A Generic Peer-to-Peer Network Simulator

Nyik San Ting
Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, S7N 5A9
nyt431@mail.usask.ca

ABSTRACT
Peer-to-Peer (P2P) systems are emerging as a "new" form of distributed computing with a strong emphasis on self-organization, decentralization and autonomy of the participating nodes. Self-organization, autonomy and decentralization allow for highly adaptive, robust and scalable networks making P2P an increasingly interesting way to design distributed systems. Unfortunately it is still very difficult to develop P2P applications due to the complex and ill understanding of interdependencies between the users, application, protocol and physical network. Furthermore, the P2P topology and the underlying network topology can greatly influence the behavior of the P2P system.

This paper provides an overview of P2P protocols and a review of existing simulators. Based on the shortcomings of existing simulators a novel layered simulator for P2P networks is introduced and evaluated.

Keywords
P2P Networks, Simulator

1. INTRODUCTION
In the middle of the 90's, the computational resources of ordinary desktop computers began to exceed the needs of their users, creating an ever-growing growing pool of unused computational resources. An Intel study [7] estimates that in the average organization the combined computational resources of the desktop machines are 2.5 times larger than those of their centralized servers and high-end compute nodes. The now famous SETI@Home [23] project demonstrates that by using simple P2P techniques it is possible to harvest the scattered resources of thousands of desktop computers in an efficient way enabling the analysis of large data at virtually now costs. The success of the SETI@Home project, which forms one of the most powerful compute-networks today, has resulted in an increasing interest in the study and deployment of P2P networks.
The field of P2P networks is still in its infancy with new applications and protocols emerging on a nearly daily basis. However, due to the difficulties in evaluating them prior to their large-scale deployments, they are often short-lived – disappearing as fast as they emerge – normally due to bad performances. As the list of failed P2P systems and protocols grows the need for evaluation tools (e.g. performance evaluation) increases.
Testing a system performance prior to its deployment is a fairly common element in the software development of applications. According to Pawlikowski [17], there are two main possible experimentation streams: experimentation with the actual system and experimentation with a model of the system (see Figure 1).
P2P networks tend to be large, heterogeneous systems with complex interactions between the physical machines, underlying network, application and user. Hence, testing of a "running" P2P network or protocol in a realistic environment is often not feasible. However, it is possible to use a simulation of such networks to evaluate the applications and protocols in controlled environment.

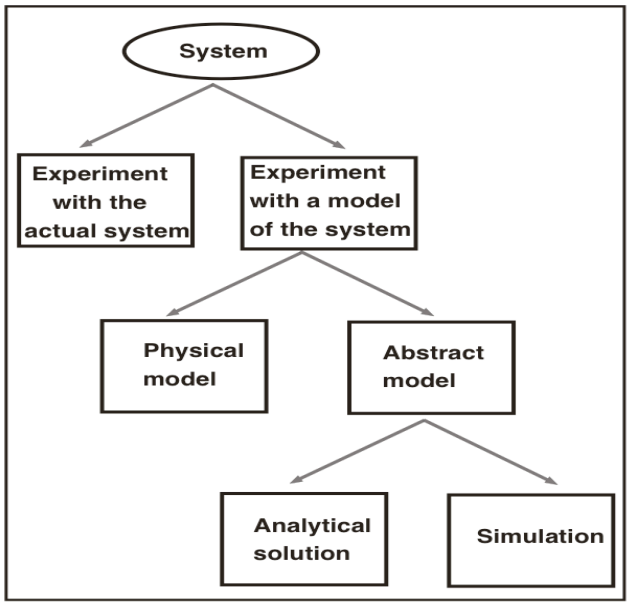

Figure 1: Studying System Performance

As a result of the excessive bandwidth consumption of Gnutella v0.4 [4], a new generation of protocols and systems, that emphasizes self-organizing due to "learning" from past experiences, emerges. These protocols are tightly coupled to the network topology, application and the users behavior. As the result, a P2P simulation, which maps to the real world, needs to reflect tight dependencies between the users, application, protocol and physical network. The aim of this

paper is to motivate the need for generic P2P network simulators. It is structured as follows: in section 2 the major P2P protocols are presented. This is followed by an overview and evaluation of the P2P simulators currently used. Based on the shortcomings of existing simulators a new approach for simulating P2P systems is presented in section 4. The paper concludes with a summary and a presentation of future work.

2. Peer-to-Peer Protocols

Over time, a variety of P2P protocols and systems have emerged. This paper therefore focuses only on the most prominent examples of certain types of P2P protocols.
Resource allocation is one central element in every P2P protocol. Existing approaches can be grouped [13] into one of the following three basic categories:

2.1 Centralized directory model.

This model is based on the availability of a server used as a centralized directory service, managing the information of all the participating peers in the P2P network. In order to join and participate in the network, a peer must directly connect to the central server. While the peers mainly in full control of their locally owned resources, they need the server for advertising their resources or locating needed resources of other peers.

This centralized approach solution provides good resource allocation performance and network manageability. However, using a centralized component introduces a central point of failure.

2.1.1 Napster [14]

Napster was a centralized P2P file-sharing network that was forced to shut down as a result of copyright infringements. In Napster, a cluster of central servers maintained the information of the content offered by the peers of the network. All the peers in the Napster network had to connect to the server, which also handled their resource request queries. Upon receiving the results of the match from the central server, the peer, establishes a connection to the resource-providing peer and started to consume the resource e.g. downloads the file directly.

2.1.2 SETI@Home

SETI@Home was designed to aid the process of discovering extraterrestrials by harvesting the unused processing power of idle desktops via the Internet. A centralized server stores and packages the signal data of radio telescopes into small chunks that can be processed by an average desktop machine. Users willing to participate in this "hunt for ET" have to download and install the signal processing data provided by SETI@Home. Depending on the OS, users download a SETI@Home screen-saver that is used as a means of detecting idle-cycles or software designed to run as a background process. When the SETI@Home software recognizes that the host has idle resources (based on the settings of the user), it contacts the central server of SETI@Home to download a chunk of data (ca. 200 KB). Upon the successful download, the peer starts a series of signal processing activities and sends the results (ca. 80 KB) back to the server requesting new data and restarting the compute cycle.

Despite the impressive performance of the SETI@Home network, it is important to point out that it can only handle single process multiple data (SPMD) compute problems due to the inability of SETI@Home peers to communicate and therefore coordinate their processing of data.

2.2 Flooded request model

The flooded request model is a "pure" P2P model since it does not require any centralized infrastructure. Due to the absence of any predefined structures, resource requests of a peer are handled by use of message flooding.

Gnutella [4] is a good example for the flooded request protocols. Gnutella has been designed for information storage and searching in distributed system with decentralized control. Currently two main Gnutella protocols exist: 0.4 and 0.6.

2.2.1.1 Overview of Gnutella v0.4

A Gnutella 0.4 peer (X) connects to the network by establishing a TCP/IP connection to a peer (Y) that is currently on the network. After connecting, X sends a request string ("GNUTELLA CONNECT/0.4\n\n") to establish a link to peer Y. Peer Y can accept the connection, by sending "GNUTELLA OK\n\n", or refuse the connection, by sending any responses other than the accept connection string. Once the peer X is connected to the network, it can communicate with other peers by sending and receiving Gnutella messages (also called descriptors).

Every descriptor has 22 bytes of message headers consisting of: descriptor ID (byte 0 to 15), payload descriptor (byte 16), TTL (byte 17), Hops (byte 18), and payload_length (byte 19 to 22). The descriptor ID is a unique identifier of the message in the network and typically created by using a random generator. The Time-To-Live (TTL) is the number of times the message will be forwarded by Gnutella peers before being deleted. Hops represent the number of times the message has been forwarded by peers. Each time a message is being forwarded, the message's TTL is being decremented by one and the hop is increased by one. The Payload descriptor contains the code for the type of the message: 0x00 for Ping message, 0x01 for Pong message, 0x40 for Push message, 0x80 for Query message, and 0x81 for a Query_Hit message. The payload_length contains the length of the remainder message (payload) following after the header. Hence, a message's total length is 22 bytes + the payload_length; the next message is located exactly payload_length after the current header.

A peer can discover the information (IP address and port number, number of resources, etc.) about other peers by sending a ping message. A ping message has payload_length of zero - no payload. Upon receiving a Ping message, a peer, will forward the ping messages to all the directly connected peers, except the peer who sent the ping message. At the same time, the peer will send a pong message, which has the same descriptor ID as the corresponding ping message, in response to the peer who sent the ping message. A pong message has a payload that contains the number of files shared, the number of kilobytes shared, IP address, and port number of the responding peer.

While ping and pong messages are used for discovering other peers in the network, query and hit messages are used for locating resources. To locate a resource, a peer sends a query message that has a payload containing the required minimum network speed of a potential resource providing peers and a search string for describing the requested resource. Similar to the ping messages, a query received by a peer is broadcasted to all the directly connected peers (except the sender of the

request). Similar to the ping/pong messages query messages are also subject to hop increase and TTL decrease as a means of limiting the range of the query message. Each message will be replicated and propagated throughout the networks until it either comes to a peer that matches the search criteria or exceeds its TTL value. Peers that are capable of offering the requested resource respond with a query_hit message. A query_hit message contains as payload information about the resource-providing peer.

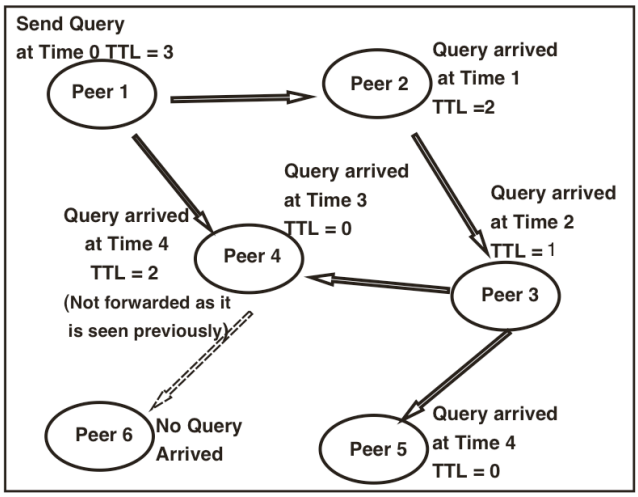

Figure 2: The flat Gnutella 0.4 Network

Hence, a resource contained in the network is not searchable if the minimum distance (hops) between the peers is greater than the lifetime of the request. However, a resource located within the lifetime of the request might not be reachable as the result of "short-circuiting" effect due to the latency of the network [11]. "Short-circuiting" effect occurs when there is a the request with smaller lifetime arrived a node through the smaller latency path, then when the same request with higher lifetime arrived through the higher latency path, this request is discarded. (see Figure 2)

Pong messages, and query_hit, messages are routed along the same path back to the peer that launched the original ping and query messages. This is made possible by keeping a record of a predefined number of message descriptor Ids, and the corresponding payload descriptors. When a peer received a response message (pong or query_hit), it will check if it generated the original message (ping or query). If the peer is not the destined receiver, it will check if it has seen such a ping, or query, that has the descriptor id. If no such a ping, or query, passed through the peer, the message will be dropped; else it will send the response message to the connection that sent the corresponding ping, or query message. Once a peer received the Query_Hit message, generated by another peer, in response to its query message, it will access the resource using the http protocol.

2.2.1.2 Gnutella v0.6 [12]

Similar to Gnutella v0.4, a peer connects to the network by first establishing a link to another already connected peer through TCP/IP. The peer wishing to join sends the request to connect string "GNUTELLA CONNECT/0.6<cr><lf>" and optional data to describe itself thus making a connection more likely.

```
GNUTELLA CONNECT/0.6<cr><lf>
User-Agemt:BearShare<cr><lf>
X-Ultrapeer:True<cr><lf>
Listen-IP:12.134.4.23:6349<cr><lf>
<cr><lf>
```

Figure 3: Connection Request in 0.6.

The sending of additional data allows for the introduction of UltraPeers [24] (super nodes) in the peer network. The concept of UltraPeers is not part of the v0.6 specification; it is an optional implementation that helps reduce the network bandwidth consumption resulting from the flood of ping and query messages. However, since the concept of UltraPeers is a way to reduce the network consumption, almost all the Gnutella v0.6 peers are implemented to support the UltraPeer functionality. In addition the departure of a peer can be announced by itself using the bye message thus minimizing the problem of dead links.

In a Gnutella 0.6 network, there are two kinds of peers: UltraPeers and leaf nodes. Leaf nodes are the nodes that are not powerful enough to be an UlraPeer or fail to provide the UltraPeer functionality. Leaf nodes only maintain connections to UltraPeers and/or to the other leaf node that wants to joint the network but can't find a suitable UltraPeer. An UltraPeer is a Gnutella peer that maintains many connections to other UltraPeers and a large set of leaf nodes. The UltraPeer maintains a record of the leaf node's resources and acts as a shield for its leaf nodes to the flood of ping and query messages.

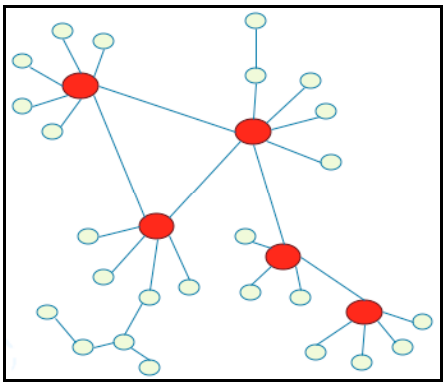

Figure 4: Hierarchies in Gnutella 0.6

2.2.2 NeuroGrid [8]

NeuroGrid was initially designed as an alternative routing model for Gnutella. It focuses on minimizing the sending of messages by increasing the query processing cost for peers. It is designed to "provide a method of communication that will piggy-back on top of http"[9].

In NeuroGrid, the resources in the P2P network are assumed to be associated with a set of keywords. When searching for a resource, the requesting peer needs to know the keywords that are associated with the resource. Unlike in Gnutella, the searches are not being forwarded to all the connections automatically; NeuroGrid expects the peer to decide how to route the request. A NeuroGrid node uses a model for neighboring peers and their contents. Using a decentralized

routing model allows a more efficient routing at the expense of added processing overhead.

2.3 Document routing model

Document routing is the most recent model for resource allocation in P2P networks. In this model, the system has no central point of control. It assigns PID (Peer ID) to each peer and random keys to each resource. The files are routed to other peers using algorithms that decide the location of the resource in the network according to the key of the resource and the PID of the nodes in the network. Consequently, the request for the resource can be routed to the destination peer without replication and broadcast. This model obtains significantly better performance at the price of reduced autonomy of the peers in regards to how resources and data about resources are handled.

2.3.1 FreeNet

FreeNet [5] is a P2P file-sharing system that provides features such as information anonymity and high security and encryption. Each FreeNet peer resides on a node that contributes storage spaces to the FreeNet network. The unique key of the file is generated using SHA-1 [15] secure hashes. There are several kinds of hash keys used within FreeNet; the main two kinds of keys used are content-hash keys (CHK) and signed-subspace keys (SSK) [3]. The content of the file to be stored is hashed to generate the CHK, ensuring the uniqueness of the keys as it is considered "nearly impossible"[3] for two different files to be hashed to the same key due to the large space a key can be picked from. Each SSK is associated with a pair consisting of a public and private key. By using SSK, there is the flexibility of letting anyone, who has the public key, read the file but only those who have the private key can perform write operations on it.

Upon joining the network, a new node first generates a public-private key pair for itself. The peer advertises its present by connecting to a remote peer that is already in the network and sending an announcement message that contains the public key, the physical address, and the TTL of the message. Once the peer received the announcement message, it randomly chooses a connection to forward the announcement message. The announce message is propagated throughout the network until the maximum TTL is reached. Then the peers that know of this new peer assign a unique random PID (also called GUID) to the new node and update their routing table.

A FreeNet peer stores knowledge about the PIDs of the nearby peers and the keys of the files contained. As each file is assigned with a unique key, upon insertion of the file into the network, the file is rerouted to a neighboring peer with a PID closest to the key of the file and is replicated for storage before it is rerouted again. The process of rerouting and replication is repeated until a user-defined number of copies have been stored in the network. Similarly, user can retrieve the file using the key of the file. When the file is being routed back for retrieval, it is being replicated and stored in the nodes along the path. Hence a frequently retrieved file tends to have a larger number of replicated files in the network and the search can be done faster. Whereas a less frequently retrieved file need a longer time for retrieval and might even be replaced by the more frequently retrieved files due to the limited storage space.

2.3.2 Pastry

Pastry [20] is a set of tools for building a P2P file-sharing system. Similar to other document routing models, each node in the Pastry network has a PID (or nodeID) and each resource in the network has a unique key. A resource is stored on a user-predefined number of nodes with PIDs that are the closest to the 128 most significant bits of the key. Each Pastry node keeps a routing table, a neighborhood set, and a leaf set. The routing table contains $\lceil log_{2^b} N \rceil$ rows with $2^b - 1$ entries. Each entry in row i contains the IP address of the other node, that matches the node's own IP address in the first i positions. The leaf set is the set of the closest PID, such that half of the PID is larger than the node's PID, and the other the half of the PID is smaller. The neighborhood set contains the PIDs and IP addresses of nodes that are closest to the local node. A Pastry node first forwards the message - that contains the key of the targeted object - to those node with PIDs, at least one digit longer than the matching number of prefixes between the key and the current node's PID, closest to the key. If this attempt failed, the message is routed to the node with a PID that shares the same number of matching prefixes with the current PID and is numerically closest to the key than the current node's ID. Using the information stored and the 2 steps described above, a Pastry node can route a message with the minimum distance/hops needed. To ensure the information of the tables are updated even after the network-changing events, such as the node arrival and departure, the routing tables are exchanged among the affected nodes.

2.4 Summary

The Centralized directory model is straightforward but lacks the robustness of the other models since it introduces a single point of failure. The flooded request model and document routing model depend largely on the network topology and the user task scheduling. User behavior such as, switching on/off the application, querying, and choosing the connection to forward the message in the case of NeuroGrid, can greatly impact the P2P network topology and consequently the efficiency and scalability of the system.

To be able to study the complex interactions of user behavior, applications, P2P protocols, physical network and machines, it is necessary to use simulators.

3. P2P Network Simulators

As mentioned earlier P2P network simulators are needed for analyzing the behavior of complex P2P systems. To date only a few P2P simulators have been implemented to aid the research on P2P-application and P2P-protocols development.

3.1 Evaluation Criteria

There are two necessary conditions for obtaining credible results from a simulator: using a valid simulation program, and executing a valid simulation experiment. "A simulation program is valid, if it is a verified computer program of a valid simulation model"[17]. A verified simulation program is a program that performs as intended. And a validated simulation model is a model that has satisfying accurate approximation of the system under study.

Consequently it is important that P2P simulators are flexible and adaptive enough for expressing the models to be used in the simulation. Table 1 contains the criteria considered important in evaluating a P2P simulator.

Criteria	Definitions
Usability	How easy is the learning for operating, preparing inputs for, and obtaining outputs of the simulator?
Extensibility	How easy can the simulator be extended to simulate modified functionality of the protocol?
Configurability (Easiness)	How much configuration power is given to the user? (And how easy is the configuration change?)
Interoperability	Can the simulator be used to interoperate with other application?
Level of Detail	How much details of the application are implemented in the simulation?
Build-ability	How easy can the application code simulated be transformed into real application?

Table 1: Criteria

3.2 Serapis

Serapis [21][22] is a java-based simulator that is designed for evaluating different caching algorithms for FreeNet. Serapis has been extended to simulate the Gnutella protocol, however, the work is halted and the extension is not yet completed (the latest update was made in November 2001). Serapis focused on the "static network designs with different connectivity patterns and routing algorithms" [8]. It has been shown that the results of simulations on FreeNet obtained using Serapis were inaccurate and that the simulator fails simulate the actual stresses and strains of a live deployment in an accurate manner [5][8].

3.3 NeuroGrid Simulator

NeuroGrid Simulator 0.1.0 [16] is a java-based P2P simulator that has been extended to support searching simulation for FreeNet, Gnutella and NeuroGrid protocols. The simulation is a single-threaded discrete event simulator. It has properties files that enable the user to modify the parameters for a simulation run. The user can specify the type of protocol to simulate, the number of searches to simulate and the type of preferred user interface (e.g. applet-based GUI). The applet displays the search messages being sent to the searching nodes at each step (see Figure 7) but tends to be useless for longer simulation runs or more complex networks.

The statistics (e.g. number of messages parsed and the states of the simulation) can be saved into files for later analysis. The NeuroGrid Simulator 0.1.0 assumes that the distance between the nodes are constant - messages with the same TTL are sent through the network in parallel. After a search message is sent out to other predefined nodes, the nodes that received the messages take turns in forwarding the messages, due to the single-threaded design of the simulator. Furthermore, it is until a search event has terminated that another new search event will be active (sequential execution of search events). NeuroGrid is still very much a work in progress an efforts are made to improve the level of detail in the network models [10]. NeuroGrid enables the user to specify the number of nodes to simulate (this is also the number of nodes to add to the current simulation after a number of searches is done), the initial number of connections for each node, the number of searches to be generated, and the initial network topology (only ring or at random networks).

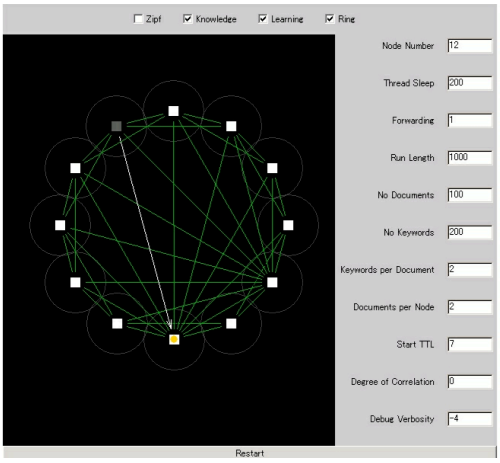

Figure 7: NeuroGrid Simulator

Since it is designed to simulate the searching algorithms of three different protocols, it is necessary to let the user specify the number of keywords used for the simulation, the number of the documents used for the simulation, the number of keywords per document, and the number of documents stored on each node (document and keyword assignments are all randomized). The latest release of NeuroGrid Simulator version 0.1.4 on December 16, 2003 has included the simulation of resource-limited nodes and the dishonest nodes. By extending the classes provided, the simulator can simulate the user-defined application on the three protocols. However, it is not widely adopted in P2P application design and development.

3.4 FreeNet Simulator

The java-based FreeNet simulator [18] supports the analysis of the FreeNet protocol, in regards to evaluating different caching algorithms. It uses a two-steps mechanism to support the event handling allowing multiple messages to be sent at a time. In the first step, the simulator will move the messages from a temporary storage space to a queue for of the node that will process them in the next iteration. In the second step, the simulator will process the messages queued at each node in the previous iteration and put the newly generated messages in the temporary storage space. With this design, all the nodes act synchronously without mixing the newly arrived messages and the old messages. The user can modify the factors for the simulation by manipulating an interface class that is implemented by the other classes. The only problem is that the source code has to be recompiled each time after the parameters of the simulation have been changed. The user can change the maximum number of nodes to simulate, the TTL of the messages, the type of nodes to be simulated (the caching algorithm), the probability of initializing the event of request for file at each node, and the probability of faulty information insertion by the node, etc.

The handshaking between nodes is not implemented in this simulator. In order to initialize a stable network, the simulator is started with three connected nodes (in a line, not a triangle).

Every five iterations a new node joins the network until the maximum number of nodes to simulate is reached. After the network is initialized with the number of nodes desired, the desired number of files will be inserted into the network. After this the simulator will be started with initiating the request events. The simulator then shows on the command prompt the statistics such as number of attempted and successful actions for file-insertions and searches. This single-threaded simulator also assumed that once the node have connected to the network, it always stay in the network.

3.5 FreePastry

FreePastry [6] is an open-source implementation of the Pastry protocol in Java. The latest release of FreePastry (released on January 28, 2003) includes the implementation of the PAST [19] archival storage system based on Pastry and an implementation of the Scribe [2] group communication infrastructure. It is an application of Pastry peer, but it can also emulate a Pastry network. It provides 3 choices of transport protocols for the user application: direct, RMI, and Wire. With direct transport protocol, FreePastry emulates a network with a user-defined number of Pastry nodes in a single Java Virtual Machine without modeling the physical network. In this situation, the main thread will setup the network and initiate the search events, as it is single threaded, the searches are done in serial as in NeuroGrid Simulator. Using RMI as a means for IPC (inter process communication),

The settings of the simulator parameters, such as the number of nodes to simulate and the number of events to generate, is done by providing the values in the command line upon starting the local simulators. The results are displayed on the command prompt screen as the messages are being processed. Since the Pastry routing uses proximity metrics, it is necessity to represent the proximity in the simulation. Random, Euclidian and sphere are currently available. In the Euclidean Network topology, the nodes are randomly placed in an Euclidean plane and the proximity is based on the Euclidean distance in the plane. Whereas, in the Sphere Network topology, the nodes are randomly placed on a sphere, and the proximity is based on the Euclidean distance on the sphere. However, the network delay for the message passing is not simulated, as the simulator is not designed to simulate time.

3.6 Summary

Current P2P Simulators do not support the customization of the initial network state (connections between the simulated computers and the network delay) and are limited in the level of detail and the scalability of the supported models. Furthermore, the simulators are mostly focusing on the caching algorithms and ignoring the fact that other activities can also impact the efficiency of the system. The NeuroGrid simulator is providing a very good network visualization using the applet, however, it does not, currently, simulate the user events, network latency and the processor delay of the nodes. Hence the simulation is not close enough to the real world situation, especially with the serial searches functionality. FreePastry and NeuroGrid executes search events only in a serial fashion and don't support the modeling of network latency and heterogeneous hardware. Due to the absence of a GUI in FreePastry the modification of parameters is cumbersome. Some of the settings are made through the command line and some have to be encoded in the program. The FreeNet simulator supports synchronous actions of nodes but fails in providing support for modeling the network latency and the user's behavior. In addition the concept of recompilation after changing is rather crude and limits the use significantly.

	NeuroGrid	FreeNet	FreePastry
Event-processing	Serial	Parallel	Serial
Usability	Very easy	Medium	Hard
Extensibility	Medium	High	Medium
Configurability (Easiness)	Mid-High (High)	Medium (Medium)	Low (Mid-Low)
Interoperability	Medium	Medium	High
Level of Detail	Medium	High	High
Build-ability	Medium	High	Very High
Simulating User behavior	No	No	No
Simulating Computer Hardware	No	No	No
Simulating network overlay	No	No	Yes
Simulating time (Network delay)	No (No)	Yes (No)	No (No)

Table 2: Evaluating existing P2P Simulators

Adapting the simulator to new or modified protocols is a question of great practical importance. NeuroGrid Simulator and FreeNet simulator did not simulate the network overlay, and hence it is hard to extend the simulation to handle new protocols that need the network proximity information, e.g. the Pastry protocol. On the other hand, though FreePastry is focused on Pastry protocol, the simulator is more decoupled and some of the code can be reused and extended to implement a simpler protocol, such as Gnutella. However, the serialized event handling and the lack of simulation on time make it hard to be extended to simulate the network delay and processor delay. Furthermore, since there is no P2P simulator that simulates the hardware of the computer, then the compute-sensitive protocol as in SETI@Home cannot be simulated by extending from the existing simulators

4. A Generic P2P Simulator

Researchers, who wanted to simulate a P2P system, tend to avoid the development of a complex simulator and focus on some selected areas (such as caching schemes). While some start an implementation from scratch an increasing number of researchers build their simulators on top of existing agent platforms like JADE [1] to speed-up the development. The general problem of having only special-purpose simulators is that the results obtained with one simulator are difficult to validate and often impossible to achieve with another simulator due to the many hard-coded assumptions of every simulator.

In this section we present an architecture and implementation of a generic P2P simulator designed to overcome the problems of existing simulators namely, extensibility, usability and level of detail.

4.1 Goals

The criteria used to evaluate the simulators in section three are used as a guideline for the design of a more generic and open simulator that will allow users to define models for the physical network, physical machines, P2P topology, P2P protocol, P2P application and user behavior.

Gnutella v0.4 is being used for the example implementation of the protocol for simulation.

4.2 Architecture

Following the desired criteria, a generic P2P simulator **(http://rila.usask.ca/~nyt431/880/Sim.zip)** is developed with the architecture shown in Figure 8. A static step-clock is used to simulate the timing. By separating the network, protocol and application from each other, the simulation of various network topologies, for different protocols, applications, and user models becomes possible. Hence, three levels have been defined:

- Network level (bottom),
- Protocol level (middle) and
- User level (top).

Communication can only happen between the directly connected levels. The Protocol level, that is responsible for simulating the protocol with desired application, acts as the interface between the User level and the Network level. Input information from the user is fed into the Network level through a GUI interface or a file.

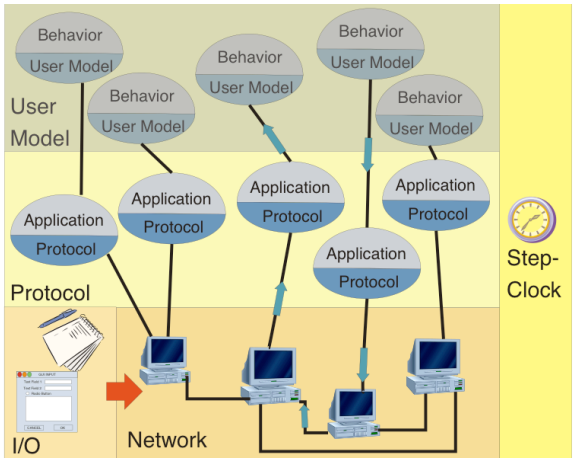

Figure 8: Architecture of the P2P Simulator

As soon as the information (total number of computer node to simulate, total number of sub-network to simulate, etc.) is fed into the simulator, the simulator will create the three levels and save them in form of serialized objects prior to the simulation run (allow for re-runs).

As the simulation is running, the events are displayed on the command prompt screen. As soon as the simulation has finished, the results (that are saved in a hash table during the running of the simulation) will be saved into a file for future analysis. According to [17], either general-purpose language (such as FORTRAN, Pascal, C and Java) or simulation language (such as GPSS, SIMAN or SLAM II) can be used for the "translation of the model into a computer program"[17]. Though simulation languages provide most of the features needed in programming a simulation model and the details of the simulation models can be easily changed, a general-purpose language was selected to provide "greater programming flexibility". Since Java is the preferred language of many P2P programmers it seems reasonable to use it as the host-language.

4.3 Network Level

Using the GUI interfaces and/or the files, the Network level creates a two-dimensional matrix storing the distance values between the nodes. The Network level is responsible for modeling the user defined aspects of a physical network deemed relevant for the simulation e.g. varying network load due to increased P2P communication.

The Network level also creates a user-defined number, of computer nodes with a user defined number of worker threads that take care of the messages passing and the processing of the messages at application node. Each node in the Network level represents a computer with user-specified hardware specification to simulate the processor delay.

Interactions between the Network level and the Protocol level are made through referencing the application node in the Protocol level obtained from its hash table.

Each node uses four queues to store the message object:

- Outbox,
- Inbox-For-Network-Delay,
- Inbox-For-Processor-Delay and
- Inbox.

When the application node sends a message object from computer node X to another application node at computer node Y the following iteration will be followed:

i. The message object, with a time-stamp, that is stored in the Outbox of the computer node X is obtained by the worker. It will deliver the message object to the Inbox-For-Network-Delay at the destination node Y.

ii. When all the workers are done with the delivery process the step-clock is incremented and the workers check the message objects in the Inbox-For-Network-Delay with the 2-D distance matrix and the congestion network delay. If the network delay has fulfilled, the message will be stored in the Inbox-For-Processor-Delay, with another time-stamp, else, it remains in the Inbox-For-Network-Delay.

iii. When the step-clock is incremented, the worker will look at the processor delay of the computer node Y and check whether the message object in the Inbox-For-Network-Delay should be moved into the Inbox for the application node to process. If the processor delay has not been reached, the message object remains in the Inbox-For-Processor-Delay.

iv. When the step-clock is incremented again, the destination port number encoded in the message object in the Inbox will be obtained by the worker,

and the reference to the application node will be obtained to send the message object to the application node for process. After the application node processed the received message object, it will check the hash table that contains the scheduled task for execution. Any created message object is stored in the Outbox and the iteration is repeated from (i) when the step clock is incremented again.

The simulator follows the 2-steps mechanism: for each unit of user-time, it takes 2 step-times in the simulation (the step clock is incremented twice). Hence, in the first step-time, iteration step ii and iv are executed. In the second step-time, iteration step i and iii are executed. With this design, the network delay and processor delay can be simulated, and there is no mixing up of the order of the message arrived at each box. And most importantly, this enables several tasks (or events) being carried out at any time.

4.4 Protocol Level

A Peer class is used to provide an interface for the workers in the network level and to enable the sending of messages from a computer node to an application node. Any protocol implementation has to implement this Peer interface class.

In the example implementation the GnutellaPeer class takes care of the Gnutella v0.4 protocol level issues, such as keeping track of the connection between the peers by keeping a table of the IP address and the port number of the directly connected peers. Though it can be used as the application node, it is not providing much for the application level. An application can be easily built by extending the protocol class by overriding the message processing methods.

Reflection is used for the creation of application nodes using data specified by the user. The user needs to specify the IP address and port number of the peers created. Upon being created, the application node/peer can use the Registration class provided by the Network level to register itself to port of the computer node using the IP address provided.

The implementation of the message object is very important in this simulator. The message object contains the time-stamp, reference to the message content object, the origin's IP address and port number, and the destination IP address and port number. An instance of the Communication class (message object) represents packet sent through the connections between the connected computer nodes. A Gnutella peer first creates an instance of the type of the message desired to be sent and then creates an instance of the class Communication to store the reference of the message instance. Then the peer sends the Communication instance as a message object using the Registration class.

4.5 User Level

Unlike the other levels the User level has not been implemented in the current version of the generic P2P simulator. A UserModel class contains the method signatures for the decision-making needed from the Protocol level. It should have a reference back to the Peer instance. Upon creation and referenced to the associated Peer, it should add the tasks into the tasks scheduler in the Peer instance. Each Peer instance should have an instance of the UserModel class, as the behavior of the user with the same UserModel can be different due to the Peer's performance. When the instance of UserModel is consulted, it should check its states and when an event is to be initiated, it should add the task into the task scheduler in the Peer instance at the protocol level by referencing the current time with the static step-clock.

4.6 Evaluation

In its current Java 1.3.1 implementation, the simulator consists of 29 classes with approximately 3599 lines of code. There are 10 classes (899 lines of code) for Platform level, 12 classes (508 lines of code) for Gnutella Protocol with user task scheduling, and 7 classes (2159 lines of code) for GUI interfaces including the main method class.

	Generic P2P Simulator
Event-processing	Parallel
Usability	Very easy (with the aid of GUI)
Extensibility	Easy
Configurability (Easiness)	Very High (Medium)
Interoperability	High (It is generic enough to connect the simulator with other application such as agents, and real application by just implementing a helper class in the protocol level.)
Level of Detail	High.
Build-ability	High.
Simulating User behavior	Yes.
Simulating Computer Hardware	Yes.
Simulating network overlay	Yes
Simulating time (Network delay)	Yes (Yes)

Table 3. Evaluation of the P2P Simulator

Table 3 shows the evaluation made using the criteria listed in Section 3.1. Figure 9, 10, 11 and 12 shows the empirical testing result obtained using two machines. The graph "For Windows" is obtained from a machine with a AMD Athlon Processor 800Mhz with 524 MB of RAM running Microsoft Windows 2000, and the graph "For Sun" is obtained using a Sun SunFire3800 with four UltraSparc III CPUs running at 750 MHz and 8 GB of RAM running Solaris 8.2. Note that the data with 225 nodes cannot be collected from the simulation running on the Sun machine due to memory leak.

The graphs show the records of the time used for object creation (to create the simulated environment and entities), saving serialized objects into file, running the simulation, and saving the results (storing the events created in the simulation) into files. For each simulation with N nodes, there is a total of (N-1)+(N-1)*(N-2) number of events (message sent from a peer to the other).

The simulator takes a longer time to create objects on the Sun machine than on the Windows platform. Hence, it is better to create the simulation environment in a Windows environment and execute the simulation on the more powerful Sun machine.

However the simulation with 225 nodes (50176 events) cannot be performed in the Sun environment due to memory limitations of the JVM under Solaris.

Memory consumption is due to the storing the events of the simulation in a hash table. While the simulator can handle more than thousand nodes if the total number of generated events is less than 50176 events, more event producing simulations require the saving of events into files to avoid memory problems.

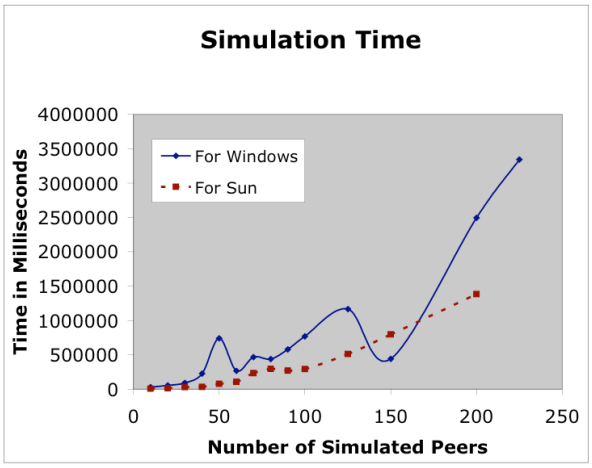

Figure 9: Time used to run the simulation for 30 step-times.

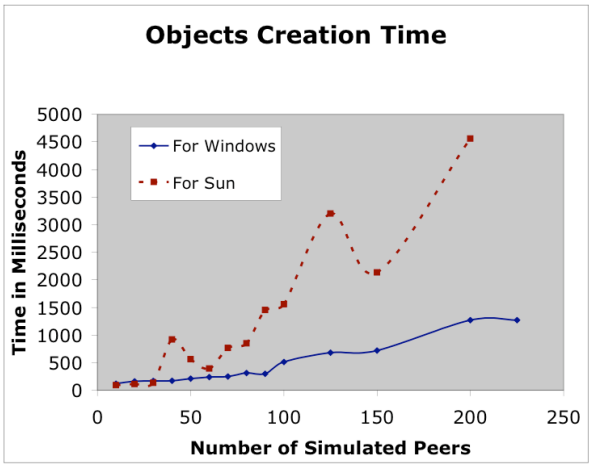

Figure 10: Time used to create the simulation objects.

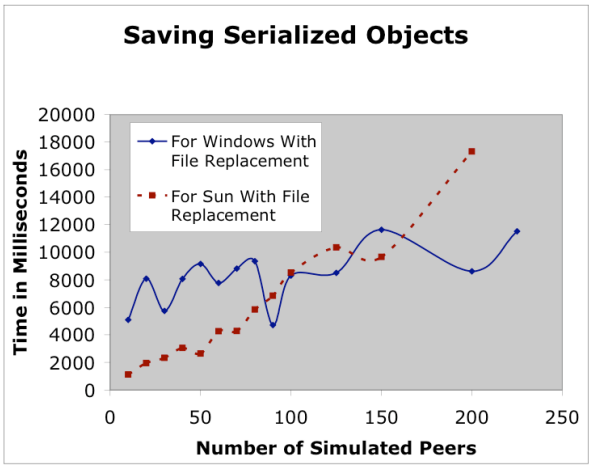

Figure 11: Time used to save the serialized objects into file.

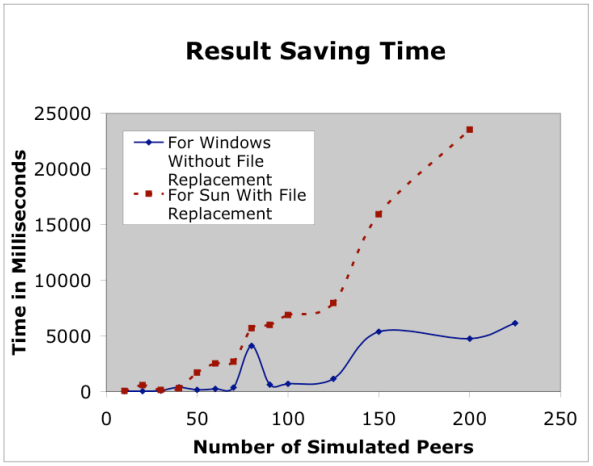

Figure 12: Time used to save the event-messages into file.

4.7 Future Work

Future work will focus on using visualization tools to create a graphical view of the events and connection in the network and adding the user model layer.

After adding the user model layer, work will focus on collecting data for the various layers e.g. human desktop usage, network traffic. To validate the results of the simulation will be compared to existing Gnutella 0.4 file-sharing applications. Especially the COMTELLA system [25] will be used for comparisons since it is a subject of an ongoing performance and usability study.

5. Conclusion

This paper provided and overview of existing P2P protocols, examines the simulators and proposes a generic P2P simulation model. The simulator enables the simulation of P2P networks with different network topology, user models, and applications.

6. REFERENCES

[1] Bellifemine, F., Poggi, A., Rimassa, G. JADE–A FIPA-compliant agent framework In Proceedings of PAAM'99, London, April 1999, 97-108.

[2] Castro, M., Druschel, P. Kermarrec A-M., Rowstron A. SCRIBE: A large-scale and decentralised application-level multicast infrastructure. IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications), to appear, 2002.

[3] Clarke, I., Miller, S. G., Hong, T. W., Sandberg, O., Wiley, B. Protecting Free Expression Online with Freenet. IEEE Internet Computing 6(1), 40-49 (2000).

http://freenetproject.org/twiki/Main/Papers/ieee-final.pdf

[4] Clip2. The Gnutella Protocol Specification v0.4.

http://rfc-gnutella.sourceforge.net/Development/GnutellaProtocol0_4-rev1_2.pdf

[5] FreeNet. The FreeNet homepage, freenet.sourceforge.net.

http://freenetproject.org/cgi-bin/twiki/view/Main/WhatIs

[6] FreePastry. The FreePastry homepage.

http://www.cs.rice.edu/CS/Systems/Pastry/FreePastry/

[7] Fried, I. Intel chip will be bigger, more expensive to make. Cnet News.com, June 2002.

http://news.com.com/2100-1001-241376.html?tag=mainstry

[8] Joseph, S.R.H. Adaptive Routing in Distributed Decentralized Systems: NeuroGrid, Gnutella, and Freenet. Proceedings of workshop on Infrastructure for Agents, MAS, and Scalable MAS, at Autonomous Agents, Montreal, Canada, 2001.

[9] Joseph, S.R.H. NeuroGrid Protocol.

http://www.neurogrid.net/ng-protocol.html.

[10] Joseph, S.R,H. Project:NeuroGrid -P2P Bookmark Organiser:Mailing Lists. NeuroGrid Simulation Mailing Archive, Jan 2003

http://sourceforge.net/mailarchive/forum.php?thread_id=1593250&forum_id=8271

[11] Jovanovic, M. A. Modeling Large-scale Peer-to-Peer Networks and a Case Study of Gnutella. M. Sc. Thesis, University of Cincinnati, 2001.

[12] Klingberg, T., Manfredi, R. Gnutella 0.6.

http://rfc-gnutella.sourceforge.net/draft.txt

[13] Milojicic, D. S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne J., and Richard, B. Peer-to-Peer Computing. Internal Report HP, March 8.

http://www.hpl.hp.com/techreports/2002/HPL-2002-57.html

[14] Napster. The Napster homepage.

http://www.napster.com

[15] National Institute of Standards and Technology. Secure Hash Standard (SHS).

http://csrc.nist.gov/cryptval/shs.html

[16] NeuroGrid. The NeuroGrid homepage.

http://www.neurogrid.net/

[17] Pawlikowski, K. Simulation Modeling and Analysis with an emphasis on applications in performance evaluation of telecommunication networks. Course 410, University of Canterbury, August 2002.

http://www.cosc.canterbury.ac.nz/teaching/handouts/cosc410/02.410.notes1.pdf

[18] Pfeifer, J. Freenet Caching Algorithms Under High Load.

http://www.cs.usask.ca/classes/498/t1/898/W7/P2/freenet.pdf

[19] Rowstron, A., Druschel, P. PAST: A large-scale, persistent peer-to-peer storage utility. HotOS VIII, Schoss Elmau, Germany, May 2001.

[20] Rowstron, A., Druschel, P. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems IFIP/ACM international Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, 329-350, November 2001.

http://research.microsoft.com/~antr/PAST/pastry.pdf

[21] Sandberg, O. The FreeNet-dev mailing list, March 2001

http://www.ultraviolet.org/mail-archives/freenet-chat.2001/0354.html

[22] Serapis. The Serapis homepage, cvs. Sourceforge.net.

http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/freenet/Serapis/

[23] SETI@Home. The SETI@Home homepage.

http://setiathome.ssl.berkeley.edu/

[24] Single, A., Rohrs, C. Ultrapeers: Another Step Towards Gnutella Scalability.

http://www.limewire.com/developer/Ultrapeers.html

[25] Vassileva, J. Motivating participation in Peer to Peer Communities. Proceeding of Workshop on Emergent Societies in the Agent World, ESAW'02, Madrid, 16-17 September, 2002.

http://www.ai.univie.ac.at/%7Epaolo/conf/esaw02/preproc/E0029.pdf