An Adaptable, Role-based Simulator for P2P Networks

Vasilios Darlagiannis, Andreas Mauthe, Nicolas Liebau, Ralf Steinmetz
Darmstadt University of Technology
Multimedia Communications (KOM)
Merckstr. 25, 64283 Darmstadt, Germany
{Vasilios.Darlagiannis, Andreas.Mauthe, Nicolas.Liebau, Ralf.Steinmetz}@KOM.tu-darmstadt.de

Abstract

Large-scale P2P networks are distributed systems that are composed of unreliable, autonomous and heterogeneous components. A large number of competing approaches for P2P overlay networks design have been proposed that aim to address a number of conflicting requirements, such as scalability, reliability, efficiency, autonomy and fairness. Realistic evaluation of such networks requires emulation and simulation techniques to tune the vast number of parameters that are inherent in their design before they are actually deployed.

In this paper we present a new network simulator that follows a novel approach to handle the complexity in the development of diverse P2P systems based on roles. Of particular interest in the design of this architecture is adaptability of the network simulator to the wide range of overlay network design approaches. The purpose of this paper is to describe this architecture.

Keywords: *P2P systems, Overlay networks, Network simulation, Adaptability*

1 Introduction

Peer-to-Peer (P2P) systems are a challenging class of distributed systems where a set of often conflicting requirements have to be met. The underlying communication paradigms (as they are deployed in the constructed overlay networks) pose a multitude of non-functional requirements such as scalability, reliability, efficiency, autonomy and fairness. Typical sizes of currently deployed P2P file sharing systems reach the order of thousands or even millions of participants. Measurement studies of such systems show that the activity time of the majority of the participating nodes lasts less than an hour. The physical capabilities of the participants vary significantly, since peers can range from PDA-devices to large clusters, while connections can range from modems to high-speed links. This results in a highly heterogeneous environment. Certain overlay operations such as routing, indexing and structure maintenance are cooperative procedures that can evolve to complex tasks, especially when overlay designers aim at maximizing the effectiveness of the networks. The complexity of the overlay operations and the great multitude of the objectives can only be justified using simulation and (to a certain degree) analytical techniques before proceeding to the expensive step of system deployment.

In order to meet the critical set of the aforementioned (and possibly additional) requirements for the operation of the P2P overlay networks a great variety of approaches have been proposed. Analyzing the design mechanisms that characterize P2P overlay networks, three major design dimensions can be identified to classify the proposed systems (Figure 1). Overlay networks vary in their structural design from tightly structured networks such as Chord [16] or Pastry [15] to loosely structured ones such as Freenet [3] or Gnutella [10]. Moreover, overlay networks vary in the dependency of the peers on each other. Approaches such as Chord or Gnutella treat all of the participants equally, while hierarchical approaches such JXTA [18] or eDonkey [5] separate the responsibilities and assign more tasks to a small subset of more powerful peers (e.g. for indexing). Additionally, deterministic or probabilistic approaches (e.g. Bloom filters [2] are used in Oceanstore [13]) are selected to improve the accuracy or the efficiency respectively of the P2P systems. Finally, approaches such as Omicron [4] follow hybrid solutions to achieve higher adaptation on the dynamically changing factors of the environment.

The decisions taken in the overlay networks design dimensions affects greatly their ability to meet the aforementioned non-functional requirements. More specifically, certain design decisions might favor a subset of requirements on the expense of meeting others. For example, following a hierarchical approach reduces the maintenance effort but increases the dependability of the peers to a certain subset of them. Organized malicious attacks can have severe im-

pact in this case. Similar trade-offs appear with each identified dimension. In order to evaluate the large number of approaches and enable the fair comparison of the results in an as realistic as possible environment, emulation and simulation techniques are mandatory.

Developing a simulator in general is a process that imposes a number of design and implementation issues. The depth at which a simulator models the problems should be sufficient to get valid results. Additionally, it should provide sufficient functionality and be able to extend and adapt to cover new concepts. However, when models become large in size they require extensive support in memory and computation power to perform the experiments. A balance of the trade-offs in the level of details is required.

Modeling P2P systems poses certain requirements on the ability to capture the involved complexity. Effective mechanisms are required to model the user behavior, the details of the communication protocols at the overlay level and in certain occasions detailed modeling of the underlying physical network. To enable the simulation of complex approaches (e.g. Omicron) advanced concepts such as clustering mechanisms and role assignment should be supported as well.

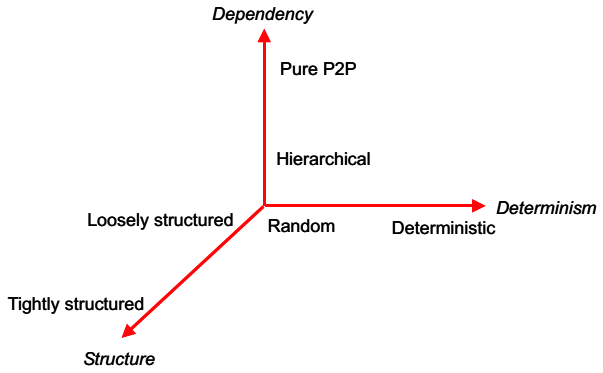


Figure 1. Overlay network design dimensions

1.1 Design Considerations and Related Work

A large number of simulators have been examined as potential candidates that could be used to extensively examine the characteristics of diverse P2P approaches. Ns-2 [7] was initially considered as a candidate mainly because of the great popularity of the tool. While Ns-2 is the default option for transport and lower (OSI) network protocols, it proved to be inefficient for P2P systems. The detailed model of the routers and the transmitted packets, made the simulation of large size P2P systems very demanding. Moreover, the duality in the language requirements made it less appealing.

J-Sim (formerly known as Javasim) [19] was another candidate that was investigated. Its layered-design sounded a promising alternative that could enable the simulation of larger systems. Initially, J-Sim was used as a base underlying network but due to the large resource consumption it was replaced by a more lightweight solution.

The Neurogrid Simulator [12] is designed specifically for P2P systems so it offers a more efficient model. However, the fact that it was initially designed for a specific system (Neurogrid) and the lack of sufficient documentation limited its potential as a general-purpose P2P simulator.

Ptolemy [6] is a general-purpose simulator that offers a large variety of simulation models. However, it is not a lightweight solution appropriate for large-scale P2P systems.

Finally, some other simulators were briefly examined, such as 3LS [17], Desmo-J [14], Simjava [11], myns [1]. For different reasons, they failed to meet the aforementioned requirements.

2 Architecture

2.1 Functionality Layers

In order to address effectively the complexity of P2P systems, an analysis of their functionality was performed to divide them in distinguishable parts. Three layers have been identified that can effectively divide their functionality[1]:

- *User behavior layer.* This layer captures the actions taken by the users. This might include their behavior using advanced services and express their cooperativeness or inclination to become free riders (a well-known phenomenon in the context of P2P systems). In addition, this layer captures important information for the reliability and stability of the P2P overlay. It is very important to simulate accurately the rates at which peers join and leave the P2P overlays. Based on this information prediction algorithms can be developed to optimize the operation of the overlays and the usage of the related resources.

- *Overlay protocol layer.* Overlay protocols (are application layer protocols for constructing the overlay networks) and performing the related operations are included in this layer. More specifically, this layer consists of the communication protocols of the overlay networks and the corresponding algorithms such as message routing, dissemination of system-related information (e.g. estimated peer reliability), maintenance of the overlay structure, etc. This layer is the most significant one with respect to the P2P paradigm.

[1] A similar functionality separation is proposed in [17].

- *Network transmission layer.* This layer captures the details of the physical network (e.g. latency and bandwidth) and the related communication protocols that are located at the four lower layers of the OSI model. Although in many cases the details of this model are not of high importance for the researchers of P2P systems, they can play very important role (especially in cases where network QoS is of great importance such as media streaming within P2P overlays). This layer is responsible to model the details of delivering packets between peers.

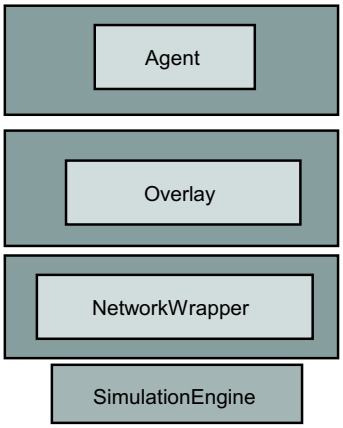


Figure 2. Functionality layers

Figure 2 shows the functional architecture of the simulator. It is designed in a way that can naturally capture the needs of the P2P systems as they have been analyzed. The *Agent* layer represents and models user's behavior, the Overlay layer includes the details of the P2P protocols and the *NetworkWrapper* is the layer that enables the attachment of a variety of network models that can capture the physical network details in a plethora of levels. Finally, the architecture includes a core component of the simulator, the *SimulationEngine*, in addition to the functional layers. The simulator has been designed in a way that a great multitude of simulation engines (e.g. event-based, process-based) can be attached to perform the simulation. A default simulation engine is provided, though users can exchange it with customized ones that may be more appropriate for certain scenarios.

2.2 Components

The aim of developing this tool is to provide a general-purpose P2P simulator, which is not dedicated to any specific P2P architecture or system. For this reason it has been designed following a framework-like approach. As a first step, many different P2P overlay architectures (structured, unstructured, flat, hierarchical, hybrid) have been analyzed to capture their requirements.

Following this step, the most important components have been identified to represent them as well-defined entities in the software architecture of the simulator. The framework of the simulator is based on the concept of plug-ins. A clear interface is defined for every identified core component to offer the required functionality. Either a default implementation (in cases where it offers general-purpose functionality suitable for any P2P approach) or an abstract base implementation with general functionality is provided. It is straightforward to replace the default implementation of each component with a customized one since there is a clear interface that can be referenced in the implementation of the other components. Changes are only locally required inside the component. As an example of a component of this category, consider the *SimulationEngine*. Users can replace the default event-based engine with a process-based one. An abstract base implementation with general functionality is provided in cases where no default implementations can be proposed. The base implementation has to be extended to satisfy the needs of the particular system. The *Agent* is an example component of this category that has to be extended.

The interaction with the default or the customized components takes place via the well-defined interfaces, making it implementation detail agnostic. Figure 3 illustrates the aforementioned design patterns of the core components.

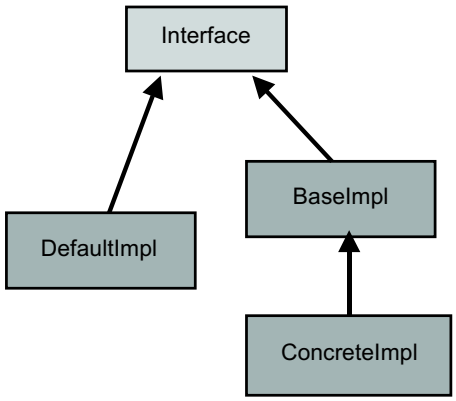


Figure 3. Component Design Pattern

The extended analysis of many well-known P2P systems provided a comprehensive understanding of their structure. The most important identified concepts are listed here.

- *Peer*. It represents the peer itself encapsulating all the related components.
- *GUID*. Every peer can be uniquely identified using a Globally Unique IDentifier (GUID). In many cases, no

central authority exists to guarantee the uniqueness of the identifiers so efficient distributed algorithms should be employed.

- *Overlays*. They encapsulate the components that are related to the construction of the overlay networks. In many cases peers participate in many sub-overlays (especially in cases of hierarchical approaches).
- *Cluster Map*. Many systems introduce the concept of clusters of peers, where peers are grouped together with respect to certain constraints, user interests or network proximity-based requirements. Cluster Maps are entities that summarize the membership information of each cluster.
- *Routing Table*. Each peer maintains a number of neighbors, in order to construct the overlays. The Routing Tables encapsulate this information. The scope of each Routing Table is the corresponding overlay network.
- *Roles*. In many cases, having identical responsibilities for each peer can lead to inefficient systems. Many approaches propose the identification of roles, appropriate for different peers in order to overcome this problem.
- *Message Dispatcher*. Although this component is not very important from a conceptual point of view, it plays an important role in the correct and valid operation of the peers. Peers that participate in multiple overlays and hold a number of different roles require a mechanism to direct the incoming messages to the appropriate receivers.
- *Messages*. A large number of different messages may exist in P2P systems. They are vital components to develop the distributed nature of the P2P systems.
- *Protocols*. This is a concept that represents the overlay protocols. It is helpful in the modeling of the systems. A protocol is constructed by s set of messages.
- *Documents*. Documents represent items of interest that can be shared between peers. They are identified by GUIDs.
- *Index*. Indices are structures that summarize the location of the documents in the P2P system. They can be detailed structures or can be modeled using appropriate probabilistic methods to simulate their operation.
- *Cache*. Caches are document repositories of limited size. They can significantly improve the performance of the overlay operations (e.g. in cases of Zipfian request distribution).
- *Link*. Links encapsulate the details of the physical connections among peers. They can include complex models of the latency and the bandwidth or simple ones that offer certain functions (e.g. in-order message delivery).
- *NetworkWrapper*. NetworkWrappers are general objects that adapt and hide the details of the underlying network from the Overlay layer. Links are encapsulated inside them.

Figure 4 shows the architectural components of each Peer. Components are grouped using similar (gray-level) colors to denote a higher dependence and interaction among them. Multiplicity of the entities is demonstrated with overlapping rectangles.

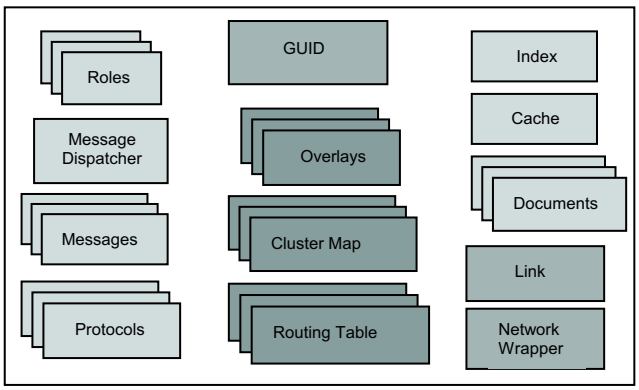


Figure 4. Core peer components

2.3 Networking Support

As it has already been mentioned, the architecture of the simulator enables the easy adaptation of many components to fulfil a variety of functionalities. Holding the same axiom for the network layer, a general interface (*NetworkWrapper*) has been defined to encapsulate the details of the networking layer. A lightweight default implementation is provided that offers a simple but sufficient for most of the cases solution. The characteristics of this model are given in the following list:

- *Reliable communication*: Taking into account the fact that most of the deployed P2P systems employ TCP as a transport protocol to exchange messages, this model provides reliable communication in end-to-end basis. This means that no packets will ever be dropped because of buffer overflows in routers.
- *Peers can depart arbitrarily (fail)*: Peers can depart without prior notification. This is the only type of fail-

ure that can take place in this simple model. Network connectivity is assumed.

- *In-order message delivery*: Messages that belong to the same session are guaranteed to be delivered in-order. Again, this follows from TCP like modeled connections.

- *Random delay on links*: In this model links are entities that connect two peers as illustrated in Figure 5. The details of the physical network are hidden in this model. In order to provide a simple solution that takes into account the delay in delivering messages, a probabilistic approach has been chosen. The delay can take some bounded random values. If the delivery of the previous message is still in progress then this delay is always additive to the delay that was calculated to the previously submitted message over the same link. Although this approach is not very sophisticated, it provides a simple and effective mechanism to guarantee in-order delivery of messages.

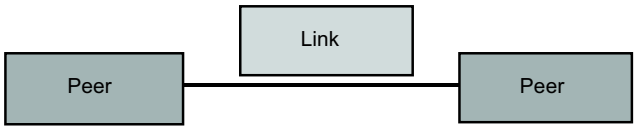


Figure 5. Link modelling

2.4 Simulation Engine

A framework-like approach has been followed to allow the easy replacement of the default simulation engine with customized engines. The requirements for the simulation engine are efficiency and sufficient functionality. Figure 6 provides the software design of the default simulation engine. The *SimulationFramework* is a general-purpose component that is called by the *Application* to create certain scenarios (build nodes, agents, topologies, etc.), start them and process the results of the experiments. The *Simulator* is an interface that provides vital information about the status and the progress of the simulation. The *Scheduler* enables the insertion of the active components into the simulator that can generate events. It ensures the correct execution of those generated events in order to provide valid experiments. The duration of each experiment is controlled by the scheduler and the parameters that the *Application* defines. *Peers* add the generated events indirectly via the *NetworkWrapper* interface. The events themselves are modeled by the so-called *Event* class. They provide information like the scheduled execution time, the content of the messages, the destination and the originator. An efficient implementation of an ordered *Queue* is used to store the generated messages and provide them for execution. The default implementation of the simulation engine is single-threaded with respect to the model actions. It is not very clear if there will be any performance improvement by making it multi-threaded. Thus, it was decided to avoid the complexity of multi-thread programming. However, costly I/O operations can be performed by different threads to improve the overall performance.

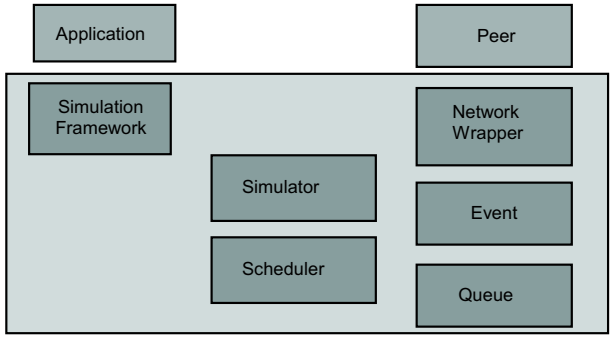


Figure 6. Simulation engine

3 Roles

A novel characteristic of this simulator is the ability to support multiple roles for each peer in an effective way. Roles represent responsibilities for certain operations, such as routing or indexing. In order to achieve that an appropriate abstract framework is necessary. A common role has been identified for each peer independently of the selected system to be simulated. It is the *Connector* role, which provides the establishment of the inter-peer connections (not necessarily in a TCP way):

- It handles the creation and tear-down of each connection.

- It checks the validity of the connections with ping/pong messages.

Furthermore, the Connector offers the means to dynamically update the roles "installed" on each peer. In addition, it dynamically updates the local Cluster Maps as they evolve over time.

Table 1 provides the identified roles as they appear in the different node types of three P2P networks. These P2P networks are representative systems of different design approaches. JXTA is a hierarchical system, Chord

is a non-hierarchical P2P approach and Omicron is a hybrid one. Five core operations are shown: *Routing* (RT), *Caching* (CH), *Indexing* (IX), *Maintaining* (MN) and *Connecting* (CN). The advantage of the role-based approach is the simple reconfiguration of the responsibilities and the reusage of the implemented modules in multiple ways. For example, users can share the same Indexing mechanism between Chord-based and Omicron-based experiments. Alternatively, simple JXTA peers can dynamically evolve to Rendezvous by installing additional roles.

Table 1. Roles

Node type	*Operations*				
	RT	CH	IX	MN	CN
JXTA Rendezvous	✓	-	✓	✓	✓
JXTA Peer	-	✓	-	-	✓
Chord	✓	-	✓	✓	✓
Omicron Router	✓	-	-	-	✓
Omicron Cacher	✓	✓	-	-	✓
Omicron Indexer	✓	-	✓	-	✓
Omicron Maintainer	✓	-	✓	✓	✓

4 Message Handling

Messages are the entities that encapsulate the information to be exchanged among peers to realize the overlay protocols. In complex systems such as JXTA, peers can participate in multiple overlay networks (e.g. per peer group). Moreover, they can be assigned multiple roles (Router, Maintainer, etc.). Peers communicate with their neighbor peers for each assigned role in every overlay abstraction. Each message carries a number of fields that can augment in this process to handle efficiently the large number of potential receivers. The following common fields are included in each message:

- *Style*. The Style determines the way a message is transmitted to the destination. There are two different styles. The recursive style where the message is forwarded to the most "promising" neighbor and then this neighbor forwards similarly the message to the most promising neighbor of his until the message arrives to the final destination. The second style is the iterative communication style. Also in this case, the message is forwarded to the most promising neighbor. However, the selected neighbor replies to the original peer with the address of the most promising of its neighbors instead of forwarding the message directly. Then the original peer is responsible to contact the new peer and get a new address until it reaches the final destination. Although the second communication style is more costly in terms of latency compared to the first one, it has the benefit that peers can check the progress of forwarding messages themselves, thus, avoiding the problems that might appear with the presence of malicious peers.
- *Type*. The type of the message determines the functionality that this message serves. For example, it can be a maintenance or a routing message.
- *Name*. The name discriminates the messages and enables the selection of how to react to each one.
- *Scope*. The scope of a message determines the overlay through which this message should be delivered to the destination. It might be possible that a destination node might be reachable through different sub-overlays at different cost.

Figure 7 displays the common structure of each message. Moreover, this figure shows how messages are handled when they arrive at one peer. The Message Dispatcher checks the Type of the message and it passes it to the appropriate Role. Subsequently, the Role examines the Name of the message in order to process it and after it performs the appropriate local actions, it examines the scope of the message in order to select the appropriate Overlay to forward or to reply to this message.

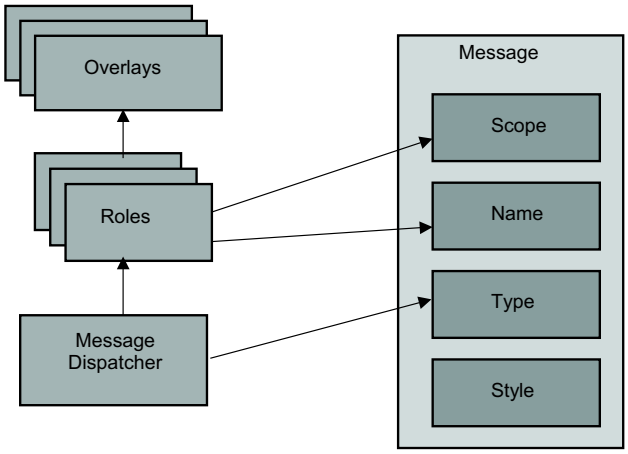


Figure 7. Message handling

5 Evaluation

5.1 Performance

Although the performance of a simulator is highly related to the details of the embedded model, the efficiency

of the implementation itself plays an important role for the acceptance of the tool. Using profiling tools, the implementation and in some cases the design of the tool has been improved. Figure 8 provides two graphs that demonstrate the consumption of the resources during a number of experiments. As it can be easily obtained from those graphs, both the memory consumption and the execution time increase linearly with respect to the size of the topology. This is very important and proves in a certain degree the validity of the implementation.

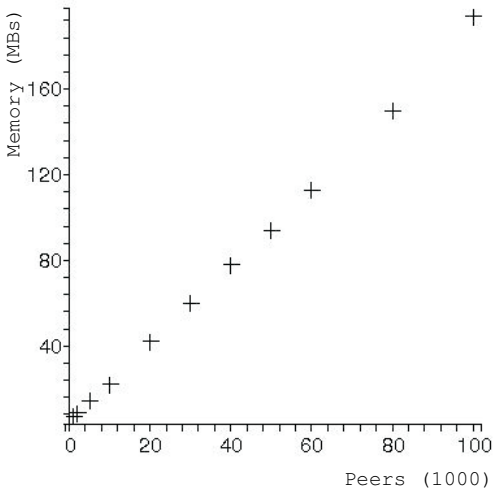
Initially, J-Sim was selected to model the details of the underlying network. Although that solution offered a rich model, the performance was heavily decreased. The analysis of the results of profiling the used resources while using the simulator showed that approximately 75% of the CPU and memory was consumed by the J-Sim component. Thus, this led to the development of the lightweight approach described in Section 2.4. The selection of the heavy J-Sim based solution is still available in cases where a very detailed model is necessary.

However, in order to evaluate better the information provided by these graphs it is necessary to describe the experiments in more detail. They include the construction of Omicron overlays and the construction of clusters with peers assigned to Omicron roles. They were performed using a laptop at 1.0 GHz speed and 384 MB memory. The employed JVM was JDK1.4. In summary, it can be derived that it takes less than a minute time and less than 200 MB of memory to construct a complex overlay like Omicron of size 100.000 active nodes. The observed linearity in terms of physical resources promises simulation experiments sized as large as half a million nodes in moderately more powerful machines dedicated for simulation.

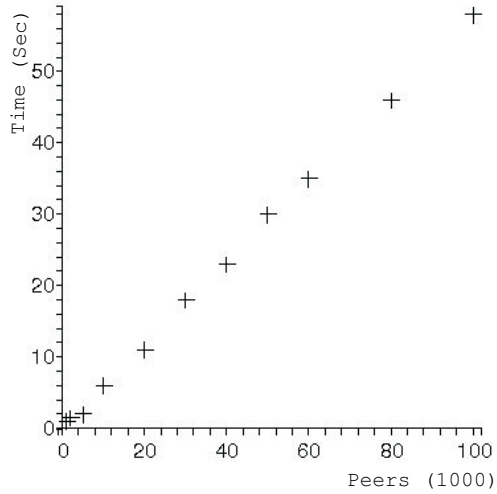
5.2 Visualization of Experiments

In order both to demonstrate the results and observe/debug the simulated experiments, graphical tools are necessary. A variety of graph tools can be used to get graphs similar or more complicated than the ones provided in the previous section. However, the graphical animation of the experiment provides additional information about the experiments, which can add great value. For this reason, the simulator can generate Nam-like traces that can be consumed by the Nam tool (Network animator). Since Nam has been designed to fulfil the needs of Ns-2 [8] and its implementation was based on C++ and Tcl languages, an alternative tool proved to be more useful for the needs of the simulator. This is Javis (Java Visualizer) [9]. It is capable of tracing Nam-files, but with more clear software architecture. Javis was enhanced to fit to the needs of the P2P simulator in three ways:

- *Support directed links*. Javis and Nam support only bidirectional links.
- *Control overlay topology*. P2P overlay networks can be displayed easier.
- *Remove failed nodes and links*. Javis as well as the original Nam have no ability to remove failed nodes and links.



(a) Memory



(b) Cpu time

Figure 8. Simulator Performance

6 Conclusions

Though simulation alone is not sufficient to guarantee successful deployment of P2P systems over the Internet, it is an obligatory step in the development cycle.

A well-developed simulator should carefully address the need for functionality and the achieved performance. Moreover, an extendable and adaptable design is necessary to enable the integration of particular systems with the general-purpose mechanisms offered by the tool.

The described tool is designed to simulate network operations particularly for the case of P2P systems. The role-based approach increases its suitability for a variety of overlay network designs. The layered-based approach enables the attachment of various underlying network models to fit the level of desired detail (e.g. J-Sim).

Furthermore, performed experiments show the efficiency achieved in terms of resource requirements (memory and computation) for sufficiently large P2P networks. The high adaptability of the simulator to a variety of simulation engines increases its suitability to a vast number of modeling approaches.

Currently three P2P systems have been ported for simulation: Gnutella, Chord and Omicron. Simulation is a preferable approach to compare P2P networks since identical environment conditions can be assured to obtain objective results.

Acknowledgments

This work has been performed partially in the framework of the EU IST project MMAPPS "Market Management of Peer-to-Peer Services" (IST-2001-34201). The authors gratefully acknowledge ongoing discussions with their MMAPPS partners.

References

[1] Suman Banerjee. myns (P2P) simulator. http://www.cs.umd.edu/ suman/research/myns/, 2002.

[2] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[3] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *ICSI Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[4] Vasilios Darlagiannis, Andreas Mauthe, and Ralf Steinmetz. Overlay Design Mechanisms for Heterogeneous, Large Scale, Dynamic P2P Systems. *Journal of Networks and System Management*, 12(3):371–395, 1 2004.

[5] eDonkey2000. http://www.edonkey2000.com, 2003.

[6] Shuvra S. Bhattacharyya et al. Heterogeneous Concurrent Modeling and Design in Java: Ptolemy II Design. http://ptolemy.eecs.berkeley.edu/ptolemyII/designdoc.htm, 2003.

[7] Kevin Fall and Kannan Varadhan. The ns manual. http://www.isi.edu/nsnam/ns/doc-stable/index.html, 2000.

[8] Sally Floyd and Vern Paxson. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking (TON)*, 9(4):392–403, 2001.

[9] Scott Gammill. Java Visualiser (Javis 2.0). http://cs.baylor.edu/ donahoo/NIUNet/javis.html, 2000.

[10] Gnutella 2. http://www.gnutella2.com.

[11] Fred Howell and Ross McNab. Simjava: a discrete event simulation package for Java with applications in computer systems modelling. In *International Conference on Web-based Modelling and Simulation*, January 1998.

[12] Sam Joseph. An Extendible Open Source P2P Simulator. *P2P Journal*, November:1–15, 2003.

[13] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Chris Wells, and Ben Zhao. Oceanstore: an architecture for global-scale persistent storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 190–201. ACM Press, 2000.

[14] Bernd Page, Tim Lechler, and Sönke Claassen. *Objektorientierte Simulation in Java mit dem Framework DESMO-J*. Libri Books on Demand, 2000.

[15] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.

[16] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 149–160. ACM Press, 2001.

[17] Nyik San Ting and Ralph Deters. 3LS-A p2p network simulator. In *Poster in The Third IEEE International Conference on Peer-to-Peer Computing*, September 2003.

[18] Bernard Traversat, Ahkil Arora, Mohamed Abdelaziz, Mike Duigou, Carl Haywood, Jean-Christophe Hugly, Eric Pouyoul, and Bill Yeager. Project JXTA 2.0 Super-Peer Virtual Network. http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf, May 2003.

[19] Hung ying Tyan. Design, Realization, and Evaluation of a Component-based Compositional Software Architecture for Network Simulation. http://www.j-sim.org/whitepapers/tyan_thesis.pdf, 2002.