Enhanced Reconfigurable Gnutella (ERG): Dynamic Performance Improvement of Gnutella Networks

Evangelos Pournaras

Submitted for the Degree of
Master of Science
from the
University of Surrey

Department of Computing
Faculty of Engineering and Physical Sciences
University of Surrey
Guildford, Surrey, GU2 7XH, UK

August 2007

Supervised by: Dr Nick Antonopoulos

You told me once and you were crying…

"Don't change where you are going…"

Finally, I have changed! But you smile…

Thank you for everything you have done for me!

ACKNOWLEDGMENTS

I would like to thank my supervisor Dr Nick Antonopoulos, who trusted me, believed in me and my abilities and he gave me the opportunity to work on this project, something that was challenge for me both in research and application level. I would also like to thank him for making me believe in P2P networks and their capabilities as one of the ultimate research challenge in computing nowadays. I could specially mention the benefits I received by his high education, technical knowledge and motivation for completing this work. His methods, collaboration, availability and advices helped significantly towards the success of this work.

I would also like to thank Mr George Exarchakos, for his invaluable advices and his patience to come to the morning meetings with our supervisor and share his ideas with us. He also motivated me to work on P2P networks and inspired me many times, even with simple words; crazy ideas cross my mind and solve problems with them.

Last but not least, I would like to thank my family for their passion to support me and make me smile. I owe them everything!

ABSTRACT

P2P networks have gained a considerable research interest in the last years. Resource discovery and heterogeneity of the overlay P2P networks cause overloading problems and especially in unstructured networks which use flooding as the searching technique. The purpose of this work is the development of a load balancing model, which shares capacity between overloaded and underloaded nodes in a Gnutella network. The load which this work focuses on is the query load. The idea is based on some logical movements of the nodes that cause the overloading to nodes that are underutilized. These movements are supported by special peers called virtual servers. Except the background provided on unstructured P2P networks with emphasis on Gnutella networks, autonomic computing and self-organised systems are also discussed. This work describes in detail the proposed models and the simulation environment for testing the various metrics of the network. Some of them include the number of discarded messaged, system availability, query success, traffic caused by load balancing, load profiles of nodes in the network, standard deviation of query rate in nodes and more. Simulation results show a great performance improvement, with the model increases query success, reduces discarded messages, increases the balanced nodes and all these with exponentially reduced cost in load balancing traffic. Furthermore, system behaviour has characteristics of self-organisation and analysis, is illustrated in this issue. Lastly, the work concludes with some future ideas and extension on the proposed model.

ABBREVIATIONS

P2P	Peer-to-Peer
ERG	Enhanced Reconfigurable Gnutella
QoS	Quality of Service
DHT	Distributed Hash Tables
MD5	Message Digest Algorithm 5
TTL	Time to Live
VSS	Virtual Server Supervised
VSD	Virtual Server Driven
RFH	Request For Help
RTUL	Request To UnderLoaded
RTOL	Response To OverLoaded
UAA	Unregister And Assign
CTOL	Confirm To OverLoaded
JDSL	Java Data Structure Language
GUI	Graphical User Interface
CPU	Central Processor Unit
DoS	Denial of Service
API	Application Programming Interface
UIP	User Interface and Parameterization
EP	ERG Protocol
SE	Simulator Engine
PN	P2P Network

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

For most people, P2P systems correspond to file-sharing networking environments. The truth is that P2P systems are one of the most multidimensional topics, both in research and applications. The facts for P2P future are very promising but there are existing problems that must be solved. The problems vary, depending on the type of P2P network. Unstructured and structured schemes can have different performance and trade-off issues.

This work studies load balancing issues in unstructured P2P networks and specifically in Gnutella (see Gnutella 0.4 and 0.6 specifications). Gnutella network is a fully decentralized system, which, however, suffers from scalability problems and high bandwidth consumption due to flooding. One of the major problems and challenges in Gnutella network is the high number of queries generated and transferred. It is believed that they may cause problems to some nodes to respond to these queries or forward them further. This has a negative effect in query success and in the overall performance of the network. Furthermore, an unstructured distributed system like Gnutella lacks resources control. The network may have unexploited resources and there is not any mechanism to take advantage of the network dynamics.

The above problems do not appear only to Gnutella. However, studies reveal these problems being more intense in Gnutella network. In general, the solution to such problems and in such systems, is the realization of smart mechanisms, which provide a potential functionality to manage their resources, their configuration and their protection. Such systems appear adaptivity under different conditions and are able to be self-organised. The whole effort is towards autonomic computing, which defines self-managed systems.

The purpose of this work is the development of such a smart, dynamic and adaptive mechanism to solve load balancing issues in Gnutella networks. The study focuses on Gnutella 0.4. The proposed model tries to support overloading nodes from queries by establishing a "logical movement" of the nodes that cause the overloading to another node that is regarded underloaded. For achieving these actions, the virtual server role is attributed to nodes with high performance profile to support the load balancing model.

This work is outlined in five parts. Part 1 provides some background knowledge of P2P networks and focuses on unstructured P2P and Gnutella. It also introduces the idea of autonomic computing and self-organised systems and associates the concepts with P2P computing. Part 2 describes the two proposed models and two variations for further investigation. Part 3 describes the developed simulator, its architecture, the technologies used and implementation issues. Part 4 outlines the results and discusses the findings of the simulations. This work concludes with Part 5, which illustrates the conclusions and discusses ideas for further work.

1. LITERATURE REVIEW

P2P systems have gained considerable research interest in the last few years. They incorporate the distributed concept in communication and self-organisation. Although the idea of P2P systems is not new, they have been utilized within the last few years because of some specific technological improvements. Nowadays over 50% and sometimes over 70% of Internet traffic is attributed to P2P systems according to Lu, Cao, Cohen, Li, and Shenker (2000). This fact shows the tendency of increased requirements in bandwidth.

What was regarded a P2P system in the past is quite different form what we consider nowadays. According to Steinmentz and Wehrle (2004), *a P2P system is a self-organizing system of equal, autonomous entities (peers) which aim for the shared usage of distributed resources in a networked environment avoiding central services*.

In a P2P network each node is treated equally and the communication is established directly without any intermediate centralized machine. Three main requirements of the future Internet-based applications can be identified:

- Scalability.
- Security and reliability.
- Flexibility and QoS.

Researchers focus on providing to P2P systems the above attributes which will offer more complete solutions and make P2P computing the dominant networking scheme in a wide range of applications.

The next subsections outline the P2P systems and challenges by focusing on unstructured schemes. It will also be illustrated here some background knowledge on graph theory and description will focus on Gnutella networks which are the foundation of the developed load balancing system.

1.1 Peer-to-peer Systems and Challenges

One common approach on the development of a new technology is the need it will serve. In P2P systems we have a wide range of needs (applications), with different and sometimes contradictive properties. The fundamental need is the realization of a decentralized self-organised system for achieving high level of QoS without the need of centralized services. There are two approaches which have been introduced:

- Unstructured P2P Systems.

These systems were the first P2P schemes. There is not a standard structure in the connections between the nodes. The first unstructured networks (and their applications like Napster - see Napster web site) were based on hybrid models. In these systems, users looked up to a central point - server and the last responded with the location of the resources. Although data transfer is direct, resource discovery is highly dependent on these servers, which have the common disadvantages of client - server model.

Other approaches have managed to offer complete distributed networks with other disadvantages which concern memory, processing power and bandwidth consumption (Gnutella with flooding, Ritter (2001), Zeinalipou (2002)).

Despite their still existent problems, unstructured P2P systems remain a useful scheme for further improvements and development of distributed models.

- Structured P2P Systems.

Structured P2P systems are based on Distributed Hash Tables (DHTs), which offer scalability, reliability and fault tolerance. DHTs provide content-addressable data storage and look up of a resource has $O(\log N)$ time complexity. A characteristic analysis is provided by Balakrishman, Kaashoek, Krger, Morris and Stoica (2003). Upon when the network forms the hash tables, it can grow arbitrarily without impact on efficiency. Sometimes the whole effort is towards retaining the hash table consistence and upon satisfaction of this condition; a significant improvement can be noticed compared to the unstructured systems.

Some structured systems include Pastry, described in Rowstron and Druschel (2001), CAN, described in Ratnasamy, Francis, Handley, Karp and Shenker (2001), and Chord, described in Stoica, Morris, Karger, Kaashoek and Balakrishnan (2001).

P2P systems are nowadays able to provide various solutions and there are several applications, which most of them concern file sharing. However, systems require establishing intelligent and more efficient mechanisms. One fundamental topic is the optimization and interconnection, in an efficient way, of the overlay and the underlying layers. Projects like Eichhorn (2006), put some effort towards the solution of this problem. Furthermore, P2P networks need more sophisticated searching techniques, which will be able to provide scalability and success in resource discovery. Another aspect is the load balancing. Almost all of the P2P networks deal with overloading issues in many operations, (storing or processing information). Mobile and wireless computing and establishment of P2P scheme seems an interesting idea but modifications are needed and adaptivity to the characteristics of these networks and their applications. Networks also require smarter behaviour. This introduces the property of autonomy, self-organisation and self-management, towards autonomic computing as it has been envisioned by IBM (2001), Kephart and Chess (2003).

1.2 Unstructured Peer-to-peer Systems

Unstructured P2P networks are an interesting and challenging networking scheme. There are still problems which have not been solved and many obstacles must be overcome.

Before the idea of the model proposed is introduced, some background knowledge on graph theory and Gnutella networks is important for understanding the definition of the problem and the proposed solution described in the next section.

1.2.1 Graph Theory for P2P Networks

The fundamental concept of P2P networks is the graph (see Fan and Chung (1997)), in which nodes are connected directionally or bidirectionally. There are two famous network models which can correspond to unstructured P2P Networks:

- Small-World model, see Watts and Strogatz (1998).
- Scale free networks, see Barabasi and Albert (1999).

If we consider again the nature of P2P networks and the conditions which form them, they have common characteristics with social networks. A social network is inherently self-organised but it does not follow a certain structure. Such a network remains structurally stable and is evolved, despite the fact that people come and leave, live and die. Furthermore, there are not a constant number of participants. This stability and evolution are characteristics described by small-world effect and scale free degree distribution properties, which also correspond to the two above network models. We can describe the network mathematically as below:

- $V = \{1,2,3,...,n\}$ is a set of n nodes.
- $N(\upsilon)$ is the set of neighbours of node υ.
- $G = (V, E)$, where G is the graph (overlay network) and E is the set of edges .
- $e = (i, j)$ with j being the neighbour of i (target and source nodes respectively).
- m , the number of edges.
- $A(G)$ is the adjacency matrix with dimensions $n \times n$.
- $k_o(\upsilon) = |N(\upsilon)|$ in the number of neighbours the node υ has.
- $k_i(\upsilon) = \sum_{\omega \subset V} [\upsilon \subset N(\omega)]$ is the number of neighbour sets in which υ is an element (indegree).
- $k(\upsilon)$ is the sum of indegree and outdegree.

- $P(i,j)$ is the path from node i to node j as a subset $P \subseteq E$ of edges $\{e_1, e_2, ..., e_k\}$ where $e_1 = (i, \upsilon_1), e_k = (\upsilon_{k-1}, j)$ and $\forall \ 1 < l < k : e_l = (\upsilon_{l-1}, l)$. The path length is defined as the number of edges in $P(i,j)$.

- $d(i,j)$ is the shortest path between nodes i and j.

- $D(G) = \max\limits_{(i,j) \in V \times V} d(i,j)$ is the diameter of a graph G (the maximum distance between any two nodes in the graph).

- $D_{avg}(G) = \dfrac{\sum_{(i,j) \in V \times V} d(i,j)}{n \cdot (n-1)}$ is the average path length of a graph G.

The simplest structure, which the entire above network properties can describe, is the random graph, described in the work of Aiello, Chung, and Lu (2000). If the objective is a degree of control in the diameter of the graph and a clustered structure, then small-world model can be introduced. This model defines the clustering coefficient which can provide more structured networks (chordal rings). Lastly, scale-free networks show a property of self-organisation and follow statistical regularities.

1.2.2 Gnutella 0.4

Gnutella 0.4, (see Gnutella 0.4 specification) is a fully decentralized network which consists of a large number of nodes connected arbitrarily. The node degree distribution is described as:

$$p(d) = \begin{cases} c \cdot d^{-1.4}, 0 < d < 7 \\ 0 \end{cases}, c = \left(\sum_d \frac{p(d)}{c} \right)^{-1} \qquad (1.1)$$

The degree d can range from one to seven with average $\overline{d} = 2.2$ and $\text{var}(d) = 1.63$. A node can enter Gnutella network by establishing an average of 3 TCP-connections to other active Gnutella nodes, whose IP addresses can be received from a bootstrap server. Messages in Gnutella are transmitted in plain text. Uniqueness is satisfied through MD5 hash keys, defined in Rivest (1992). Flooding is used as the searching technique, in which messages are sent to all neighbours except to the one from which it was received. The procedure is recursive until a Time-to-Live (TTL) value which is, by default, seven. If a node has already received a message, it is not forwarded further.

Gnutella messages have a common descriptor header which describes information important for every message transmitted in the overlay network. Descriptor header has the following structure:

Table 1: Descriptor Header

Descriptor ID	Payload Descriptor	TTL	Hops	Payload Length	Payload n Bytes

Payload descriptor defines the following Gnutella messages:

- **Ping (**0×00**)**: is used for nodes discovery.
- **Pong (**0×01**)**: is the response to a Ping message.
- **Push (**0×40**)**: is used for downloading from firewalled nodes.
- **Query (**0×80**)**: is the message initiating the search information from the generating node.
- **Query Hit (**0×81**)**: contains information about the discovered resources.

The respective fields of the above messages are illustrated below:

Table 2: Ping message. A simple descriptor header.

Descriptor ID	Payload Descriptor	TTL	Hops	Payload Length	Payload n Bytes

Table 3: Pong Message

Descriptor Header	Port	IP Address	Number of Shared Files	Number of Kilobytes Shared

Table 4: Push Message

Descriptor Header	Servent Identifier	File Index	IP Address	Port

Table 5: Query Message

Descriptor Header	Minimum Speed	Search Criteria

Table 6: Query Hit Message

Descriptor Header	Number of Hits	Port	IP Address	Speed	Result Set	Node ID

1.2.3 Gnutella 0.6

Gnutella 0.6 (see Gnutella 0.6 specification) is a hybrid model which creates a hub-based network by introducing another dynamic hierarchical layer. In this way, reduction of bandwidth consumption is achieved through the hierarchies of *superpeers* and *leafnodes*, their importance is outlined in Singla and Rohrs (2002). Each superpeer should not have registered more than 50-100 leafnodes depending on the processing power and the network connections. The number of superpeers increases according to the number of leafnodes. The node degree distribution in this network is:

$$p(d) = \begin{cases} c \cdot d^{-1.4}, 1 < d \leq 7 \\ c \cdot 1^{-1.4} - 0.05, d = 1 \\ c \cdot 0.05, d = 20 \\ 0 \end{cases}, c = \left(\sum_{d} \frac{p(d)}{c} \right)^{-1} \tag{1.2}$$

The average degree is $\overline{d} = 2.8$ and $\text{var}(d) = 3.55$. Although the messages follow the specification of Gnutella 0.4 network, superpeers must keep track of the content of leafnodes. They use these tracks to announce later any potential contents which leafnodes may request. The information can be sent in two ways. The first sends all the table information to superpeer, whereas the second sends small chunks (maximum 255), subset of the whole table. Figures 7 and 8 illustrate the two messages:

Table 7: Route Table Update (variant 0×0)

Variant	Table Length	Infinity

Table 8: Route Table Update (variant 0×1)

Variant	Sequence Number	Sequence Size	Compressor	Entry Bits	Data

Variant field specifies if the message resets or updates. Infinity field was intended to clear the route-table if it was broadcasted in the network. Sequence number and sequence size specify the chunk characteristics and compressor states the compression scheme for the route table (0×0 for no algorithm and 0×1 for ZLIB algorithm).

1.3 Load Balancing in P2P Networks

In client-server model, the server was the one which should manage the high workload caused by the clients. Usually, servers are machines with high capabilities that can serve a high number of clients without problems. In practice this is infeasible. Even in the case when the machine is capable of handling the workload, the cost is extremely high. Mainframes cost millions of dollars and even in this case there must be guaranteed solutions in cases of single point of failures and overloading. The solution to these problems is the introduction of more highly capable machines which will be able to support cases of emergence. This increases the cost even more and also introduces maintenance and management issues.

On the other hand, P2P models do not suffer from single point of failures but there are overloading issues. Although these problems may not lead to complete unavailability of services, usually they reduce performance of systems and exploitation of resources. Solutions of overloading nodes in P2P world can not be faced in the same way as in client-server model. The load must be distributed equally as all peers are equal in a P2P network. Equally means distribution according to the capabilities of nodes. This is one of the most serious challenges in P2P systems. Many of them suffer from extra load and developing algorithms and models for balancing the load is a research topic of extreme importance.

In addition, workload is a multidimensional concept in P2P systems. It may concern anything, varying from simple messages, storage, CPU cycles, connections, memory caching, bandwidth and many more. The degree of adaptivity and tolerance depends on the application and the type of network. A solution which can associate the causes and handle all of them as a holistic solution seems the ultimate challenge.

There are various problems that have been solved partially with solutions having advantages and disadvantages. Research has been focused mostly on structured P2P networks. In Karger, and Ruhl (2006), the problem of uneven partitioning of the address space and items in DHTs is addressed. According to the authors, some machines may receive more keys during mapping by a factor of $O(\log n)$ the average. Furthermore, even if the address space is evenly distributed the uneven distribution of keys may cause overloading. The algorithms are based on the idea of virtual nodes which form a real node. Then distribution of keys is based on the Markovian properties and the algorithm achieves $O(\log n)$ degree per real node, $O(\log \log n)$ look up hops and constant factor load balancing. As far as item balancing is concerned, the algorithm is based on movements of underloaded nodes to portion of address space occupied by large number of items.

In another work for structured P2P networks, see Godfrey, Lakshminarayanan, Surana, Karp and Stoica, (2006), the address space problem with invariable number of nodes inserting and leaving the network and also the number of stored items are solved in a slightly different way. This work introduces the idea of virtual servers which are responsible for storing data and tracking routing information. The virtual servers are moved from overloaded to underloaded nodes. The algorithm is based on two conflicting goals. The minimization of load imbalance and the minimization of load moved. The goals can be achieved by applying two operations. The first remaps objects to different points in ID space and the second changes the regions that are associated with a node. The authors prove that the latter is the recommended because items can be queried by IDs and each ID can be produced by hashing the content. The algorithm is able not only to act on request but also to predict potential overloading. It also predicts changes in load and the effects affect multiple resources because actions are based on a load vector. Lastly, the authors support that replication is not a good solution because it introduces overhead, immutable data and complex algorithms in order they support the maintenance of data consistency. Similar technique for load balancing is described in Zhu and Hu (2005).

A completely different load balancing scheme is proposed in Bienkowski, Korzeniowski and Heide (2005). The idea is based on balancing the length of intervals in which a node can enter or leave the network. In Aberer, Datta and Hauswirth (2005), attention is focused on two conflicting load balancing issues, concerning storage and replication. In the proposed algorithms peers dynamically change their associated key space (bin adaptation) decoupled from their unique and stable identifier. The routing between peers is based on the associated key space rather than on the peer identifier. The algorithm also splits key space in partitions and assigns peers in multiple partitions.

More complete solutions in load balancing are proposed in Exarchakos, Salter and Antonopoulos (2006). G-ROME protocol aims to balance multiple independent DHT-based P2P networks for transferring capacity from underutilized nodes to overutilized. The protocol is based on ROME protocol which balances ring-based P2P topologies for reducing the number of failed queries. The communication among multiple independent P2P networks is achieved by creating a Gnutella-like interconnection.

Although most research focus on structured networks, there are several approaches for unstructured. The issue seems more complicated because such networks have unpredictable behaviour. In Suri, Toth and Zhou (2004), the load balancing focuses on downloading and replication of data for downloading from multiple resources. A peer that shares its resources looks for optimization to avoid high bandwidth consumption. The uncoordinated load balancing scheme imitates the interactions among self-interested agents which are modeled as a game. In this scheme each node has its own objective and the outcomes are described by Nash equilibria, Papadimitriou (2001). This direction introduces in the models a price of anarchy which is described by the worst case ratio between a Nash outcome and the social optimum. The greedy algorithm is also described which is based on the myopic strategy. In practice this means that a peer will select the server with the lower latency.

Load balancing can also be inspired by other fields and be supported mathematically through various disciplines. In Uchida, Ohnishi, Ichikawa, Tsuru and Oie (2006), a dynamic storage load balancing technique is described inspired from physics and specifically thermal diffusion.

The above, related to load balancing works, reveal the diversity of approaches and different strategies and objectives towards different types of load balancing. More research is needed which will bridge the gaps between the different conflicting objectives.

1.4 Autonomic P2P Computing – Self-Organisation

The evolution of computing shows a constant effort of making and offering faster and more powerful infrastructures. Nowadays, such an approach in modern computing systems is not enough and can not provide and support the services in order to satisfy users' needs. P2P computing can not be integrated to an advanced networking paradigm if it lacks some specific characteristics. Systems like this need smart mechanisms, adaptivity, and self-management. Computing is multidimensional, including different technologies, different devices, and a wide range of services. The increasing number of interactions and relationships between all the concepts mentioned raises management issues. There is a whole exponentially increasing complexity, and it must be reduced and handled automatically. Management must be part of the systems and support a variety of other services.

The effort is towards autonomic computing in which design and implementation of computer systems, software, storage, and support must be characterised, from user perspective, by flexibility, accessibility and transparency.

According to IBM (2001), there are short and long term benefits from the realization of autonomic computing. They are outlined below:

- Short-term technical benefits:
    - Simplified user experience through a more responsive real-time systems.
    - Cost-savings – scale to use.
    - Scaled power, storage and costs that optimize usage across both hardware and software.
    - Utilization of underused processing power.

  - Better descriptive queries languages which will be more natural and accurate.
  - Seamless access to multiple file types. Store data from various resources on-the-fly based on re-formatting and open standards.
  - Self-healing which will provide stability, high availability, and security with fewer errors.

- Long-term business benefits:
  - Enablement by shifting available resources to higher-order business.
  - Autonomic capabilities and autonomic federated systems.
  - End-to-end service level management.
  - Distributed computing will allow higher level of collaboration and complex problems solving.
  - Massive and time consuming simulations.

1.4.1 Defining autonomic computing – elements and characteristics

There is not a unified and concrete definition of autonomic computing as it has not been realised yet. Most definitions are based on descriptions. One very simple descriptive definition according to webopedia (2007) is the following:

”*Autonomic computing is a type of computing model in which the system is self-healing, self-confugured, self-protected and self-managed*.”

The important concept of an autonomic system is the “self-actions” which the definition mentions. Although many other properties have been introduced, for example self-anticipation, self-adjustment, self-criticality, self-government and so forth, as it is mentioned in Sterritt and Hinchey (2005), some explanations of the main four and a contrast with the current systems is illustrated below following the thoughts of Kephart and Chess (2003):

- **Self-configuration**: Configuring, installing or integrating systems which provide and incorporate a large amount of data, is a time-consuming and challenging task for system administrators. Most of them must solve problems of incompatible systems between different platforms and of different vendors. In an autonomic system, configuration works automatically and follows high-level policies which fulfill business objectives.

- **Self-optimization**: There are nowadays numerous software systems that need complicated configuration in order they are optimized. These systems are more difficult to be managed when they interact with other similarly complicated software systems. Autonomic systems could support their own functionality in order they improve their operation and be benefited in performance or cost.
- **Self-healing**: Computing systems nowadays incorporate an extremely high cost for maintenance and error correction. Developers spend a lot of effort and time trying to fix problems, restoring data and maintaining the consistency of systems. This fact comes at a high price for businesses. Taking advantage from self-configuration mechanisms, systems can monitor their state, predict dangerous operations and find fast and reliable solutions for failures.

- **Self-protection**: Most systems, although they incorporate functions for self-protection, these are mostly managed by the user who must decide how he will protect his system. Autonomic systems would be able to protect themselves against large-scale correlated problems and cascading failures as a whole and also take actions for avoidance.

There are eight elements and characteristics that describe and outline the autonomic computing paradigm according to IBM (2001). These elements come with examples in a P2P autonomic system:

- The system must be self-aware of its components and its attributes. This means that autonomic systems must keep the status of its state and the components it interacts, and set any limitations and potential extensions. In a P2P system this could mean every node to be aware of the load capacity it can handle, how the limitations have been set and which the shared or isolated resources are.

- An autonomic computing system must be reconfigurable under a varying and unpredictable environment. Any configurations must occur automatically and also the properties of the configurations must be produced automatically. In a P2P network, trade-offs between network resource consumption and searching resource success should be handled under reconfigurable mechanisms which would consider the heterogeneity of the network and the users' demands.

- An autonomic system should monitor its state and always find ways to optimize its functions. In a P2P network nodes may keep statistics about the searching success of the path that they choose to forward the searching messages. Each time the optimum path must be selected.

- Such systems must find ways to recover and be adapted to situations of failures or extraordinary events. Under such extreme conditions the system must find alternative solutions and reconfigure its elements. In a P2P system, nodes are not always connected. In cases of failures, the nodes must find alternative resources and always under transparency to the user.

- An autonomic system will exist in unreliable and unstable networking environments. Security, integrity and robustness should be satisfied and maintained through self-protection techniques. P2P systems face many security challenges. Lack of centralised entities creates a more unstable security context and the challenge is to satisfy the requirements over distributed mechanisms.

- The system must be adaptive. This requires knowledge of the environment, its activity and the parameters that can influence the different outputs. Furthermore, advanced interactions must be applied between the neighbouring entities and between different systems and sub-systems. Load balancing in P2P networks is an adaptive property because it allows the system to work more efficiently and exploit underutilized resources or, act towards supporting overloaded areas.

- Although an autonomic system is a self-managed and independent system, it must be able to interact with different systems, support open standards and be oriented to interoperability. This means that proprietary solutions are difficult to realise autonomic computing. In P2P world, proprietary systems are oriented to one specific application and most of them can not work and interact with other systems. Furthermore the systems are close to the public and these restrictions can not form an autonomic P2P system.

- An autonomic system should be adaptable and optimized and at the same time fulfilling the business and user requirements. This goal should be achieved by hiding the details to the user and providing a transparent autonomic functionality. The whole effort is to reduce the complexity of the system. In a P2P system, users may request different QoS. These services should be allowed by the business policies and the whole system configuration should be provided to the user transparently.

1.4.2 The challenges in Autonomic Computing

The vision of autonomic computing is not the solution of a single problem but rather a change of view and philosophy to computing. The difficulties come from the interdisciplinary scientific collaborations and business environments which must support the whole idea. Autonomic computing is based on the idea of holistic approach, in which there are not the improvements on single machines that will benefit and bring greater advantages to users but rather the adaptation of open standards and new technologies which will allow systems to interact more effectively and be able to fulfill business policies. These systems must be able to protect themselves, recover from failures and be reconfigurable by always having the minimum dependence on traditional I/T support. This idea forms a system with different conceptual foundations:

- The transition from dependence on computational power to dependence to data.

- The change of estimating computing performance from CPU speed to immediacy of response.

- The importance of dispersed computing attributes will become more important than individual computers.

- The business part of computing concerning the usage reflection will be attributed to the term e-sourcing.

If there is a view from individual systems perspective, then these are systems with:

- Scalable storage and processing power which must be able to serve the needs of both individual and multiple autonomic systems.
- Transparent routing and formatted data over variable devices.
- Chips with better memory management and utilization.
- Monitoring functionality which will offer a high degree of security and decision-making.
- Smarter and more robust microprocessors.

The concept of autonomic computing is wide. P2P networks seem an interesting paradigm to study the formation of autonomic computing. Solving the various problems in P2P systems requires the form of well defined structures. P2P networks must have a high degree of self-organisation in order to support the vision of autonomic P2P computing. Self-organisation is an issue of extreme importance. In the next sections some of the concepts are outlined and their associations with P2P systems.

1.4.3 Self-Organisation and Associations in Peer-to-Peer Networks

Self-organisation is strongly associated with autonomic computing. Self-organised systems are characterised by autonomy, self-maintenance, self-optimised, adaptivity, rearrangement, reproduction and emergence. These characteristics can be also attributed to autonomic systems and the concepts follow the same pattern. However; there is a historical background to the idea of self-organisation which does not exist in autonomic computing. Aristotle (1957) stated that "*The whole is more than the sum of its parts*" which describes an attribute of nowadays self-organised systems called *emergence*.

Self-organisation in computing has been inspired by multi-discipline fields and it seems that it can be modeled by following various natural phenomenons. In Biology the term *autopoiesis* describes a way of organisation in which every living organism seems to exhibit, view of Maturana and Varela (1980). Similarly there is a new field in the P2P world, the autopoietic P2P networks, which have a purpose to model complex relationships from organised systems in nature (e.g. flying birds). Similar systems can be studied in physics and chemistry.

An important aspect of self-organised systems is how they are described and are identified. Several attributes and definitions support the modeling of these systems and they are outlined below according to Steinmetz and Wehrle (2005):

- **System**: "*A system is a set of components that have relations between each other and forms a unified whole. A system distinguishes itself from its environment*".

  A computer network is an example of a system as defined above. It has components and relationships between them (computers and connections respectively) and forms a unified whole. A network can be seen as an individual entity which can interact with other entities (networks).

- **Complexity**: "*The term complexity denotes the existence of system properties that make it difficult to describe the semantics of a system's overall behaviour in an arbitrary language, even if complete information about the components and interactions is known, Bar-Yam (1997)".*

  Systems with high complexity have various properties that describe their behaviour. If we manage to change the behaviour of systems, the properties will change and this may lead to reduced complexity.

- **Feedback**: "*Feedback describes the return to the input of a part of the output of a machine, system, or process".*

  Feedback can lead to effects that do not have a proportional relationship with the causes. The effects may amplify or attenuate the external influences the system receives depending to the positive or negative feedback respectively.

- **Emergence**: "*Emergence refers to unexpected global system properties, not present in any of the individual subsystems, which emerge from components interactions*".

  Emergent behaviour can be noticed through nature in various biological systems like ant colonies and also to computing systems like some software agents and neural networks. In such systems, the final output is unpredictable if it is based on the distinct behaviour of individual elements. The properties of these systems are influenced by the various parts but they are not contained in any of them.

- **Complex System**: "*Complex systems are systems with multiple interacting components whose behaviour can not be simply inferred from the behaviour of the components*".

  This definition imposes that the elements and their relationships emerge unpredictable behaviour and patterns. So, emergency is a necessary property of complex systems.

- **Criticality**: "*An assembly in which a chain reaction is responsible is called critical and is said to have obtained criticality* ".

  Criticality refers to the degree of order. A system may vary from total order to complete disorder. In the first case, all relations are structured homogenously and are stable without any unpredictable behaviour. In the second case, the system is characterised by persistent stability and usually can be described by stochastic methods. *Subcritical* are the systems with a low degree of order and *supercritical* the systems which are largely structured. Criticality lies between these two states and is associated with stability of system. Systems which have the tendency to appear in a state of criticality without external influences are called *self-organised critical systems*.

- **Hierarchy and Heterarchy**: "*Hierarchy is defined as a rooted tree, which is an undirected simple graph G, satisfying the conditions that any two vertices in G can be connected by a unique simple path. A tree is called a rooted tree if one vertex has been designated the root, in which case the edges have a natural orientation, towards or away from the root. Heterarchy is a type of network structure that allows a high degree of connectivity. By contrast, in hierarchy every node is connected to at most* *one parent node and zero or more child nodes. In Heterarchy, however, a node can be connected to any of its surrounding nodes*".

Hierarchy provides a degree of order and structure. In a network each node can be accessed through a unique path. In contrast, within heterarchy any node can be connected arbitrarily with a number of other nodes. This creates a network with more potential paths and provides to the whole system feedback. Each structure has the advantages and disadvantages.

- **Stigmergy**: "*Stigmergy defines a paradigm of indirect and asynchronous communication mediated by an environment*".

  It is a property of decentralised environments in which the elements of the systems communicate between each other by modifying their local environment.

- **Perturbation**: "*A perturbation is a disturbance which causes an act of compensation, whereby the disturbance may be experienced in a positive or negative way*".

A self-organised system must be able to determine its boundaries. This means that the systems by themselves must be able to distinguish the borders between system and environment. If this can not be achieved, the system lacks control and the environment could influence its behaviour arbitrarily in a non-predictable way. From functionality perspective, the system must be able to operate independently from its environment. This does not mean autarchy, because from the definition of system there must always be an environment for interaction. The system must find a balance and manage the trade-off between autonomy and interaction with the environment. When it fulfills this condition, the system is said to be *operationally closed* and *energetically open,* Maturana and Varela (1980).

In some systems it can be noticed that although their structure varies, their organisation remains the same. It seems that structure and organisation refer to two different things. A system with structure does not mean that it is organised. Usually, the first one refers to the components and relations that constitute a certain unity in a concrete manner and realise its organisation. Organisation refers to these relations between components of something to recognise it as a member of a certain class, see Steinmetz and Wehrle (2005). The structure can be a variable of the system, but the organisation must always exist and it is a characteristic that classifies the system. For this reason, it is referred as the *identity* of the system.

Organisation of the system, or acting, should be maintained in various forms in a self-organised system. Maintenance means the ability of the system to repair and manage its components. The components must support the *viability* of the system and they must not be *isomorphic*, due to the fact that the purpose is the retaining of identity although the structure may change. In addition, system perturbations may disorder the system. By allowing a degree of heterarchy in the system in order to enable restructuring, the components can interact and receive or provide useful feedback. This feedback can be either positive or negative.

Self-organised systems can also be unstable which may lead them to breakdown, or inflexible which prevents them form evolving and adapting themselves to new conditions. The gap is bridged by criticality which balances the dynamics of the systems.

Self-organisation can not always provide improved systems in computer science. It is clear that existing self-organised systems lack management for controlling security and efficiency issues. The other properties of autonomic computing may fulfill the requirements, but it is certain that further research is needed towards this orientation.

Focusing on more details in the P2P world, there is the need to classify different P2P systems according to the degree of self-organisation. For this reason self-organisation must be outlined according to criteria and characteristics, which; H. De Meer and C. Koppen (2005) describe. There are two classes of self-organisation criteria. The *basic* and *autonomy* criteria. The following points outline these characteristics:

For the basic criteria there are the following:

- **Boundaries**: The boundaries of P2P systems must be self-defined. The boundaries are meant as the points where new nodes can enter the system. One way that this knowledge can be provided is by establishing special nodes called *bootstrap peers*.

- **Reproduction**: In a P2P network, nodes can form different structures and operations like adding, removing or changing a peer, its data, its connections, its neighbours or generally the relationships as a sign of reproduction. It is not important for these operations to lead to isomorphic relationships but the most important is to support mutations and viabilities.

- **Mutability**: This is similar with reproduction with the difference that mutability refers to the change of structure whereas reproduction to reproduction of the structure.

- **Organisation**: Refers to the outline of the system in terms of hierarchy and heterarchy.

- **Metrics**: The metrics are associated with the detection of perturbations which can be the following:

    - Failure of peer or connection.

    - Overload of DoS attacks.

    - Data manipulation.

  Similar effects to perturbations have the "freeriders", which are peers that consume resources of a system without offering their own. However; they are not perturbations because their effects influence the system internally.

- **Adaptivity**: Adaptivity refers to the reactions of the system to avoid the negative effects of perturbations.

For the criteria for autonomy the following are defined:

- **Feedback**: The negative or positive feedback in a P2P network can appear in a form of exchanged messages between the peers.

- **Reduction of complexity**: The purpose of some P2P systems is to introduce some entities with various roles in order for them to hide details from the environment and reduce the complexity. The virtual servers that are introduced in the proposed model (see section 2.2.1) follow this concept.

- **Randomness**: Organisation and structures in P2P networks are driven by random events for creativity. This can reduce the complexity and provide positive results.

- **Self-organised criticality (SOC)**: The effects of perturbations in P2P systems require flexible systems for avoidance and robust operation. Having a system with too much order or complete disorder can not support the fade out of perturbations. Criticality can provide the desired flexibility for managing perturbations.

- **Emergence**: It is possible that during the design phase of P2P systems, some of the properties have not been predicted. The system does not react by following certain rules and this behaviour is attributed to emergence.

Based on the above criteria, the self-organisation of Gnutella 0.4 (see Gnutella 0.4) can be examined. Starting from the boundaries, in Gnutella every peer is equal and can have the role of bootstrap server. For this reason any peer can enter and leave the system, and there are not determined boundaries. The invariant heterarchical structure also restricts the reproduction and mutability of the system. The organisation property can not be attributed to Gnutella as it is based only to the ping-pong messages. For this reason there is lack of feedback and adaptivity in the system. However; metrics can be realised in the system for controlling overloading issues caused by flooding. Randomness and self-organised criticality can not be attributed to Gnutella as it does not lead the system to creativity state and there is not the desired adaptivity between complete order and disorder. The system appears the attribute of emergence as there are nodes with high capabilities and they can offer resources to many peers and respectively there are other peers which consume resources without offering (freeriders). The whole structure is formed without external influence. The reduction of complexity can be associated with the power law distribution of node degree.

To conclude, autonomic computing can lead the design evolutions and can be the base of the concept of future systems. P2P networks can really benefit from this philosophy. Self-organisation seems to offer powerful characteristics in these systems but it is not panacea for solving the problems, view of Li and Liao (2005). It may lead to additional barriers. More research is needed. Biological models and more widely theories from multidiscipline fields can inspire the realisations of new P2P systems and the development of solutions to existing problems.

2. MODEL DESCRIPTION

The purpose of this developed model is the load balancing of Gnutella networks which is used as this case study. Gnutella networks, as it is outlined in section 2, suffer from high bandwidth consumption. This creates unbalanced nodes which may fall in the state of being unable to respond and process messages. At first sight, the solution seems difficult because flooding forces a horizon on the load and creates some virtual regions of high loading. The inspiration and orientation of this work is exactly to "break" this horizon and create a dynamic and adaptive way to load balance the network. There are also more reasons that networks which adopt the flooding are interesting for development of load balancing algorithms. Some of them are:

- It is a searching method which its effects over a P2P network have been studied extensively and the advantages and disadvantages are known to research community.
- It is one of the main techniques used in unstructured P2P networks.
- The network is evolved rapidly.
- It is a method which can be modified in an under development model like ERG or be upgraded to a more efficient searching technique.
- It creates an ideal load environment for experiments and model developments on capacity sharing.
- It is easy to be deployed in simulation level.

Balancing the load in an unstructured peer-to-peer network is challenging. Reasons include:

- The spatial distribution of the nodes and resources is heterogeneous.
- There is not a significant amount of information about the network. Structured schemes are better defined and easier to control and predict their behaviour.

- It is dynamic and this results in a high number of load changes in the network.

- They have been proven quite weak on this issue, Ritter (2001).

The next sections explain the functionality of the model and illustrate a detailed description. The algorithms are described in detail and figures show what happens when there are overloaded nodes. Two variations are proposed which have the same result, however; in a different way. Theoretical analysis of the two variations and investigation of how these algorithms may fail are illustrated. Furthermore; the load balancing messages specification is presented. Finally, the metrics of the evaluation of the load balancing models are introduced with explanations of their concept and their importance to the models success.

2.1 The Purpose and Focus of Load Balancing Scheme

As it is mentioned in 1.3, load balancing is a multidimensional operation. The first objective of this work was to define where exactly the load balancing scheme would be applied. The most serious problem in Gnutella networks is the high number of query messages that can cause failures in nodes. In Gnutella, every query is forwarded until the TTL value. This introduces a negative factor to the success of queries. If the node does not process the message, it will not forward it and in the end the query success may be affected (it is one of the purposes of this work to examine if the overloading can affect the query success, see section 4.7). In addition, it is speculated that the load balancing that is applied benefits the system positively in other overloading factors of the networks.

Another purpose is to achieve the best results at the lower cost. This means, keeping the information and process overhead low without introducing additional problems to the system.

Finally, it would be interesting to examine any new properties of the system that are gained due to ERG model. Is the system better self-organised? Does it get any new properties from the criteria defined in section 1.4.3? Does it appear adaptivity? And in the end, does it make Gnutella more efficient and scalable?

2.2 Load Balancing Model

In this section the load balancing model is described. Details about the proposed algorithms are provided and explanations of how the capacity sharing is achieved between the underloaded and overloaded nodes. Furthermore, additional details and discussions about how the models behave in extreme conditions, when they can fail and the comparison and contrast between the two variations are provided.

2.2.1 The concept

In a Gnutella network a node may become easily overloaded by the queries it receives. Regardless if this node has the resources or not, it must forward the message to all its neighbours. The importance here is that the load of all the nodes is caused by the nodes that send queries to them. It seems that the node degrees together with the heterogeneity of query generations are strongly associated with the overload in each node.

ERG focuses on modifying these relationships between the nodes that cause the overload and other nodes that are considered underutilized. The last are the nodes that do not seem to receive many queries from other nodes. Figure 1 shows a state of an overloaded and underloaded node in a Gnutella network:

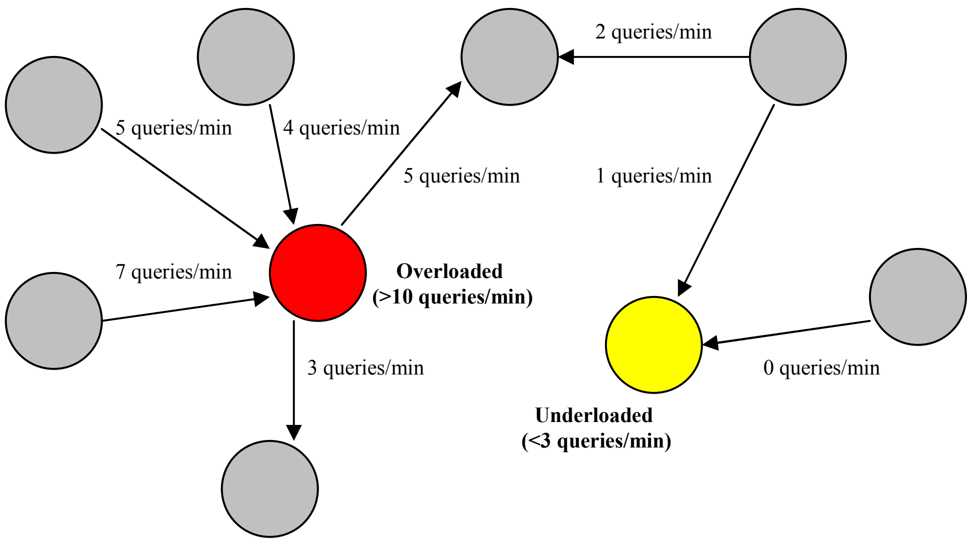


Figure 1: Overloaded and Underloaded nodes in a Gnutella network

In such a state, the proposed solution is based on the idea of unregistering the overloading node as a neighbour to the nodes that cause the overloading and register as a new neighbour the underloaded node. This action can be seen as a *logical movement* of the nodes that cause the overloading to the underloaded node. It would not be robust if the movement simply transfers the overloading to the underloaded node, and for this purpose special attention in queries rates must be paid. Figure 2 illustrates the result of the balancing:

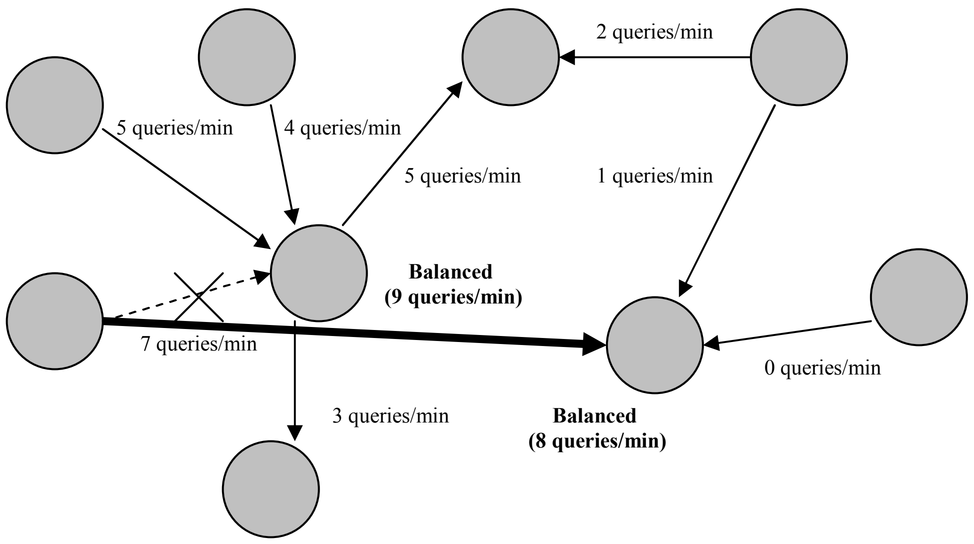


Figure 2: Logical movement of a node and load balancing

It is important to point out that the node can remain underloaded but that it must not be brought to an overloading state.

This whole action is not simple; it must be initiated and coordinated in a consistent way without influencing the functionality of the network. It does not seem a good solution for the overloaded node to start searching arbitrarily for underloaded nodes. It might be possible that it is unable to do this due to its overload and as such a searching may add to the network overhead of information. This approach seems to create more problems than it can solve. The proposed model is based on the idea of *virtual servers* for coordinating and managing the load balancing operations between all the interested components as they are outlined in the two above figures. The next subsection outlines the role of the virtual server in ERG model.

2.2.2 The Role of Virtual Servers

A virtual server, also known as; superpeer, ultrapeer or supernode, in a P2P network, is a normal computer that retains its basic functionality with all the other nodes in the network but it is assigned to it a special role for a special purpose because of its high capabilities. Performance is the criterion that makes a node virtual server in a P2P network. Performance incorporates a range of other criteria, but the most important are the hardware resources (CPU, hard disk, memory) and networking availability (bandwidth, low number of failures and high number of time connectivity). The purpose of the virtual server is not to bring back centralized architectures but to exploit the non-uniform resources that exist in every network by assigning some special roles to them.

In ERG, we establish the idea of virtual servers for load balancing. More specifically they support the actions of the logical movements described in the previous subsection.

Every node can be registered to one or more virtual servers. The registration and the discovery of virtual servers in the network take place during searching. If a virtual server receives a query from any node it can send a PONG message (see section 1.2.2) with an extra flag saying that the node is a virtual server. The query generator or the node which forwarded the query can then decide if it adds the virtual server to the *virtual server list* as long as it does not already exist there. It must be noticed here that a virtual server also keeps record of other virtual servers for two reasons:

- A virtual server is an equal peer and its only extra role is to balance the load of the network.

- A virtual server must know others in case it is unable to fulfill its role and can rely. In this case, one other virtual server from its list becomes responsible. The next section provides more details about such scenario.

Upon when nodes have registered virtual servers, the load balancing scheme can begin its work. More specifically, according to the capabilities of each node, there are thresholds which state if a peer is *underloaded*, *balanced*, *overloaded* or in a state to loose messages (*loss state*). If a peer is in the state of being underloaded, it sends a message to one of the virtual servers to advertise itself (ULN, see section 2.5). The load balancing operation is initiated when an overloaded node request help from a virtual server (VSS_RFH / VSD_RFH, see section 2.5). More details are illustrated in next sections.

Before the detailed description of the algorithms, it is imperative to mention an important issue. Before experiments are carried out it is not clear if it is the right decision to enable the virtual server to manage all the operations of the load balancing. On the other hand, it is not best practice to leave the overloaded node balance itself. During model design, two intermediate and moderate approaches have been identified in which the one charges the cost of balancing mostly in virtual server and the other to overloaded node. It is expected that following these two different approaches, interesting points can be raised about the overhead of the models and what the potential difficulties are for realising them. It is important for the comparison that the algorithms lead to the same load balancing outcome and examine other factors as extra overhead of the models, overloading threat of virtual servers due to the models e.t.c.

The next two sub-sections introduce the two model variations and the algorithms.

2.2.3 Virtual Server Supervised Model

In this variation the virtual server has a more passive role and its purpose is to inform the overloaded node about which available underloaded nodes exist in the network. Then it is the overloaded node which must negotiate about the logical movements.

If a node becomes overloaded, it searches its list of the virtual servers and selects one of them randomly. Then it sends a message to the selected one requesting help and with it, it sends its query rate distance, which is the total amount of overload. The virtual server will look up for an underloaded with the lower query rate per minute in its queue and will also check if the overload (query rate distance) of the overloaded node can make the underloaded node overloaded as well. If the conditions are satisfactory, the virtual server responds to the overloaded node with the address of the chosen underloaded node and its query rate. The overloaded node keeps statistics of the nodes by which it receives queries. It checks which of these nodes can be logically moved to the underloaded node in order for it to become balanced, maintaining the underloaded node, underloaded or balanced but not overloaded. It also calculates the total load that is transferred if one of these nodes satisfies the mentioned conditions. If there are nodes available, then it sends to the underloaded node a message requesting the logical movement of the nodes (the reason for the need of confirmation is to check if the underloaded node has become balanced through time and the underloaded to update its statistics information). The message contains the total amount of load that is desired to be moved together with the list of the nodes in order the statistics table of the underloaded node to be updated. It is chosen to retain the consistency of the data, so any logical movement retains the total loads and the indexes of the statistics tables. If the rates of the underloaded node have not changed to make the logical movements infeasible, the underloaded node confirms the actions. Then the overloaded send message to the nodes that cause the overload to send queries to the new node. The scenario is illustrated in figure 3, and the messages description is outlined in table 9. More information about the messages and their content is provided in section 2.5.

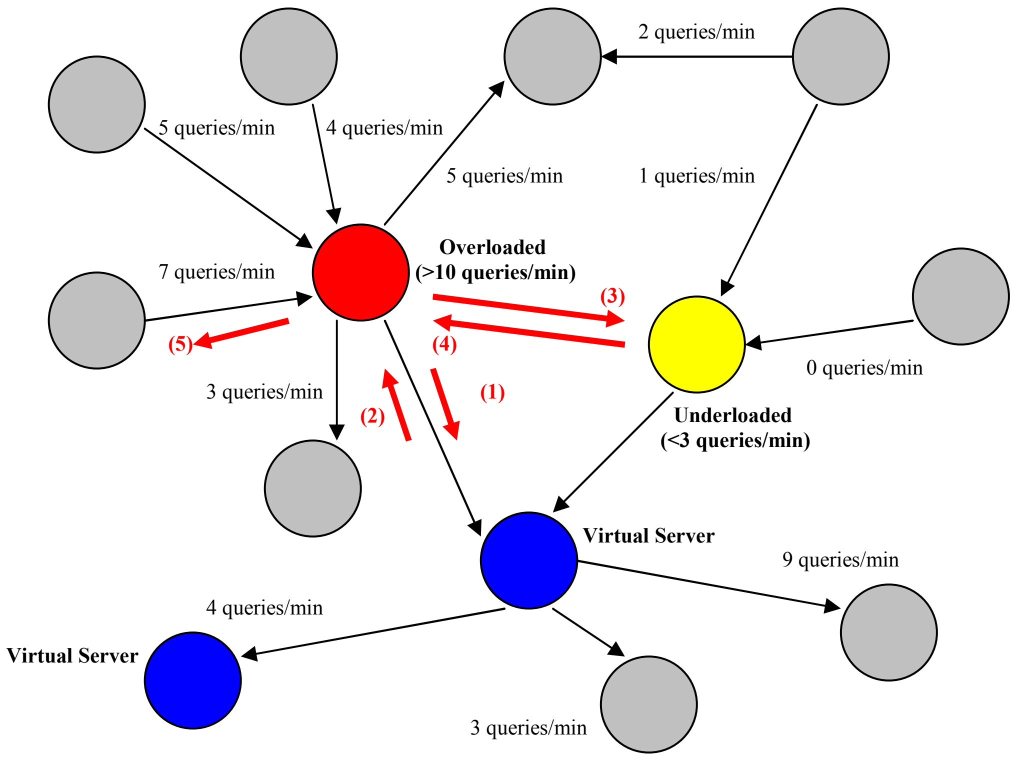


Figure 3: Load balancing in VSS variation

Table 9: Messages in VSS variation

	Message ID	Message
1	VSS_RFH	Request for help
2	VSS_RTOL	Response to overloaded
3	VSS_RTUL	Request to underloaded
4	VSS_CTOL	Confirmation to overloaded
5	VSS_UAA	Unregister and assign

If the virtual server does not have any available underloaded node, or the existing ones can be potentially turned to be overloaded if there are logical movements, the request for help from overloaded node is forwarded in another virtual server. The procedure is recursive until a TTL value. Figure 4 illustrates such a scenario:

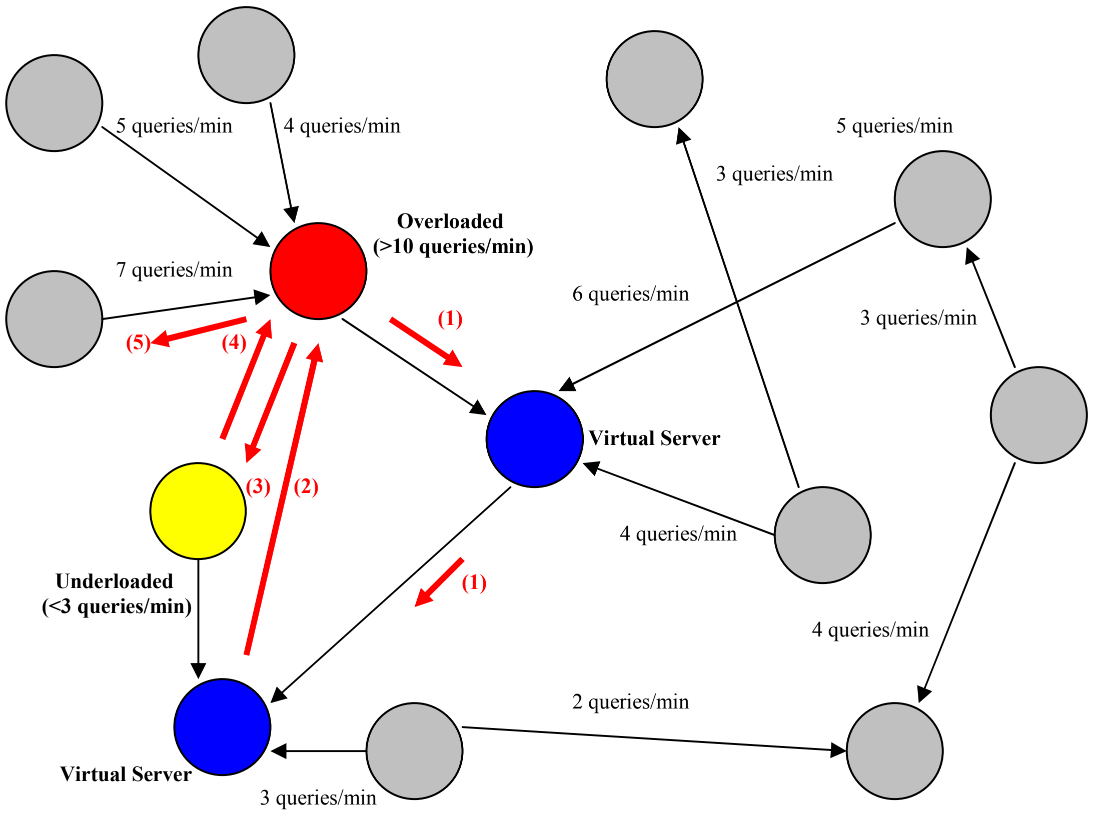


Figure 4: Forwarding of the request for help in another virtual server in VSS variation

The algorithm in detail is illustrated below together with some name conventions (message names are explained in details in section 2.5):

- OLThr: Overloading threshold.

- ULThr: Underloading threshold.
- VSIP: Virtual server IP.
- VSList: Virtual servers list.
- QRDist: Query rate distance.
- QRDistTotal: Total query rate that is moved.
- ULIP: Underloaded node IP.
- ULQueue: Queue of underloaded nodes.
- LBList: Load balancing list with the nodes that have been calculated to be moved.
- neighbList: List of neighbours of a node.
- LBStatDict: Load balancing statistics kept in a dictionary.

```
Algorithm ERG(VSS):
    while running do
        if QR > OLThr then
            VSIP←selectRandom(VSList)
            QRDist← QR - OLThr
            sendMessage(VSS_RFH, VSIP, QRDist)
        else if QR < ULThr then
            VSIP←selectRandom(VSList)
            sendMessage(ULN, VSIP, QR)
        if message queue is not empty then
            if message ID = ULN then
                if message.sourceIP exists in ULQueue then
                    update(ULQueue, message.QR)
                else
                    add(message.QR, message.ULIP)
            else if message ID = VSS_RFH then
                if ULQueue != empty and checkQR(message.QRDist) then
                    QRDist←ULQueue.getFirstKey()
                    ULIP←ULQueue.getFirstElement()
                    sendMessage(VSS_RTOL, ULIP, QRDist)
                    ULQueue.remove(QRDist, ULIP)
                else if message.TTL !=0 then
                    message.VSIP← selectRandom(VSList)
                    Message.TTL← Message.TTL-1
                    sendMessage(message)
            else if message ID = VSS_RTOL then
                i←0
                QRDistTotal← 0
                while i < size of LBStatDict do
                    if QR(i)+QRDistTotal+message.QRDist< OLThr then
                        LBList.add(IP(i), QR(i))
                        QRDistTotal← QRDistTotal+QR(i)
                    i←i+1
                if LBList is not empty then
                    sendMessage(VSS_RTUL, message.ULIP, QRDistTotal,
                    LBList)
            else if message ID = VSS_RTUL then
                if QR+message.QRDistTotal<OLThres then
                    LBStatDict.add(message.LBList)
                    QR←QR+QRDistTotal
                    decision←true
                    sendMessage(VSS_CTOL, message.sourceIP, decision)
                else
                    decision←false
                    sendMessage(VSS_CTOL, message.sourceIP, decision)
            else if message ID = VSS_CTOL then
                if decision = true then
                    LBStatDict.update(LBList)
                    QR←QR-QRDistTotal
                    i←0
                    while i < size of LBList do
                        sendMessage(VSS_UAA,               LBlist(i),
                        message.sourceIP)
                        i←i+1
            else if message ID = VSS_UAA then
                neighbList.replace(message.sourceIP, message.ULIP)
```

2.2.4 Virtual Server Driven Model

In this variation, the load balancing is achieved successfully as in the previous one, but in a different way. The purpose here is to avoid the extra cost to the overloaded node and the virtual server undertakes most of the actions to balance the nodes. This seems the most practical, but may introduce overloading issues to virtual servers from load balancing messages. One of the purposes of this work is to examine what really happens in these models and how much extra workload the virtual server could handle under extreme conditions.

The load balancing starts with the overloaded node requesting help to one of its virtual servers. It sends the query rate distance and the dictionary with the statistics of the nodes that send queries. Then the virtual server must complete the whole process to calculate, decide and send messages to all the nodes. It starts by searching an underloaded node in the queue that will satisfy the condition of avoiding the transfer of overload from the one node to the other. It also chooses which nodes will be moved and the total query rate that will be transferred. After finishing calculations and decisions, first it sends a message to the underloaded node informing it for the new load that is transferred and the nodes that will send queries in order to update the dictionary with the statistics. The next messages are to the nodes that should stop sending messages to the overloaded node, with the new address to replace in their neighbour list. The last message is a confirmation to the overloaded node that the procedure was successful together with the load that was moved and the new updated dictionary with the statistics. Figure 5 illustrates the interactions between the nodes during load balancing together with brief messages descriptions in table 10. More information about the messages and their content is provided in section 2.5.

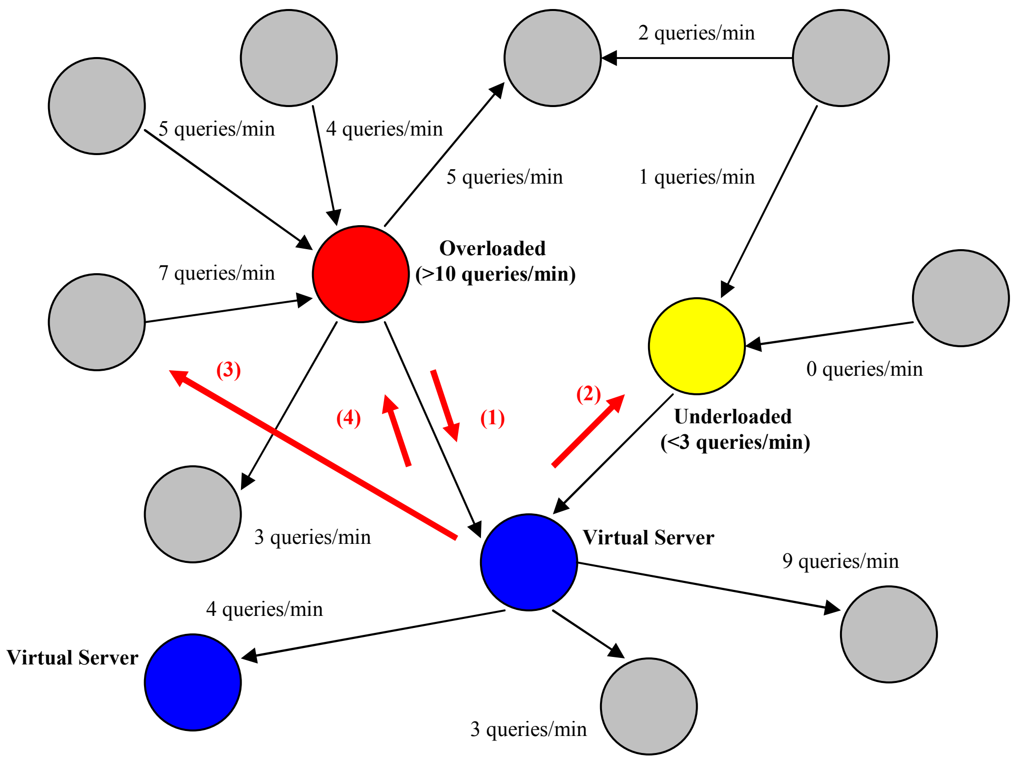


Figure 5: Load balancing in VSD variation

Table 10: Messages in VSD variation

	Message ID	**Message**
1	VSD_RFH	Request for help
2	VSD_RTUL	Request to underloaded
3	VSD_UAA	Unregister and assign
4	VSD_RTOL	Response to overloaded

If again, for any reason the virtual server fails to find an underloaded node, either because there is not any in the queue or the logical movements make the underloaded node overloaded, the request for help is forwarded to another virtual server until a TTL. A respective example is illustrated in figure 6:

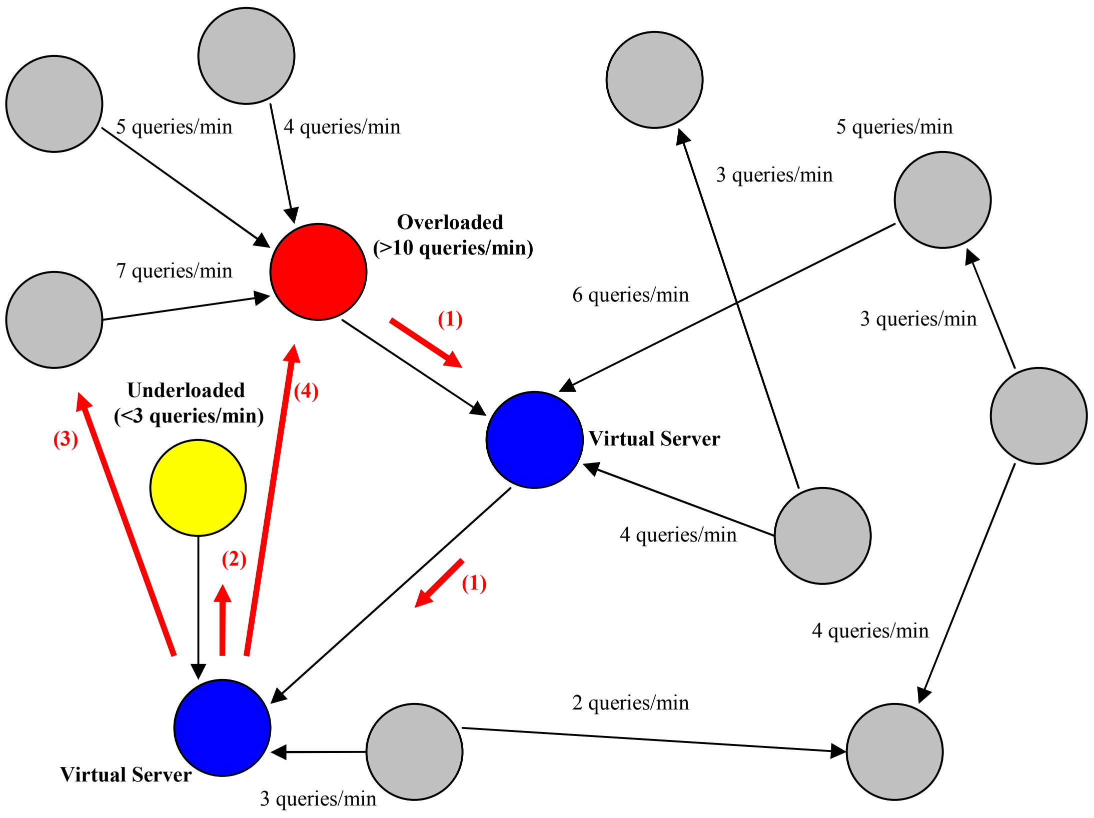


Figure 6: Forwarding of the request for help in another virtual server in VSD variation

The VSD algorithm together with some name conventions is illustrated below in detail:

```
Algorithm ERG(VSD):
    while running do
        if QR > OLThr then
            VSIP←selectRandom(VSList)
            QRDist← QR - OLThr
            sendMessage(VSD_RFH, VSIP, QRDist, LBStatDict)
        else if QR < ULThr then
            VSIP←selectRandom(VSList)
            sendMessage(ULN, VSIP, QR)
        if message queue is not empty then
            if message ID = ULN then
                if message.sourceIP exists in ULQueue then
                    update(ULQueue, message.QR)
                else
                    add(message.QR, message.ULIP)
            else if message ID = VSD_RFH then
                noMoreUL←false
        if ULQueue != empty and checkQR(message.QRDist) then
                QRDist←ULQueue.getFirstKey()
                ULIP←ULQueue.getFirstElement()
        else
            noMoreUL←true
        if noMoreUL= true and message.TTL != 0  then
            message.VSIP← selectRandom(VSList)
            Message.TTL← Message.TTL-1
            sendMessage(message)
        else
            i←0
            QRDistTotal← 0
            while i < size of LBStatDict do
                if QR(i)+QRDistTotal+ QRDist< OLThr then
                    LBList.add(IP(i), QR(i))
                    QRDistTotal← QRDistTotal+QR(i)
                i←i+1
            if LBList is not empty then
                ULQueue.remove(QRDist, ULIP)
                sendMessage(VSD_RTUL,     ULIP,     LBList,
                QRDistTotal)
                i←0
                while i < size of LBList do
                    sendMessage(VSD_UAA,     LBList(i),
                    message.sourceIP, ULIP)
                    i←i+1
                LBStatDict←update(message.LBStatDict)
                sendMessage(VSD_RTOL,     message.sourceIP,
                LBStatDict, QRDistTotal, true)

            else
                sendMessage(VSD_RTOL,     message.sourceIP,
                null, null, false)
    else if message ID = VSD_RTUL then
        LBStatDict←update(message.LBList)
        QR←QR+message.QRDistTotal
    else if message ID = VSD_UAA then
        neighbList.replace(message.OLIP, message.ULIP)
    else if message ID = VSD_RTOL then
        if decision = true then
```

```
                    LBStatDict←update(message.LBStatDict)
                    QR←QR-message.QRDistTotal
```

2.2.5 Where load balancing may fail - Extreme Conditions

From a theoretical perspective, the two proposed algorithms support robustness and successful load balancing without introducing to the network additional problems. However, the cases of failures, can be distinguished and also the reasons. It is important to clarify that failure of load balancing is faced in the overlay network and not in the underlying layer.

Failure of load balancing is defined as *the failure of an overloaded node to initiate the procedure of load balancing or the unsuccessful logical movement of the nodes that cause the overload*.

The following reasons, which belong to the above definition, have been identified:

- The overloaded node does not have any registered virtual server.
- The virtual server does not have any registered underloaded node for moving the load and the TTL of the RFH message is zero and it can not be forwarded further.
- The virtual server has registered underloaded nodes but the overload of the overloaded node makes it overloaded as well or the transferred nodes can not make the overloaded node balanced. In addition, the TTL of the RFH message is zero and it can not be forwarded further.
- The virtual server has found an underloaded node but it is not any more underloaded.

Most of these reasons are extreme conditions and it does not seem that they can appear in the network often except the last case. There is the need for queues on the virtual servers containing the underloaded nodes to be consistent. In VSS model the VSS_CTOL message guarantees this consistency. In VSD the virtual server is based on its local data. It may need to negotiate further with the underloaded nodes as a mechanism for retaining the consistency. The other reasons are related to the number of virtual servers in the network, the number of connections with the virtual servers and the TTL value for the requests for help. All these can be handled easily and in a predictable way.

2.2.6 Virtual Server Supervised versus Virtual Server Driven

Before experimental evaluation, the two variations can be examined from a theoretical perspective. The purpose of this comparison is to reveal which are the advantages and disadvantages of each one and how each one scales in a Gnutella network.

The first noticeable difference is the number of messages. VSS consumes 4+k messages, where k is the number of nodes that should be logically moved. VSD consumes 3+k messages. There is an additional message in VSD but VSS seems to retain the consistency of underloaded nodes in virtual servers better. In case of failure VSS initiates the procedure again. In VSS either the virtual server looks for underloaded nodes or it ends the load balancing unsuccessfully and the procedure is initiated in a next time interval. Furthermore, although VSD requires fewer messages, it transfers more data in the messages (the statistics dictionary).

The process overhead in VSS is charged in the overloaded node. This runs the risk of being unable to respond. The local state of the overloaded node is not transferred in the network (except the query rate distance) as it happens in VSD model. It is believed that a node that can tolerate an extra processing overhead can adopt VSS model because it reduces data transferred and allows the virtual server fewer tasks to process. On the other hand, if the overloaded node is in the extreme condition that may not be able to complete load balancing, then VSD model can be applied with extra processing overhead to virtual servers.

Experiments reveal how much the overhead is in each case, the extra messages that are required for load balancing and the success of each model.

2.5 Messages

ERG messages follow Gnutella v0.4 protocol specification (see Gnutella 0.4 specification). All the messages described in section 1.2.2 can be supported by the system. As far as the load balancing messages are concerned, the descriptor header is kept and the following messages are defined in the payload descriptor:

Table 11: UnderLoaded Notification (ULN)

Descriptor Header	Queries Rate

For VSS variation:

Table 12: Request For Help (VSS_RFH)

Descriptor Header	Queries Rate Distance

Table 13: Response To OverLoaded (VSS_RTOL)

Descriptor Header	Underloaded IP	Queries Rate

Table 14: Request To UnderLoaded (VSS_RTUL)

Descriptor Header	IP Cause Servents List	Queries Rate Distance Total

Table 15: Confirmation To OverLoaded (VSS_CTOL)

Descriptor Header	Confirm

Table 16: Unregister And Assign (VSS_UAA)

Descriptor Header	Underloaded IP

For VSD variation:

Table 17: Request For Help (VSD_RFH)

Descriptor Header	Load Balancing Statistics List	Queries Rate Distance

Table 18: Request To UnderLoaded (VSD_RTUL)

Descriptor Header	IP Cause Servents List	Queries Rate Distance Total

Table 19: Unregister And Assign (VSD_UAA)

Descriptor Header	Underloaded IP	Overloaded IP

Table 20: Response To OverLoaded (VSD_RTOL)

Descriptor Headed	Load Balancing Statistics List	Queries Rate Distance Total

2.6 Evaluation and Metrics

The following metrics are defined for the evaluation of ERG system:

- **Number of underloaded, balanced, overloaded and in loss mode nodes,** N_{UL}, N_{BAL}, N_{OL}, N_{LOSS}.
- **Number of nodes Handled Successfully,** N_S **:** The number of nodes subset of the sum of number overloaded and underloaded which handled successfully ($N_S \subset (N_{OL} + N_{UL})$).
- **Load Balance Success**, LB_S: The number of nodes that handled successfully divided by the sum of number of overloaded and underloaded nodes, $\frac{N_S}{(N_{OL} + N_{UL})}$.

- **Recall**, R: The total number of files discovered divided by the total number of relevant files in the network, $\frac{F_{Disc}}{F_{\mathrm{Re}l}}$.

- **Traffic**, M_{ERG}: The total number of messages the model produces.

- **Discarded Messages, M_{DISC}:** The number of messages that are discarded due to the overload in nodes.

- **Standard Deviation of Query Rate, σ_{QR}:** The standard deviation of the query rates of all the nodes in the network, $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}$.

- **Query Success**, Q_S: The number of successfully answered queries divided by the total number of queries generated, $\frac{N_{QH}}{N_Q}$.

- **System Availability, Av_{Syst}:** The percentage of the availability of the system, calculated from the total overload and the total load of all the nodes in the system, $1 - \frac{OL_{Total}}{L_{total}}$.

The most obvious and simple results can come from the calculations of the number of underloaded, overloaded, balanced and in loss mode nodes in the system. The desired result would be a system with most nodes being balanced. This is associated with the standard deviation, so by eliminating the standard deviation, it is expected that the number of balanced nodes will increase.

For evaluation of the load balancing scheme and its failures, we can calculate the load balance success. Furthermore the traffic of the model is a good indication of the overhead of the proposed models. The discarded messages are one of the most important metrics. Comparison between the pure Gnutella and the two variations will reveal if there is real improvement in the performance of Gnutella.

Another important metric is the system availability, as it is defined in Li and Liao (2005), which is calculated globally in the system. This is a metric for revealing the effects of load balancing in the overall system.

Finally, it is believed that discarded messages and failures in the system affect the query success and recall. It would be an interesting investigation of any existence of potential improvements in these metrics by the load balancing models.

3. MODEL SIMULATION

In this section the simulation of the described model as a tool for evaluating its success and study its behaviour are outlined. Testing of an algorithm or a system can be realised either in reality, by building a real networking system which will incorporate the functionality and behaviour of the model / algorithm or in the simulation level. The first way is closer to transferring a technology to the form of a prototype, or a functional demo. This reveals a mature technology with positive indications that it can work well and efficiently. In practice, before this stage there is the simulation stage. For avoidance of problems, developers first want to see how the system behaves in a simulating environment, with assumptions, different parameters and multi-varied conditions. Furthermore, in academic world, simulation is a serious tool for theory proof and it is used extensively in various scientific fields.

In P2P world, simulations are extremely important and can be sometimes the only tool of researchers to test and evaluate algorithms and networking models. This is because P2P networks do not have any centralized entity to collect global information in the network. Nodes may be counted to hundreds of thousands and the relationships to millions. Such a networking state can not exist in a university environment or in any lab or other research facilities. Simulation seems the only way in P2P world, at least at the beginning.

For this work, the concept is the same. An unstructured P2P network like Gnutella is fully decentralized, with an arbitrarily number of nodes and connections. Additionally, the proposed algorithms are also distributed and dynamically functioning, for this reason, studing their behaviour locally is infeasible and bad practice.

ERG is simulated in a Java simulator, developed from scratch for the purpose of serving the needs of the model, but also as a framework for a future and further realization of a generic P2P simulator with advanced functionalities. It deserves describing its architecture, its functionalities and its flexibility it provides for ERG simulation. The next subsections focus on the mentioned descriptions.

3.1 Supported Functions

The purpose of this simulator is on the one hand to serve the simulation of the ERG model in a Gnutella 0.4 Network and on the other hand to support the foundations of more advanced and generic functionalities. The supported functions are outlined below:

- **Creation and basic functionality of a node**: The simulator can realise the creation of a node by allowing it to specialise its behaviour, take special roles (be a virtual server), keep useful information and monitor its state.

- **Creation of a Gnutella Network**: The simulator creates a Gnutella Network and some of the parameters that the user can modify include the following:

    - Number of nodes.

    - Number of neighbours of each node.

    - The TTL of the queries.

    - Searching techniques.

        - Flooding.

        - Agents, Pournaras (2007) .

    - Circles (cut or not).

- **File Distributor**: There are different schemes in a P2P network for looking up resources. The idea of keywords is established. The distributor assigns in all the nodes an equal number of files, with a certain number of keywords and each file is described by a variable number of keywords defined be the user.

- **Downloading**: The simulator supports downloading simulation with files being copied in the nodes. This provides more realistic evolutionary searching scenarios.

- **3-D Network Visualization**: This functionality is partially supported and it is part of further work and extension.

- **Load Balancing**: The simulation of the two variations. Some of the parameters that can be modified include:

    - Choice of the load balancing variation, VSS or VSD.

    - Number of virtual servers.

    - TTL of the requests for help.

    - The underloaded, overloaded and in loss mode thresholds.

- **Running time**: The execution of the simulation, which can be dynamic with:

    - Different number of queries in each iteration.

    - Different number of iterations.

- **GUI**: A simple user interface for controlling all the above functionalities.

3.2 Choice of Technologies

The choice of the technology has been regarded an important issue and study of related simulators, like the work of Ting and Deters (2003), has been critical. Java has been chosen as the main technology for building the simulator for the following reasons:

- It has become faster, sometimes even faster than C++, Almaer (2004).

- It can support the conceptual background of the model in the best way, due to its fully object-oriented nature.

- It makes memory management an easy task.
- It has various libraries and APIs for embedding in Java other powerful extensions and functionalities.
- The simulator can benefit from the future evolutions of the language.

A P2P simulator also needs a powerful environment for working, in a low level, with data structures. Java supports many powerful data structures, however; JDSL (Java Data Structures Language, see Tamassia, Goodrich, Vismara, and Handy (2005)) is chosen, as integration, for the following reasons:

- It has one of the best range of data structures which includes and provides support of:
    - Sequences (lists, vectors).
    - General-purpose trees.
    - Priority queues (heaps).
    - Dictionaries (hash tables, red-black trees).
    - Graphs.
    - Template algorithms.
    - Sorting algorithms.
    - Graph traversals.
    - Shortest path, Minimum spanning tree.
    - Iterators.
    - Accessors (positions and locators).

  - Decorations (attributes).

- Flexibility: For modeling a network, a database and a message by using the developed data structures or other built on top of JDSL library.

- Reliability: A range of exceptions which secure the code.

- Efficiency: Most data structures offer the best-possible asymptotic time complexity for every supported operation and caching.

- Object-orientation: View of data structures and algorithms as objects.

The project has been developed in NetBeans 5.5. It is also powerful and this version is robust with plenty of functionalities.

The 3-D network visualization, which is under development, is built with JOGL (Java OpenGL) API, which provides high performance and the advantages of OpenGL (3-D hardware acceleration).

3.3 Design and Architecture

At first, the components and their relationships are defined. The simulation is consisted of four components in a layer hierarchy. The *User Interaction and Parameterization* (UIP) is the upper layer and parameterises the system according to the user feedback. It also receives the output from the simulation. The *ERG Protocol* (EP) layer, is the set of parameters and rules that create the Gnutella network and form the load balancing scheme. It is parameterised by the UIP layer and provides information to PN layer (see below).

The next layer is the *Simulator Engine* (SE). This layer is a middleware and it coordinates the relationships between the UIP and the EP and PN. It also collects useful information for calculating various metrics. The P2P Network layer (PN) is the last one in this hierarchy. It forms the Gnutella network and communicates both with EP and SE layers. Figure 7 illustrates the layers hierarchy of the components and their relationships and interactions in ERG simulator:

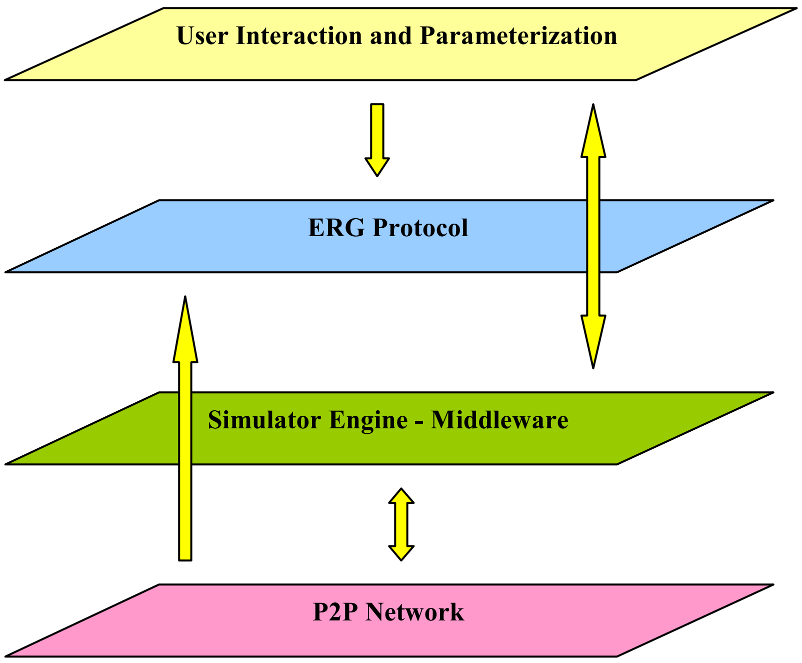

User Interaction and Parameterization
ERG Protocol
Simulator Engine - Middleware
P2P Network

Figure 7: Layer Architecture of ERG Simulator

During the design phase, the following classes have been decided to form the simulator:

- `Node`: It keeps the information and the functionality of a node in the network. It must be able to receive, process and send messages. It must also retain information and be able to process it on demand.

- `Message`: This class is the messages specification of ERG protocol. It defines the messages, the fields and other useful properties .

- `ERGProtocol`: It retains the parameters that configure the network and the load balancing model

- `P2PNetwork`: Creates the network and implements the simulation.

- `Agent`: It implements the agent based searching technique

- `Visualiser`: The GUI of the simulator.

- `Net3D`: The class that creates the 3-D visualization of the network.

- `ScreenManager`: Configures the screens of GUIs.

Figure 8 illustrates the UML class diagram of the ERG simulator. For simplicity, only the operations have been included and not the attributes. However, the arguments and the return types of the methods are included:

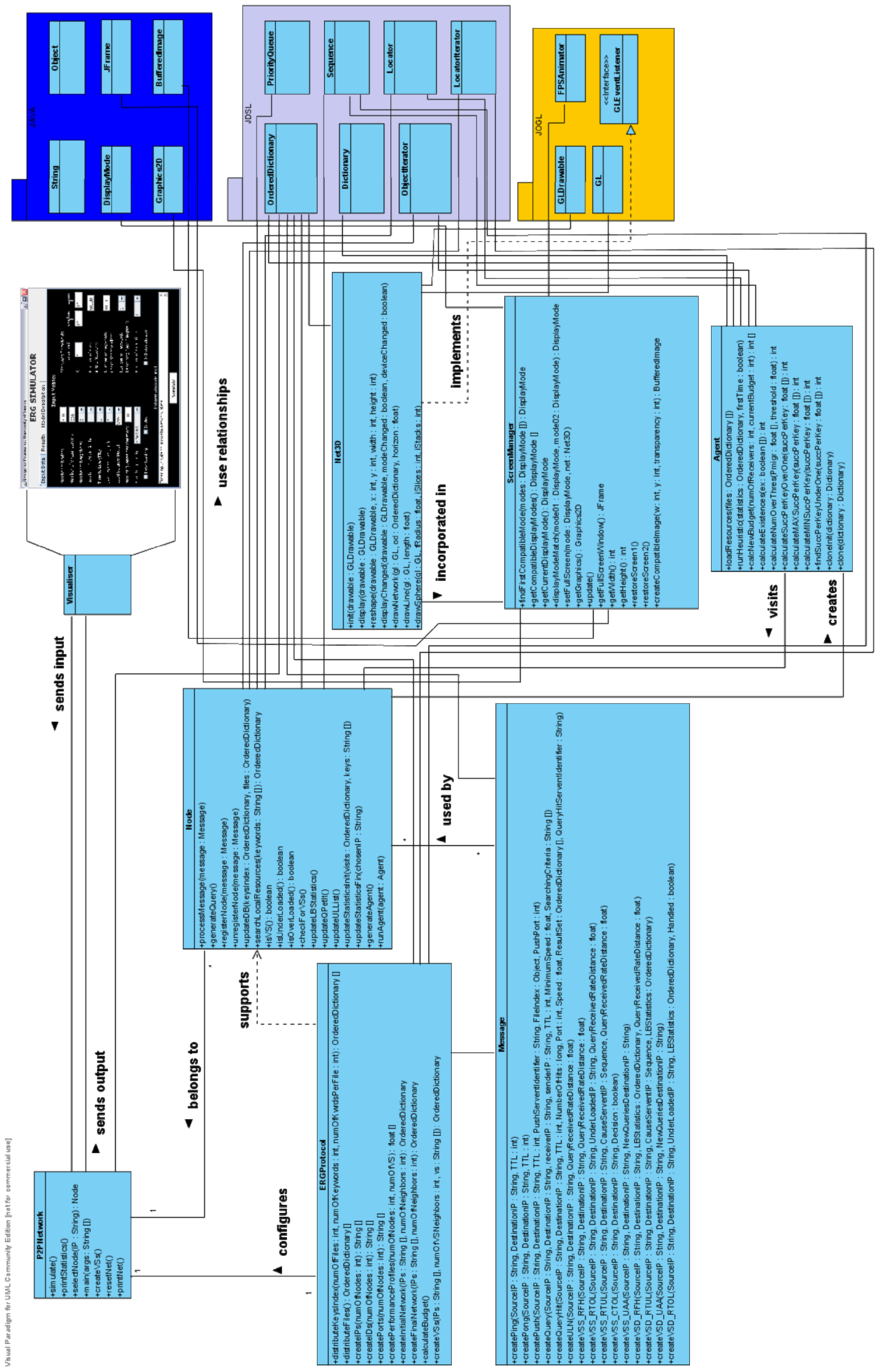


Figure 8: UML Class Diagram of ERG Simulator

From the above diagram, it can be noticed that `P2PNetwork` class instantiate Node class for the creation of the Gnutella network and these two classes form a one-to-many relationship. Furthermore, the `P2PNetwork` uses always one instance of `ERGProtocol` (one-to-one relationship). `Message` class has a many-to-many relationship with `Node` class, because many messages can be sent and received in many nodes of the network. The packages which are depicted in the diagram provide usage access to their classes by the classes of ERG simulator and the input / output is transferred between the `P2PNetwrok` and `Visualiser` class.

Based on this design and architecture, the implementation of the ERG simulation is realised and is described in detail in the next subsection.

3.4 Implementation and Graphical User Interface

Implementation started by building the general components of the simulator and then going into deep details of the simulation. `Message` class is the messages specification. It defines the message identifiers as static `Strings` and also defines the fields of the various messages. The class contains methods for creating a message. For example, the method `createVSS_RFH(…)` configures an instance of a `Message` class to carry information about the VSS_RFH message. The following code shows how the appropriate message can be configured:

```
Message mess=new Message();
String source=”111.111.111.111”;
String destination=”222.222.222.222”;
float load=5.2;
mess.createVSS_RFH(source, destination, load);
```

Then this instance can be sent and accessed simply by its variables which the above method has configured.

`ERGProtocol` class contains information about the network, how it is configured, the load balancing parameters and also is a global component which keeps track of the metrics and the statistics that are calculated. It also creates some basic information about the network, such as neighbour lists, virtual server connections and more. In this class, `P2PNetwork` and `Node` have access to get useful data.

`P2PNetwork` class accesses the `ERGProtocol` to create the network and initialise the system. Then it creates instances of `Node` class and puts them in an `OrderedDictionary` with the IP being the key and the instance the element. In this way, access to every node, from a global perspective, can be achieved. This class is the main class and it runs the simulation. It also prints the statistics of the calculated metrics.

The node class is the most important. It incorporates some basic functionality that must exist in every node and also has some fields that indicate if the node is virtual server. The node contains the load balancing algorithm and can run it if its query rate exceeds the overload threshold. The most important method of this class is the `processMessage(Message mess)`, which can be outlined with the following code:

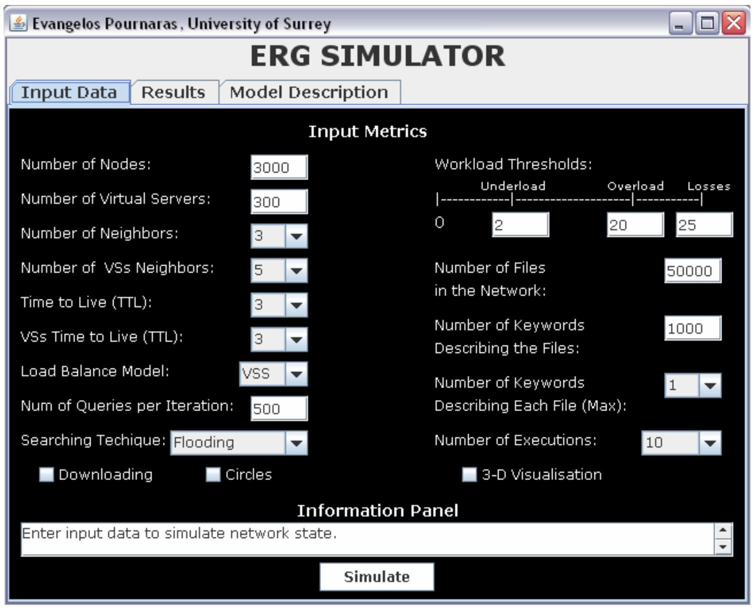
```
public boolean processMessage(Message mess){
      boolean success=false;
      if(message.PaypadDescriptor.equals(message.Query)){
            //make processing for a query
      }
      else{
            //check every possible message descriptors
            //and embed the respective operations
      }
      return success;
}
```

The files in the node are saved in a `Dictionary`, with the key of the dictionary being the keyword of the file. The neighbours of the node are kept in an `ArraySequence`. The load balancing statistics are retained in a `Dictionary` and the underloaded nodes in virtual servers in a `PriorityQueue`, prioritised by their query rate. The `OrderedDictionary` is implemented with a `RedBlackTree`.

At the beginning of every iteration the queries are generated. The `processMessage()` method of the node works recursively, by first accessing the node from method `selectNode()` of `P2PNetwork` class and then calling the `processMessage()`

method of the desired `Node`. This is efficient and conceptually accepted after having defined simulation engine as a middleware.

A graphical user interface has also built for parameterising the system. It is better to provide a degree of control to the user to allow him make a wide range of experiments without having to access the code. The GUI is built with Java Swing and figure 9 illustrates a screenshot:


Evangelos Pournaras, University of Surrey

ERG SIMULATOR

Input Data | Results | Model Description

Input Metrics

Number of Nodes:	3000	Workload Thresholds: Underload / Overload / Losses	0 · 2 · 20 · 25
Number of Virtual Servers:	300	Number of Files in the Network:	50000
Number of Neighbors:	3	Number of Keywords Describing the Files:	1000
Number of VSs Neighbors:	5	Number of Keywords Describing Each File (Max):	1
Time to Live (TTL):	3	Number of Executions:	10
VSs Time to Live (TTL):	3		
Load Balance Model:	VSS		
Num of Queries per Iteration:	500		
Searching Techique:	Flooding		

☐ Downloading ☐ Circles ☐ 3-D Visualisation

Information Panel

Enter input data to simulate network state.

Simulate

Figure 9: GUI of ERG Simulator

3.5 Evaluation of the ERG Simulator

It is important the simulator works successfully and produces the expected results. Effort has been given to satisfy that the simulator calculates correct the metrics and it is robust.

The first phase of the evaluation is simple mathematical calculated metrics for comparison with the results of the simulator. For example, we want to show that the total messages that Gnutella produces in the simulator are the same with the value expected. We can calculate the number of messages for a network with 500 nodes, 3 neighbours per node, TTL=3 and 100 generated queries as:

$$M_{ERG} = N_{Queries} * \sum_{i=1}^{TTL} N^{i}{}_{Neighb} = 100 * (3^1 + 3^2 + 3^3) = 100 * 39 = 3900 \text{ Messages}$$

In the simulator, if we run the experiment (without cutting the circles) we get the following results:

- Total average messages: 3950.
- Query Success: 45 %.

We have generated 100 queries and we got at least 45 query hits. This means that the simulator works correctly and the 3950 messages is an expected value.

The same concept is followed for other metrics as well. In load balancing, calculations of average query rates and standard deviation have been very useful for checking the underload/overload thresholds and verifing the correctness of the outputs.

As a last evaluation action, effort has been given to form experiments similar to other work for comparison of the results. Some experiments, oriented to query success have taken place in comparison with Oeztunali, Rusitschka, and Southall (2006), and under different network parameters the query success is almost the same with the ERG simulator.

This secure and robust simulation environment can be used for the load balancing experiments and can provide reliable results.

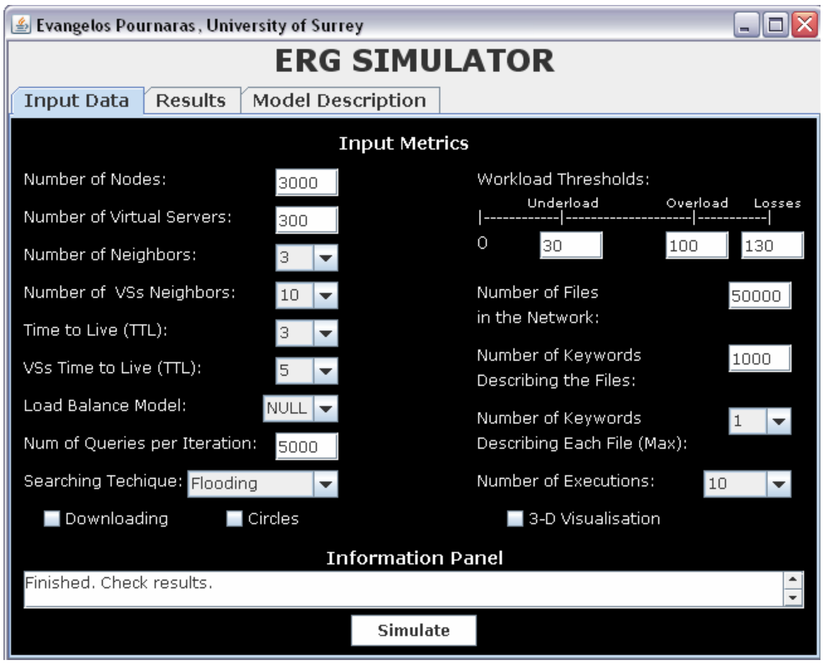
4. RESULTS AND DISCUSSION

Evaluation of the proposed load balancing model is done through various experiments, each one focuses on different aspects, examination and analysis of the model. The results are illustrated in different sections, each one describing and analyzing the findings. Before this, the different scenarios and data inputs in the simulator are discussed.

4.1 Experimental Environment

For presenting the results, two basic simulation environments were chosen. More were used during experimentations, but it is believed that these two represent in the best way the final results and outline the success of the proposed models.

The first environment is a realistic and average condition of the network. The parameterisation of the system is according to figure 10:



Evangelos Pournaras, University of Surrey

ERG SIMULATOR

Input Data | Results | Model Description

Input Metrics

Number of Nodes:	3000
Number of Virtual Servers:	300
Number of Neighbors:	3
Number of VSs Neighbors:	10
Time to Live (TTL):	3
VSs Time to Live (TTL):	5
Load Balance Model:	NULL
Num of Queries per Iteration:	5000
Searching Techique:	Flooding
Downloading	☐
Circles	☐

Workload Thresholds:

	Underload	Overload	Losses
0	30	100	130

Number of Files in the Network:	50000
Number of Keywords Describing the Files:	1000
Number of Keywords Describing Each File (Max):	1
Number of Executions:	10
3-D Visualisation	☐

Information Panel

Finished. Check results.

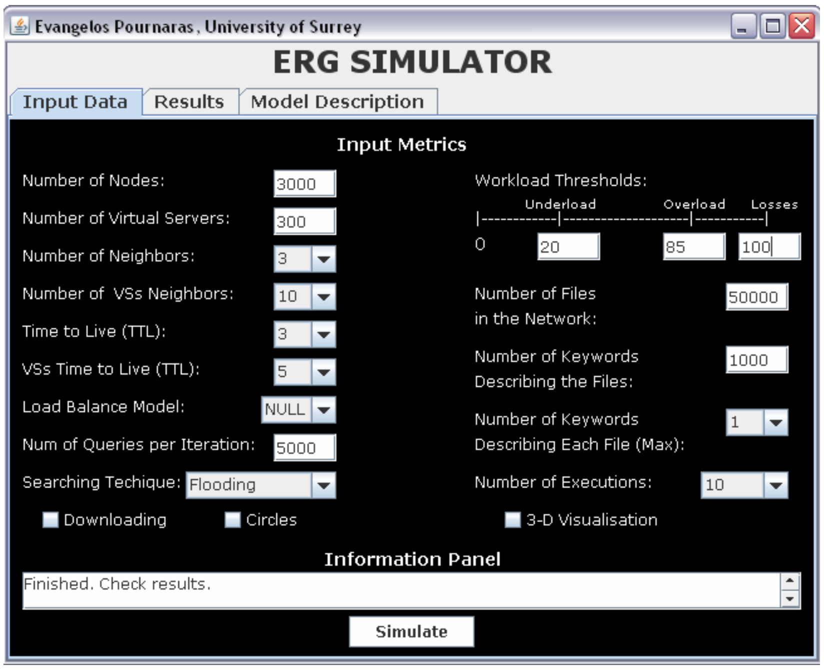
Simulate

Figure 10: The main simulation environment with the parameters

A high number of nodes have been used, 3000 in total. More nodes are supported but for faster response, this value is reasonable and good for the experiments in question. The number of virtual servers is selected arbitrarily to be the 10% of the total nodes in the network. TTL=3 can deteriorate query success and we want to see any potential improvements with the proposed models. The TTL for load balancing has been set to five. It can be considered quite a high value, however; the purpose is to exploit the dynamics of the model by eliminating the probabilities of the failures that are described in section 2.2.5. A highly loaded environment with 5000 queries per iteration has been formed. The queries are generated randomly and this satisfies, with high probability, that all the nodes will generate at least one query. It was preferred in these experiments not to put another dynamic in out network with enabled downloading. First, it is reasonable to examine the success of the models in an environment with standard resources. There is also no need to enable circles.

One of the greatest challenges in creating the experimental environment was the set up of the thresholds. The way that the thresholds has been chosen follows the logic of the average query rate per node and the standard deviation of all the query rates in all the nodes in the network. When the pure Gnutella runs, this value can be accessed and provides an indication of how to challenge the models. For the input data the average query rate per node has been calculated approximately 65 and the standard deviation approximately 37. The overload threshold is set to 100, near the average, plus the standard deviation of the queries. The losses mode threshold is arbitrarily set to 130. 30 units are left for enabling the system to load balance itself before starts losing messages.

In the second experimental environment all the data has been kept the same except the thresholds. An environment that probabilistically loses messages was created. The number of these messages is large. The load for both loss mode and underload thresholds is reduced (fewer available underloaded nodes in the network). Figure 11 illustrates the second environment:



Number of Nodes: 3000
Number of Virtual Servers: 300
Number of Neighbors: 3
Number of VSs Neighbors: 10
Time to Live (TTL): 3
VSs Time to Live (TTL): 5
Load Balance Model: NULL
Num of Queries per Iteration: 5000
Searching Techique: Flooding
Downloading
Circles

Workload Thresholds: Underload 0 – 20, Overload 85, Losses 100
Number of Files in the Network: 50000
Number of Keywords Describing the Files: 1000
Number of Keywords Describing Each File (Max): 1
Number of Executions: 10
3-D Visualisation

Finished. Check results.

Figure 11: The second simulation environment with the parameters

The results of the metrics, that are illustrated in the next subsections, include the following (for pure, VSS and VSD simulations):

- Load profiles of the nodes in the network (simulated in the 1st environment).
- Extra traffic in the network by the load balancing models (simulated in the 1st environment).
- Standard deviation of the query rate in all the nodes of the network (simulated in the 1st environment).
- Number of processes by the overloaded and the virtual server during load balancing (simulated in the 1st environment).
- System availability (simulated in the 1st environment).
- Query success (simulated in the 2nd environment).
- Number of discarded messages (simulated in the 2nd environment).

4.2 Load Profiles of Nodes in the Network

Initially, it is interesting to examine the loading profiles of the nodes in the network. This means how many nodes are balanced in pure Gnutella and how many after the load balancing. Furthermore, showing the number of nodes which are in the overload and loss mode. Figures 12, 13 and 14 illustrate the results for pure Gnutella and the two proposed models:

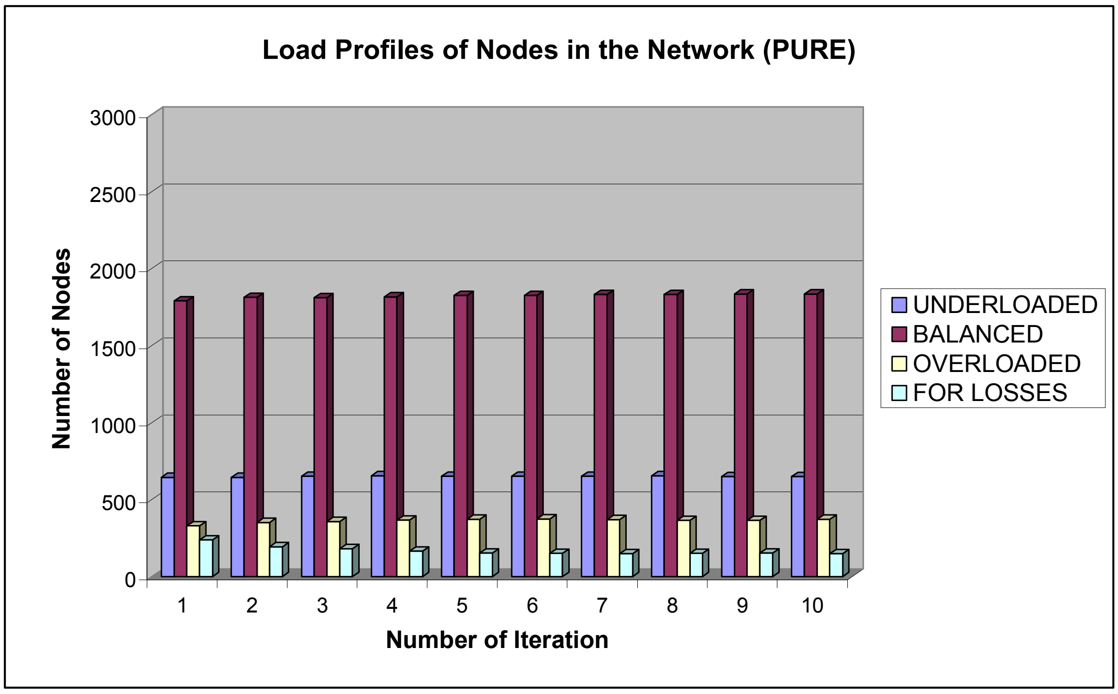


Figure 12: Load profiles of nodes for pure Gnutella

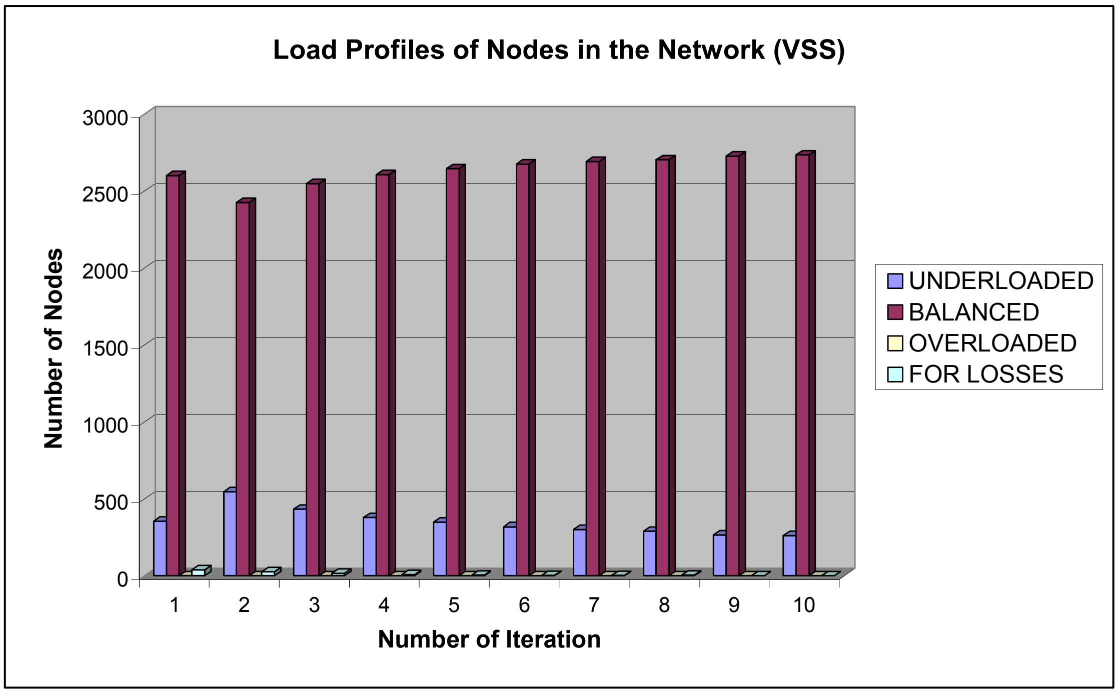


Figure 13: Load profiles of nodes in VSS load balancing model

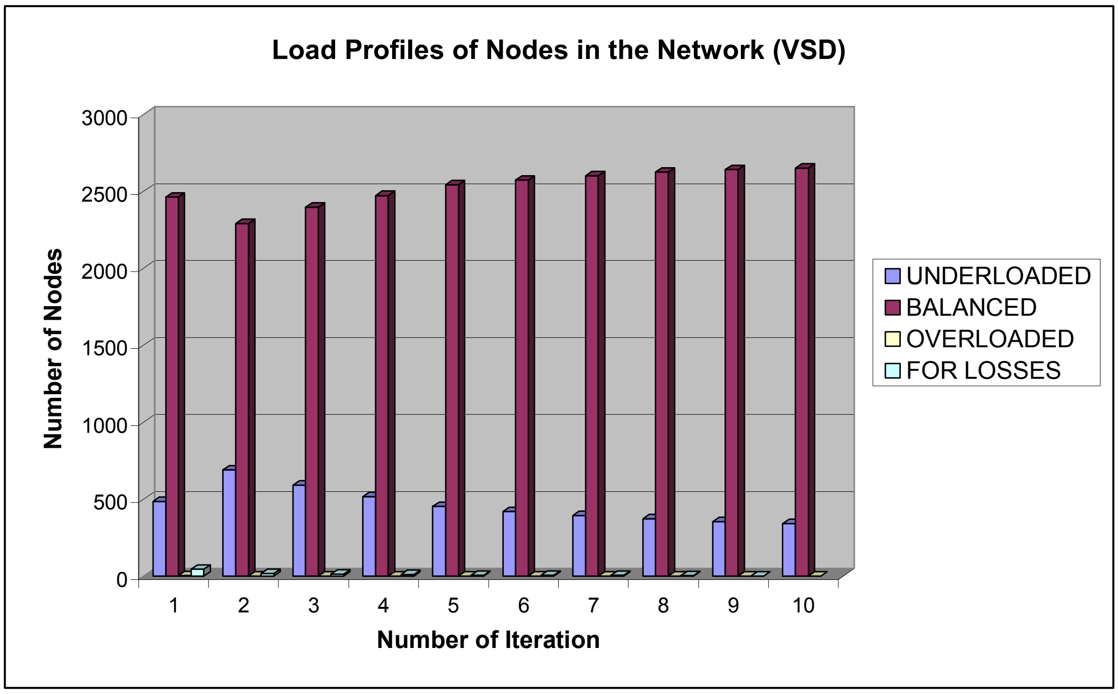


Figure 14: Load profiles of nodes in VSD load balancing model

As it can be noticed from the above figures, there is a significant improvement. More specifically, the number of overloaded and loss mode nodes has been eliminated and it approaches values near to zero. In addition, the number of underloaded nodes has been reduced by more than 50%, which reveals a better exploitation of the system resources.

Furthermore, the number of balanced nodes has been increased significantly from 60% to 83% approximately.

Comparing the two models, there is no significant difference in results. The only comment could be that VSS evolves slightly faster. Evolve means reaching the optimum performance.

4.3 Standard Deviation of Query Success in Nodes

Previous results show that the number of balanced nodes has increased significantly. What has happened is that the standard deviation of query success in nodes has decreased and it is more difficult for the query success to extend the overload thresholds. It is clear that the standard deviation is closer to average query success and figure 15 illustrates the results and the comparisons between the pure Gnutella and the two load balancing models:

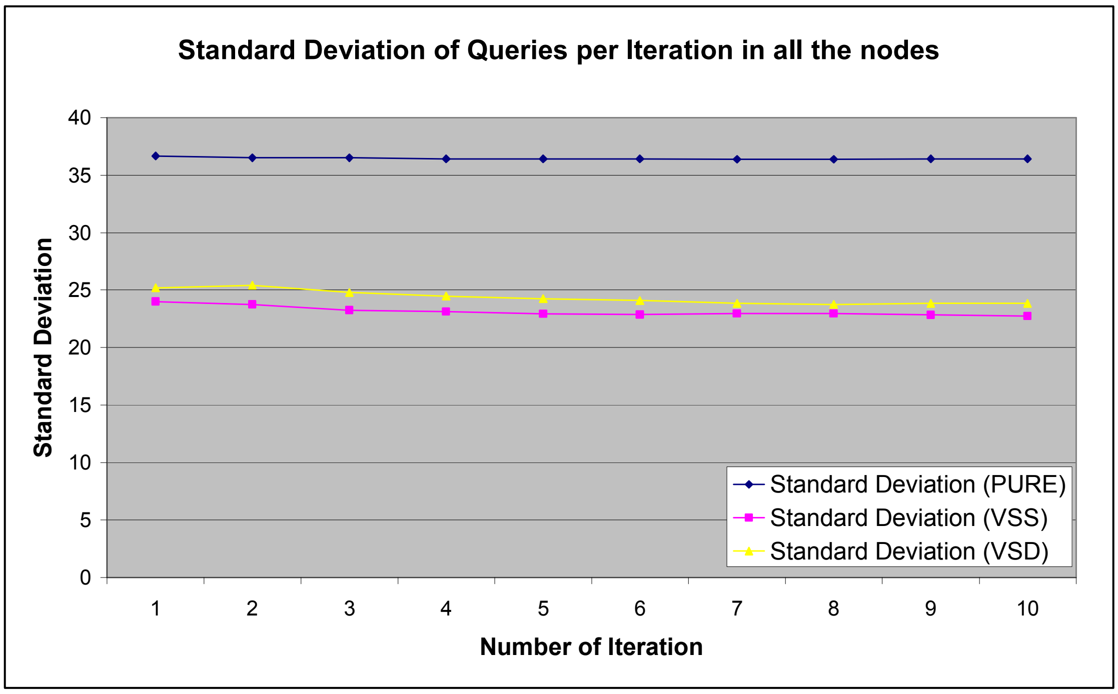


Figure 15: Standard deviation of query success in nodes of the network

4.4 Traffic of Load Balancing

It is obvious from the previous experiment that that there is a significant improvement. The important question now is what the cost for this improvement is. Cost is the number of messages that are introduced in the system. Figure 15 illustrates the results for the same simulation environment:

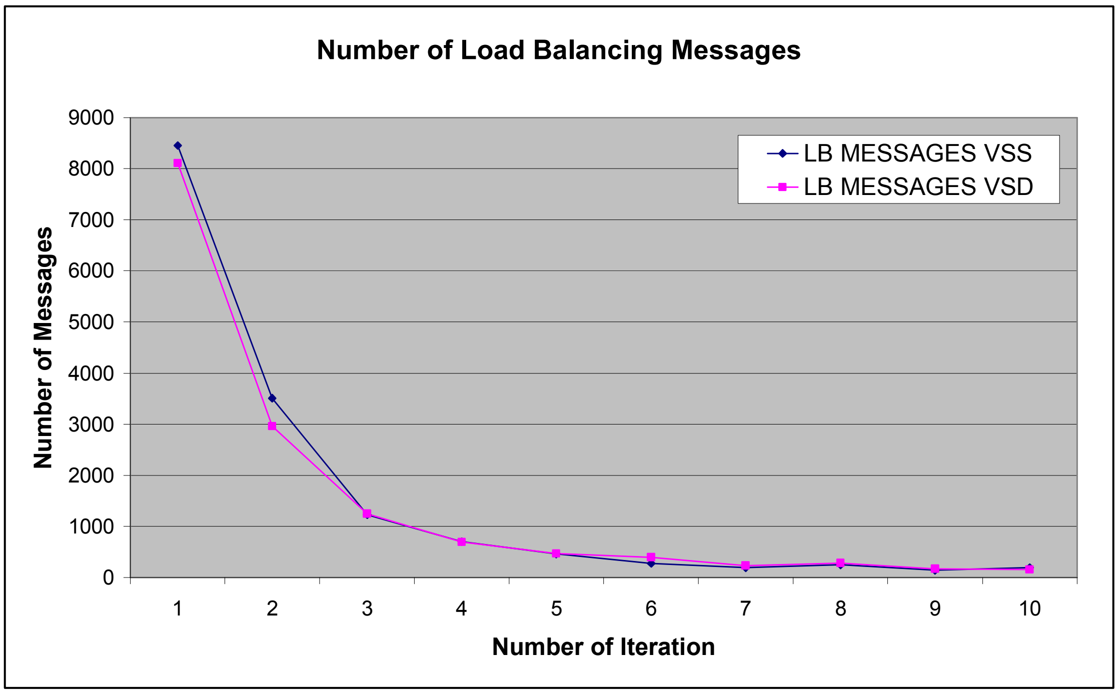


Figure 16: Extra traffic introduced by load balancing

There is something very interesting in these results. Although the system appears to retain its successful balance that has been achieved by the models, the cost for retaining this balance is reduced exponentially. This means that the system finds a state that does not need to apply load balancing, a state of balance (not load balance but functional balance imposed by the proposed models).

Comparing the two variations, it seems that they consume almost the same number of messages. It was mentioned in the previous subsection that VSS seems to evolve faster and this faster evolution can explain the slightly increased number of messages compared to VSD.

4.5 System Availability

In section 2.6, the system availability is defined. System availability provides a global quantified indication of the improvement by load balancing scheme. Figure 16 illustrates the results:

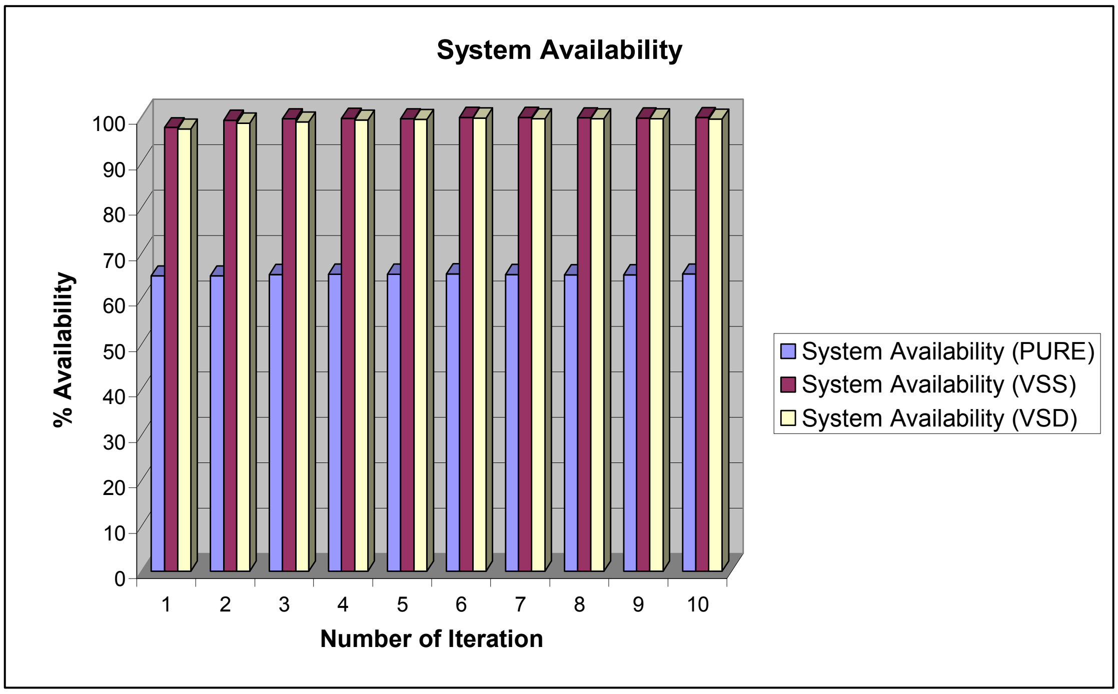


Figure 17: System availability in pure Gnutella and in load balancing versions

The figure reveals that the load balancing provides a significantly improved robust system, with over a 30% increase in system availability. The two variations approach almost the same availability.

4.6 Workload of Virtual Servers and Overloaded Nodes in the two Models

It has been mentioned that the two load balancing models follow a different strategy to reach the same result. The differentiation is based on which entity undertakes the workload of the load balancing. Examining the workload of the overloaded node and the virtual server, it can be analysed if there are risks for virtual servers to fall in overload state due to the load balancing scheme or if this extra workload can not be supported by the overloaded node.

When workload is mentioned, it incorporates not only the messages but the number of processing tasks. Although it is difficult to define the discrete tasks of the nodes, some proportions of workload between virtual servers and overloaded nodes, in the two variations, can be proposed.

The following tasks are defined for the two models:

- **VSS** (3/5 ratio):
  - **Virtual Server**: 1(ULN)+2(VSS_RFH)= **3 workload units.**
  - **Overloaded Node**: 1(VSS_RFH)+2(VSS_RTOL)+2(VSS_CTOL)= **5 workload units.**
- **VSD** (5/2 ratio):
  - **Virtual Server**: 1(ULN)+4(VSD_RFH)= **5 workload units.**
  - **Overloaded Node**: 1(VSD_RFH)+1(VSD_RTOL)= **2 workload units.**

It is important that the counting of tasks and processes is not precise and there is not an exact quantum unit of workload. For example, scanning a list is not always one unit, because the workload of this task may vary according to the size of list, how the list has been implemented, if it is cached and other factors which can affect the quantification. However, most importantly are the ratios, which it is believed, represent the real ratios in the proposed models.

In the simulation, the workload is calculated according to the above definitions. Figure 17 illustrates the results of the two models:

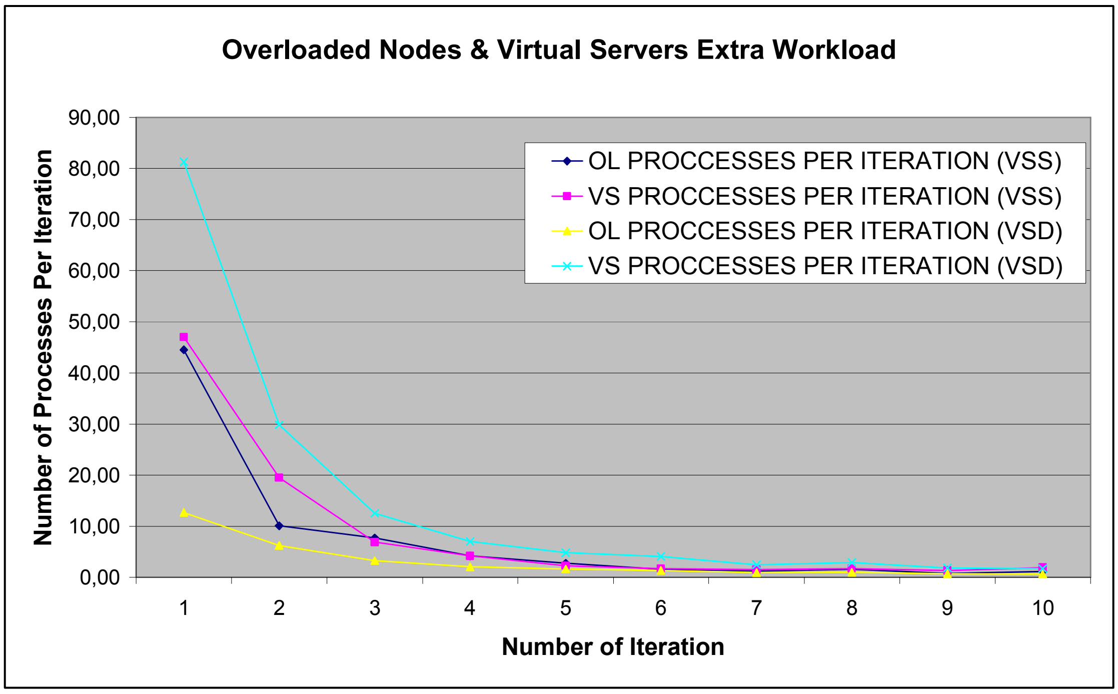


Figure 18: Workload of virtual servers and overloaded nodes in the two load balancing models

Results show that the two models behave differently. The workload in the two entities in VSS does not differ significantly. It is quite the same and it becomes equal rapidly. However; in VSD the workload is uneven. Virtual servers manage most of this and overloaded nodes are quite flexible. Their workload is significantly less than both entities in the VSS model. Furthermore, it seems that the mean value of the workload in the two entities in VSD is the same value of the two entities in VSS (the two lines of VSS are between the two lines of VSD).

The two variations have a powerful combination for establishing dynamically. For example, starting with VSS and if the overloaded nodes can not respond to the load balancing workload, the system can switch to VSD which reduces the workload of overloaded and increases respectively for the virtual server. Then, if virtual servers reach their capacity limits, the system can switch back to VSS.

4.7 Query Success

It is obvious that if there are discarded queries in the network, the query success may be affected because some queries may not reach potential resources. This is related with the TTL value, the number of neighbours in each node and the thresholds for losses. Investigating the query success with the second experimental simulation environment provided the results of the following figure:

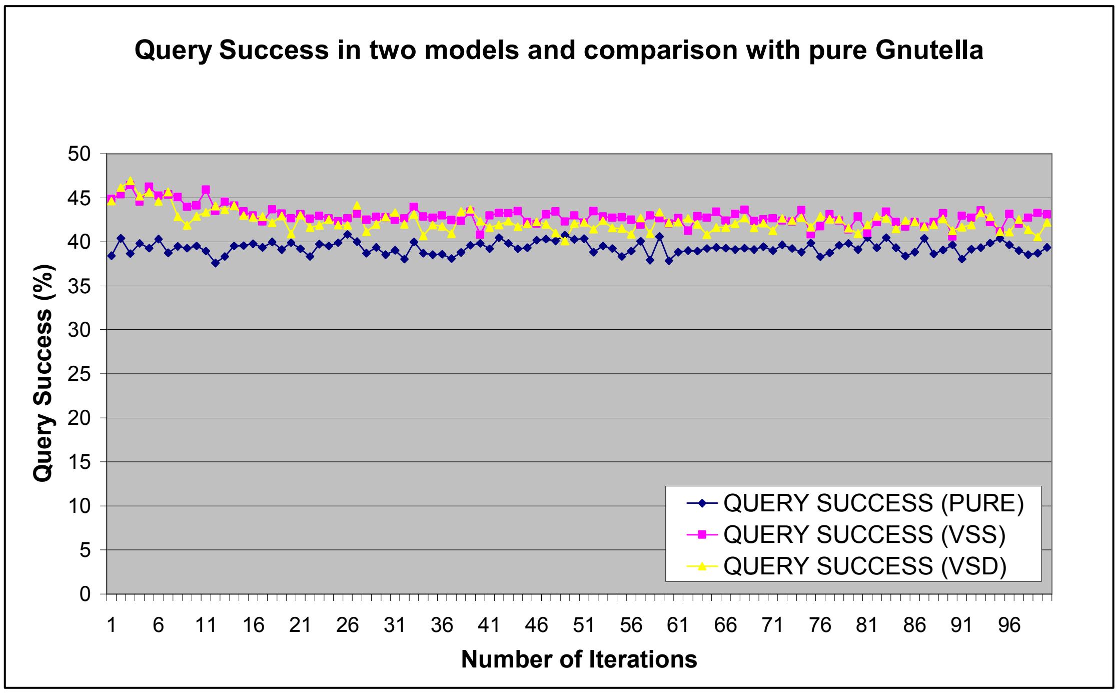


Figure 19: Query success for the two variations and pure Gnutella

The results reveal that there is an improvement in query success. The lost messages reduce the number of query hits. The difference is not high because the flooding technique generates many messages and a query can reach a node from many paths. It is believed that the improvement would be more significant in other searching techniques (random walkers). It must also be added that, in the simulation level, only queries are discarded and not query hits (the focus is in queries).

4.8 Discarded Messages

Most of the experiments conducted under different conditions show one simple thing. In most cases, *there are no discarded messages*. This means that if the thresholds follow the standard deviation concept, there is optimum success without discarded messages compared to pure Gnutella which appears to have lost messages.

However, the second simulation environment is extreme, with nodes that can not process the 5000 queries per iteration. It is a very challenging environment for examining how the load balancing behaves in this extreme condition. Figure 20 illustrates the results:

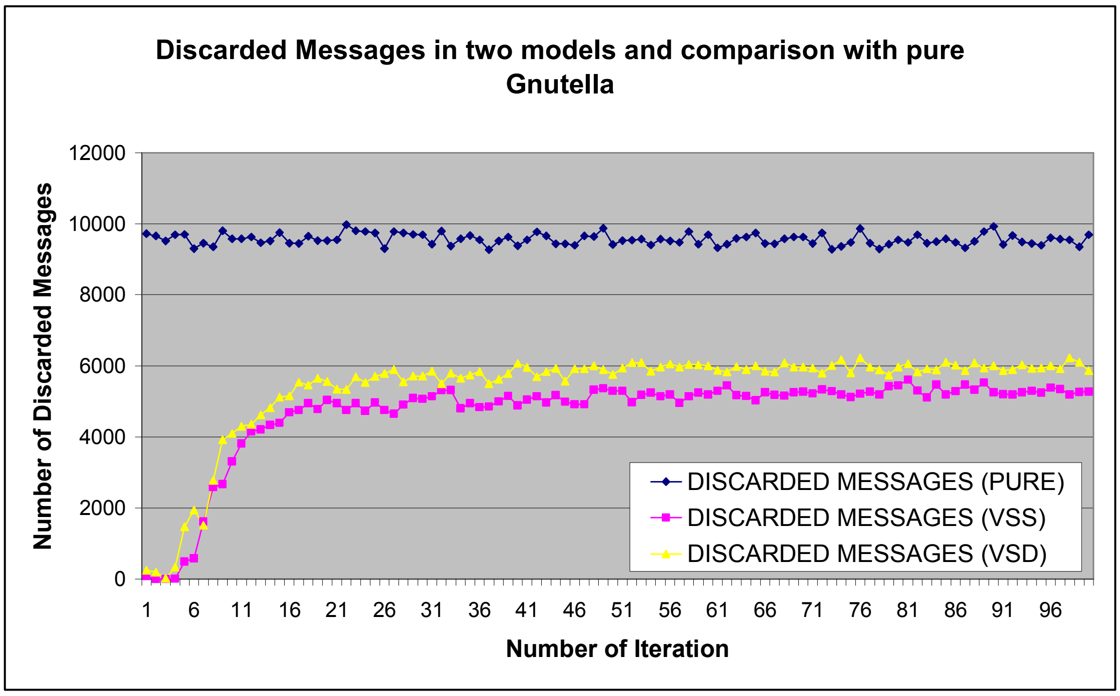


Figure 20: Discarded messages in pure Gnutella and in the two load balancing models

We notice that the system has a perfect performance. However, it can not support so many messages, thus it starts losing messages exponentially until a point where it finds a balance and it loses a constant number of messages. The interesting fact is, this number is much less than the number of discarded messages in Gnutella. The system shows a property of *resistance* in this extreme environment and it succeeds its purpose.

Another interesting thing has been that this experiment needed 100 iterations in order for the system behaviour to be clear.

5. CONCLUSIONS AND FUTURE WORK

5.1 Conclusions and Evaluation towards Autonomic P2P Computing

This work proposed two load balancing models for Gnutella 0.4 network. Gnutella network suffers from high numbers of queries and it seems that nodes may become unable to process and forward the messages. The proposed load balancing scheme focuses on this side of the Gnutella load. The idea is based on some logical movements of the nodes that cause the overload to nodes that are regarded underloaded. This logical movement is supported by virtual servers, peers with a role to manage the load balancing actions.

The proposed models and results show a significant overall improvement. Gnutella network is more robust with the VSS and VSD load balancing. The number of discarded messages is reduced, the query success is affected positively and the load profiles of the nodes reveal increased number of balanced nodes. The most important fact of these improvements is that the load balancing messages (cost) is reduced. Lastly, there is a global improvement in the system due to the significant increase in system availability. The system appears a property of adaptivity and operational balance.

As far as the comparison of the two variations is concerned, they seem to behave similarly. The difference is the workload distribution between the virtual servers and the overloaded nodes. VSS has a closer workload between virtual servers – overloaded nodes. VSD reduces the workload in overloaded nodes but increases it significantly in the virtual servers. The system could take advantage of this fact and establish a dynamic switching model between the two variations regarding the load condition of virtual servers and overloaded nodes.

In section 1.4.3, criteria of self-organisation are defined and the analysis of Gnutella self-organisation is outlined. It is interesting to discuss the changes in self-organisation dynamics that have been introduced by the load balancing model. Table 21 below compares pure Gnutella and Glutella with load balancing according to the self-organisation criteria.

Table 21: Self-Organisation profile of pure Gnutella and Gnutella with load balancing

Self-organisation Criteria	**Pure Gnutella**	**Gnutella with Load Balancing**
Boundaries	No	No
Reproduction	No	Yes
Mutability	No	Yes
Organisation	No	No
Metrics	Partial	Yes
Adaptivity	No	Yes
Feedback	No	Yes
Reduction of Complexity	Yes	Yes
Randomness	No	No
Self-Organised Criticality	No	Partial
Emergence	Yes	Yes

Boundaries remain unspecified. The load balancing model does not introduce control to points where nodes can enter the system.

Load balancing introduces a degree of reproduction and mutability. The structure is influenced and also the relationships. The logical movements that the model proposes affect the relations and the connections of peers. It also affects the number of relations. For example, an overloaded node after load balancing has fewer relationships (fewer nodes send queries to this node).

Organisation is not affected by load balancing. The concept of Gnutella organisation remains the same.

Metrics have been extended because the system can detect perturbations caused by overloading. This additional property benefits Gnutella and for this reason full conformance is attributed to Gnutella (it is not completely full because the model does not cover all types of perturbations, but in order we depict the improvement, full conformance in attributed).

Gnutella with load balancing is regarded more adaptive on the grounds that overloading perturbations are faced. The system is adaptive to the extra load in nodes caused by queries.

Furthermore, load balancing systems provide feedback to Gnutella network. The feedback is both positive and negative. Positive when underloaded nodes advertise themselves to virtual servers. This positive feedback indicates resource discovery. The negative feedback is from the overloaded nodes which ask for help from virtual servers. The system takes actions for this negative feedback and tries to balance the load of nodes. The feedback in both cases is in the form of messages.

Self-organised criticality partially exists in the system due to load balancing. More specifically, the actions of balancing have a global effect to the system. The number of messages is reduced exponentially, although the system continues to remain balance. This critical point reveals a degree of self-organised criticality.

Emergence by load balancing can be noticed when query success and standard deviation is examined in the extreme experimental simulation environment described in the previous section. Calculations show that the standard deviation is reduced although the number of queries remains the same.

Overall, the Gnutella network with load balancing show improved characteristics of self-organisation. Towards autonomic computing, the models support the idea by having partial self-healing attributes, self-configuration and self-optimization. However, it can not be regarded an autonomic computing system because it focuses only on load balancing. The system can receive perturbations by a wide range of other factors. The most important fact is that a single model for load balancing affects positively the effort of more than one "self-actions".

5.2 Future Work

Results give motivation for further work and exploitation of model dynamics. The following subsections illustrate some interesting future focus on this work:

5.2.1 More Advanced and Heterogeneous Simulations

More advanced simulation can provide more reliable results within different simulation environments. For example, the load thresholds are the same for all the nodes, something that is not realistic but experimentally accepted. Virtual server is expected to have more load capacity and the load in nodes can result from different statistical distributions. The same concept can be applied to queries generations. Instead of random generation, each peer could send a query with a probability, which can be calculated through different statistical distributions. This also concerns the files, the number of neighbours and other parameters of the network.

There are also other assumptions that can be omitted. Lost messages, node arrivals and leavings, latency and more. The more functionality supported, the better and more reliable results can be generated.

5.2.2 Detailed Study and Dynamic Utilization of the Load Balancing Variations

The results showed that the two variations have both, advantages and disadvantages. A further study, with better definitions of the processes and the ratios of the workload of virtual servers and overloaded nodes is required. This can provide a dynamic establishment of both variations which can make virtual servers and overloaded nodes more robust.

5.2.3 Build of a more Advanced and Generic Simulator

A better simulator can provide a more powerful environment for experiments. Section 5.2.1 mentions some improvements that can be incorporated in the simulator. In general, a generic simulator, with options for creating various networks with various searching techniques and methods - which attribute to the networks a range of information - can be a flexible and powerful research tool.

3-D visualisation is also under construction and can be a special element for network management in P2P networks.

5.2.4 Study and Development of the Role of Virtual Servers

There is one question that must be answered in this work; what is the number of virtual servers that can provide the best results? Does this number vary during the network changes? It is certain that it must vary. It is believed that the design of a model that could create dynamically virtual servers, according to the network state, is the idea that can complete this work. More research is needed and detailed studies of the behaviour of virtual servers.

5.2.5 Development of a Prototype for Real Testing and Evaluation

As the research is evolved, the findings must be confirmed through a prototype which will test the model in reality. A P2P client for file sharing can be a first prototype for extracting the final results of the model.

REFERENCES

(2000) *The Gnutella Protocol Specification v0.4 , Document Revision 1.2*. Available: http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf [Accessed: August 01, 2007].

Klingberg, T. and Manfredi, R., (2002) *Gnutella Protocol Specification v0.6*. Available: http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html [Accessed: August 01, 2007].

Ripeanu, M., (2001) 'Peer-to-Peer Architecture Case Study: Gnutella Network*' in Proceedings of the First International Conference on Peer-to-Peer Computing, IEEE*.

Ritter, J., (2001) *Why Gnutella can't scale. No, really*. Available: http://www.darkridge.com/~jpr5/doc/gnutella.html [Accessed: August 01, 2007].

Zeinalipour, D., (2002) *Exploiting the Security Weaknesses of the Gnutella Protocol.* Available: http://www.cs.ucr.edu/~csyiazti/courses/cs260-2/project/html/index.html [Accessed: August 01, 2007].

Fan, R. and Chung, K., (1997) 'Spectral graph theory', *Regional conference series in mathematics,. Published for the Conference Board of the mathematical sciences by the American Mathematical Society, Providence, R.I.*, **no. 92.**

Fan, R. and Chung, K., (1994) '26 cm', *CBMS Conference on Recent Advances in Spectral Graph Theory held at California State University at Fresno, 6-1,* **T.p. verso**.

Ripeanu, M., (2001) 'Peer-to-Peer Architecture Case Study: Gnutella Network', *University of Chicago Technical Report*.

Adamic, L., Lukose, R., Puniyani, A. and Huberman, B., (2001) 'Search in power-law networks*'*, *Phys. Rev.* **E 64,** *046135*.

Aiello, W., Chung, F. and Lu, L., (2000) 'A random graph model for massive graphs' *in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing,* ACM Press, pp. 171–180.

Lu, Q., Cao, P., Cohen, E., Li, K. and Shenker, S., (2000) 'Search and replication in unstructured peer-to-peer networks' *in Proceedings of the 16th international conference on Supercomputing*, ACM Press, pp. 84–95.

Azzouna, N. and Guillemin, F., (2004) 'Experimental analysis of the impact of peer-to-peer applications on traffic in commercial IP network' *in European transactions on Telecommunications: Special issue on P2P networking and P2P services*, ETT, 15/6:511 – 522

Steinmentz, R. and Wehrle, K., (2004), 'Peer-to-Peer Networking & Computing' *in Informatic-Spektrum*, Springer, Heidelberg, **27(1)**:51-54.

Napster web site. Available: http://www.napster.com/ [Accessed: August 01, 2007].

Balakrishman, H., Kaashoek, M., Krger, D., Morris, R. and Stoica, I., (2003) 'Looking up Data in P2P Systems', *Communication of the ACM*, **46(2).**

Rowstron, A. and Druschel, P., (2001) 'Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems' *in IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pp.329-350, Heidelberg, Germany, Springer.

Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Shenker, S., (2001) 'A Scalable Content-Addresable Network' *in SIGCOMM*, pp. 161-172, ACM Press.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F. and Balakrishnan, H., (2001) 'Chord: A Scalable Lookup Peer-to-Peer Service for Internet Applications' *in Proceedings of the 2001 ACM SIGCOMM Conference.*

Watts, D. J. and Strogatz, S. H., (1998) '*Collective dynamics of 'small world' netwo*rks' *in Nature*, **393(6684)**:440-442.

Barabasi, A. L. and Albert, R., (1999) '*Emergence of Scaling in Random Networks*' *in Science*, **286**:509-512.

Rivest, R., (1992) 'The MD5 Message-Digest Algorithm' *in RFC*, **1321.**

Singla, A. and Rohrs, C., (2002) 'Ultrapeers; another step towards Gnutella scalability' *in Gnutella developer forum.*

Tamassia, R., Goodrich, M. T., Vismara, L., and Handy, M., (2005) *An Overview of JDSL 2.0, the Data Structures Library in Java*. Available: http://www.jdsl.org/other_modules/overview/overview.pdf [Accessed: August 01, 2007].

Aristotle, (1957) 'Metaphysica' *in W. Jaeger*, editor, Metaphysica, Oxford University Press.

Bar-Yam, Y., (1997) 'Dynamics of Complex Systems', *Westview Press.*

Maturana, H. R. and Varela, F. J. (1980) 'Autopoiesis and Cognition: The Realization of the Living' *in D. Reidel*, Dordrecht, Holland.

Steinmetz, R. and Wehrle, K., (2005) *Peer-to-Peer Systems and Applications.* Springer-Verlag Berlin Heidelberg.

Karger, D. R. and Ruhl, M., (2006) 'Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems', *Springer*, New York.

Godfrey, B., Lakshminarayanan, K., Surana, S., Karp, R. and Stoica, I., (2006) 'Load Balancing in Dynamic Structured P2P Systems' *in Elsevier Science.*

Zhu, Y. and Hu, H., (2005) 'Efficient, Proximity-Aware Load Balancing for Structured P2P Systems' *in IEEE Transactions on Parallel and Distributed Systems.*

Bienkowski, M., Korzeniowski, M. and Heide, F.M. auf der, (2005) 'Dynamic Load Balancing in Distributed Hash Tables*' in Springer,* Berlin.

Aberer, K. Datta, A. and Hauswirth, M., (2005) 'Multifaceted Simultaneous Load Balancing in DHT-Based P2P Systems: A New Game with Old Balls and Bins' *in Springer.*

Exarchakos, G., Salter, J. and Antonopoulos, N., (2006) 'Semantic Cooperation and Node Sharing among P2P Networks' *in Proc. of 6th International Network Conference*, , pp. 11-19, Plymouth.

Suri, S., Toth, C. D. and Zhou, Y., (2004) 'Uncoordinated Load Balancing and Congestion Games in P2P Systems' *in Springer*, Verlag.

Papadimitriou, C. H., (2001) 'Algorithms, Games, and the Internet' *in Proceedings of the thirty-third annual ACM Symposium on Theory of Computing.*

Uchida, M., Ohnishi, K., Ichikawa, K., Tsuru, M. and Oie, Y., (2006) 'Dynamic Storage Load Balancing with Analogy to Thermal Diffusion for P2P File Sharing' *in proceedings of Interdisciplinary Systems Approach in Performance Evaluation and Design of Computer and Communication Systems.*

Eichhorn, D., (2006) *NATaWare Java Framework*. Available: http://sourceforge.net/projects/nataware [Accessed: August 01, 2007].

IBM, (2001) *Autonomic Computing from IBM*. Available: http://www.research.ibm.com/autonomic/overview/solution.html, [Accessed: August 01, 2007].

Definition of Autonomic Computing. Available: http://www.webopedia.com/TERM/A/autonomic_computing.html [Accessed: August 01, 2007].

Kephart, J. O. and Chess, D. M. (2003) *The Vision of Autonomic Computing*. Available: www.research.ibm.com/autonomic/research/papers/AC_Vision_Computer_Jan_2003.pdf [Accessed: August 01, 2007].

Sterritt, R. and Hinchey, M. (2005) '*Autonomic Computing. Panacea or Poppycock?.* Available: http://ieeexplore.ieee.org/iel5/9677/30561/01409959.pdf [Accessed: August 01, 2007].

Li, Z. J. and Liao, M. H., (2005) 'Modelling Load Balancing in Heterogeneous P2P Systems' *in Science Publications*.

Pournaras, E., (2007) 'V-Agents: Software Agents for Resources Discovery in Unstructured P2P Networks' *in Master Coursework for "Software Agents" module*, Internet Computing Course, University of Surrey.

Ting, N. S. and Deters, R. (2003) ‘3LS – A peer-to-peer Network Simulator’ *in Proceeding of the 3rd International Conference on Peer-to-Peer Computing*

Almaer, D., (2004) *Another Java-C++ Benchmarking.* Available: http://www.theserverside.com/news/thread.tss?thread_id=26634 [Accessed: August 01, 2007].

Oeztunali, S., Rusitschka, S. and Southall, A. (2006) ‘MULTILAYER GNUTELLA: P2P Resource Sharing with an Efficient Flexible Multi-Keyword Search Facility’ *in TechRepublic*