications were made to the interfaces exposed by the simulator. There is a learning curve associated with all simulators, however this could be significantly eased with better documentation and by making use of languages such as Macedon [45]. Macedon is a language based on event-driven finite-state machines, which one can use to specify overlay protocols and compile them to languages such as C++. The compiled overlay code can then be easily integrated into a simulator.

Rather surprisingly, it seems that the authors of many of the simulators we evaluated considered statistical analysis of simulation results of little importance. Some of the simulators, such as PlanetSim, have no means to collect statistics and those that do provide some kind of statistics are limited in the variables that are made available to the user. Overlay Weaver allows access to a message counter, while P2PSim allows the user to produce graphs from a specific set of variables. The P2PSim variables are incomprehensive as they neglect churn and detailed information on node failure. A user using P2PSim or PeerSim wishing to change the variables for which data can be captured, would have to modify the code as required.

Reproducible results are important when performing simulations and experiments. An initial investigation into this was performed using a simple scenario of a key lookup on a stabilising Chord network of five nodes. Repeating this scenario a number of times on DHTSim resulted in both successful and unsuccessful lookups, but using Overlay Weaver, it always resulted in a successful lookup. These different results imply that perhaps the implementations of Chord used were different, perhaps one was incorrect, maybe both, or there was a problem with the simulators used. Given the state of current simulators, correctness is difficult to ascertain. Much work needs to be done in this area to improve the reproducibility of results, however making use of CAPI and languages such as Macedon could improve the situation. Finally, socket-like interfaces that allow code to be ported from the simulator to an experimental testbed such as Planetlab, would allow for the validation of results from a simulator with those from a testbed subject to more realistic network characteristics.

References

[1] A. M. Law and D. M. Kelton, *Simulation Modeling and Analysis*. McGraw-Hill Higher Education, 1999.

[2] S. Joseph, "An Extendible Open Source P2P Simulator," *P2P Journal*, pp. 1–15, 2003.

[3] D. Qiu and R. Srikant, "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks," in *SIGCOMM '04: Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. New York, NY, USA: ACM Press, 2004, pp. 367–378.

[4] B. Cohen, "Incentives Build Robustness in Bittorrent," in *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, June 2003.

[5] S. Saroiu, K. Gummadi, and S. Gribble, "Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts," *Multimedia Systems*, vol. 9, no. 2, pp. 170–184, 2003.

[6] M. T. Schlosser and S. D. Kamvar, "Simulating a File Sharing P2P Network," Stanford University, Tech. Rep., 2002.

[7] "Planetlab," accessed 01-May-2006. [Online]. Available: http://www.planetlab.org

[8] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, 2003.

[9] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *SIGCOMM '01: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. New York, NY, USA: ACM Press, 2001, pp. 149–160.

[10] F. Dabek, J. Li, E. Sit, J. Robertson, M. Kaashoek, and R. Morris, "Designing a DHT for Low Latency and High Throughput," *Proc. NSDI*, vol. 4, 2004.

[11] "DHTSim," accessed 01-May-2006. [Online]. Available: http://www.informatics.sussex.ac.uk/users/ianw/teach/dist-sys/

[12] "P2Psim: A Simulator for Peer-to-Peer (P2P) Protocols," 2005, accessed 30-April-2006. [Online]. Available: http://pdos.csail.mit.edu/p2psim/

[13] J. Li, J. Stribling, R. Morris, and M. Kaashoek, "Bandwidth-Efficient Management of DHT Routing Tables," *Proceedings of the 2nd NSDI*, 2005.

[14] M. Kaashoek and D. Karger, "Koorde: A Simple Degree-Optimal Distributed Hash Table," *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[15] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse, "Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead," *Group*, vol. 30, no. 1490, p. 23ms.

[16] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing," *Computer*, 2001.

[17] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, vol. 258, p. 263, 2002.

[18] "King Data Set," 2005, accessed 30-April-2006. [Online]. Available: http://pdos.csail.mit.edu/p2psim/kingdata/

[19] K. Shudo, "Overlay Weaver," 2006, accessed 01-May-2006. [Online]. Available: http://overlayweaver.sourceforge.net/

[20] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," *Lecture Notes in Computer Science*, vol. 2218, no. 0, pp. 329–350, 2001.

[21] U. R. i Virgili, "PlanetSim: An Overlay Network Simulation Framework," 2006, accessed 01-May-2006. [Online]. Available: http://planet.urv.es/planetsim/

[22] G. Manku, M. Bawa, and P. Raghavan, "Symphony: Distributed Hashing in a Small World," *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, pp. 127–140, 2003.

[23] Michael Himsolt, "GML: A Portable Graph File Format," Universität Passau, Tech. Rep. [Online]. Available: http://www.infosun.fmi.uni-passau.de/Graphlet/GML/gml-tr.html

[24] "Pajek - Program for Large Network Analysis." [Online]. Available: http://vlado.fmf.uni-lj.si/pub/networks/pajek/

[25] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica, "Towards a Common API for Structured Peer-to-Peer Overlays," *Proceedings of IPTPS*, vol. 58, 2003.

[26] "PeerSim P2P Simulator," 2005, accessed 01-May-2006. [Online]. Available: http://peersim.sourceforge.net/

[27] "BRITE - Boston University Representative Internet Topology Generator." [Online]. Available: http://www.cs.bu.edu/brite/

[28] P. García, C. Pairot, R. Mondéjar, J. PUJOL, H. TEJEDOR, and R. RALLO, "PlanetSim: A New Overlay Network Simulation Framework," *Proc. 19th IEEE ASE*, 2004.

[29] "BISON: Biology-Inspired Techniques for Self-Organization in Dynamic Networks." [Online]. Available: http://www.cs.unibo.it/bison/

[30] M. Jelasity and A. Montresor, "Epidemic-Style Proactive Aggregation in Large Overlay Networks," in *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*. Tokyo, Japan: IEEE Computer Society, Mar. 2004, pp. 102–109.

[31] A. Montresor, M. Jelasity, and O. Babaoglu, "Robust Aggregation Protocols for Large-Scale Overlay Networks," in *Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*. Florence, Italy: IEEE Computer Society, June 2004, pp. 19–28.

[32] A. Montresor, "A Robust Protocol for Building Superpeer Overlay Topologies," in *Proceedings of the 4th International Conference on Peer-to-Peer Computing (P2P 2004)*. Zurich, Switzerland: IEEE, Aug. 2004, pp. 202–209.

[33] M. Jelasity and O. Babaoglu, "T-Man: Gossip-based Overlay Topology Management," *Engineering Self-Organising Applications (ESOA05)*, 2005.

[34] W. Yang and N. Abu-Ghazaleh, "GPS: A General Peer-to-Peer Simulator and its Use for Modeling BitTorrent," *Mascots*, vol. 00, pp. 425–434, 2005.

[35] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," *INFOCOM'96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, vol. 2, 1996.

[36] "NeuroGrid," 2001, accessed 01-May-2006. [Online]. Available: http://www.neurogrid.net/

[37] S. Joseph, "NeuroGrid: Semantically Routing Queries in Peer-to-Peer Networks," *Proceedings of the International Workshop on Peer-to-Peer Computing*, 2002.

[38] I. Clarke, O. Sandberg, B. Wiley, and T. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," *Workshop on Design Issues in Anonymity and Unobservability*, vol. 320, 2000.

[39] "The Annotated Gnutella Protocol Specification v0.4." [Online]. Available: http://rfc-gnutella.sourceforge.net/developer/stable/index.html

[40] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The Eigentrust Algorithm for Reputation Management in P2P Networks," in *WWW '03: Proceedings of the 12th International Conference on World Wide Web*. New York, NY, USA: ACM Press, 2003, pp. 640–651.

[41] T. Condie, S. D. Kamvar, and H. Garcia-Molina, "Adaptive Peer-to-Peer Topologies," in *Peer-to-Peer Computing*. IEEE Computer Society, 2004, pp. 53–62.

[42] "Narses Network Simulator," 2003, accessed 01-May-2006. [Online]. Available: http://sourceforge.net/projects/narses

[43] E. Buchmann and K. Bohm, "How to Run Experiments with Large Peer-to-Peer Data Structures," *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pp. 27–36, 2004.

[44] "The Network Simulator - ns-2." [Online]. Available: http://www.isi.edu/nsnam/ns/

[45] A. Rodriguez, C. Killian, S. Bhat, D. Kostic, and A. Vahdat, "MACEDON: Methodology for Automatically Creating, Evaluating, and Designing Overlay Networks," *Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*.

APPENDIX

TABLE I
A LIST OF P2P NETWORK SIMULATORS

Name	URL	Language	Status
P2PSim	http://pdos.csail.mit.edu/p2psim/	C++	Active
PeerSim	http://peersim.sourceforge.net/	Java	Active
Query-Cycle Sim.	http://p2p.stanford.edu/	Java	Inactive
Narses	http://sourceforge.net/projects/narses	Java	Inactive
Neurogrid	http://www.neurogrid.net/	Java	Inactive[1]
GnutellaSim	http://www-static.cc.gatech.edu/computing/compass/gnutella/	C++	Inactive
GPS	http://www.cs.binghamton.edu/wyang/gps/	Java	Inactive[2]
myNS	http://www.cs.umd.edu/suman/research/myns/index.html	C++	Inactive
Overlay Weaver	http://overlayweaver.sourceforge.net/	Java	Active
DHTSim	http://www.informatics.sussex.ac.uk/users/ianw/teach/dist-sys/dht-sim-0.3.tgz	Java	Active
VPDNS	http://p2p.cs.mu.oz.au/software/vpdns/	C	Active
PlanetSim	http://planet.urv.es/planetsim/	Java	Active

[1]The Author of Neurogrid has stated his intention to resume work on the project via the Neurogrid-Mailing List.
[2]The GPS author stated via private communication a willingness to continue work on the project at some point this year.