JYVÄSKYLÄ STUDIES IN COMPUTING
166

Annemari Soranto

Interest-Based Topology Management in Unstructured Peer-to-Peer Networks

JYVÄSKYLÄN YLIOPISTO

JYVÄSKYLÄ STUDIES IN COMPUTING 166

Annemari Soranto

Interest-Based Topology Management in Unstructured Peer-to-Peer Networks

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella
julkisesti tarkastettavaksi yliopiston Agora-rakennuksen Alfa-salissa
joulukuun 20. päivänä 2012 kello 12.

Academic dissertation to be publicly discussed, by permission of
the Faculty of Information Technology of the University of Jyväskylä,
in building Agora, Alfa-hall, on December 20, 2012 at 12 o'clock noon.

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2012

Interest-Based Topology Management in Unstructured Peer-to-Peer Networks

JYVÄSKYLÄ STUDIES IN COMPUTING 166

Annemari Soranto

Interest-Based Topology Management in Unstructured Peer-to-Peer Networks

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2012

Editors
Timo Männikkö
Department of Mathematical Information Technology, University of Jyväskylä
Pekka Olsbo, Ville Korkiakangas
Publishing Unit, University Library of Jyväskylä

URN:ISBN:978-951-39-5023-1
ISBN 978-951-39-5023-1 (PDF)

ISBN 978-951-39-5022-4 (nid.)
ISSN 1456-5390

Jyväskylä University Printing House, Jyväskylä 2012

ABSTRACT

Soranto, Annemari
Interest-Based Topology Management in Unstructured Peer-to-Peer Networks
Jyväskylä: University of Jyväskylä, 2012, 100 p. (+included articles)
(Jyväskylä Studies in Computing
ISSN 1456-5390; 166)
ISBN 978-951-39-5022-4 (nid.)
ISBN 978-951-39-5023-1 (PDF)
Finnish summary
Diss.

Peer-to-peer networks consist of autonomous nodes that communicate with each other to share and exploit resources in totally decentralized manner. Current P2P applications focus mainly on providing file storage and sharing that scales to the demand without heavy investment to centralized coordination. Unstructured P2P networks allow varied resource search queries based on keywords but suffer from scalability of used search algorithms. Delivering messages in the physical network is fast, but the processing of queries and messages at the application level requires capacity and causes delays, limiting the usability and scalability of P2P applications.

This thesis studies algorithms both for self-organizing and managing logical topology and for performing efficient resource discovery in unstructured peer-to-peer networks using only local information that nodes can collect while within the network. Topology algorithms aim to organize the overlay topology so that peers can find the resources they need close to them in logical topology. Thus the scope of the search queries could be reduced and results would be found with smaller overhead of query traffic.

In addition to topology management algorithms intended for general use this thesis also studies the combination of topology management with search algorithms that are tailored to function optimally in specific environments or use cases.

The work covers the art of simulating P2P networks, various approaches to topology management and adaptive search methods, some theoretical approximations of ideal search efficiency, and a systematic experiment to compare topology management and adaptive search methods in a simple, controllable case.

Keywords: peer-to-peer networks, P2P, overlay topology, topology management, self-organizing, resource discovery

Author's address

Annemari Soranto
Department of Mathematical Information Technology
University of Jyväskylä
Finland
annemari.k.soranto@jyu.fi

Supervisors

Prof. Timo Tiihonen
Department of Mathematical Information Technology
University of Jyväskylä
Finland

Dr. Jarkko Vuori
Helsinki Metropolia University of Applied Sciences
Finland

Reviewers

Assoc. Prof. Maria Papadopouli
Department of Computer Science
University of Crete
Greece

Docent, Dr. Sergey Balandin
Fruct Oy
Finland

Opponent

Prof. Jarmo Harju
Department of Communications Engineering
Tampere University of Technology
Finland

ACKNOWLEDGEMENTS

This thesis is a result of a long process involving collaboration with many persons over a long period of time. Everything started in the P2P research group headed by Professor Jarkko Vuori. The articles included in this thesis originate from this period. I would like to thank Jarkko Vuori for his inspiring guidance, Mikko Vapa for several years of continued collaboration, and all other co-authors of the articles, especially Niko Kotilainen and Teemu Keltanen. Working with you was instructive but also fun!

In 2007, the research appeared to have reached a dead end, and despite several ideas and plans for advancement, the work did not lead to any progress. I want to thank Professor Vagan Terziyan and Jani Kurhinen, both who provided instruction and guidance during this difficult phase of the process that occurred before Professor Timo Tiihonen became my supervisor. As a simulation expert, Timo provided valuable input from a different perspective, although it took a while before mutual understanding of the research topic was reached. The simulation described in this introduction was designed and established under his supervision, mostly along with other work assignments, which proved to be quite a challenging and slowly-progressing task. I am grateful to Timo for his patient guidance and the instruction he provided during the writing process. I also thank all colleagues who have encouraged me and created a pleasant working environment during these years.

I would like to thank Associate Professor Maria Papadopouli and Docent, Dr. Sergey Balandin for reviewing this thesis and for their valuable comments.

This study was financially supported by GETA (Graduate School in Electronics, Telecommunications and Automation), COMAS (Graduate School in Computing and Mathematical Sciences) and the Emil Aaltonen Foundation.

Finally, I want to thank my family for encouraging me, especially Mikko for his patience, love, motivation, and understanding. Last but not least, I thank my dear aunt, who always believed in me, but passed away before this thesis was finished.

Jyväskylä 12.12.2012
Annemari Soranto

LIST OF FIGURES

LIST OF TABLES

CONTENTS

LIST OF INCLUDED ARTICLES

PI M. Vapa, N. Kotilainen, A. Auvinen, H. Kainulainen, J. Vuori. Resource Discovery in P2P Networks Using Evolutionary Neural Networks. *Proceedings of the IEEE International Conference on Advances in Intelligent Systems - Theory and Applications*, 2004.

PII A. Auvinen, M. Vapa, M. Weber, N. Kotilainen, J. Vuori. Chedar: Peer-to-Peer Middleware. *Proceedings of the IEEE 20th International Parallel and Distributed Processing Symposium*, 2006.

PIII N. Kotilainen, M. Vapa, A. Auvinen, M. Weber, J. Vuori. Peer-to-Peer Studio - Monitoring, Controlling and Visualisation Tool for Peer-to-Peer Networks Research. *Proceedings of the ACM International Workshop on Performance Monitoring, Measurement and Evaluation of Heterogeneous Wireless and Wired Networks*, 2006.

PIV N. Kotilainen, M. Vapa, A. Auvinen, T. Keltanen, J. Vuori. P2PRealm - Peer-to-Peer Network Simulator. *Proceedings of the IEEE 11th International Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks, pp. 93-99*, 2006.

PV A. Auvinen, M. Vapa, M. Weber, N. Kotilainen, J. Vuori. New Topology Management Algorithms for Unstructured Peer-to-Peer Networks. *Proceedings of the IEEE Second International Conference on Internet and Web Applications and Services*, 2007. Best Paper Award.

PVI A. Auvinen, T. Keltanen, M. Vapa. Topology Management in Unstructured P2P Networks using Neural Networks. *Proceedings of the IEEE Congress on Evolutionary Computation*, 2007.

PVII M. Vapa, A. Auvinen, Y. Ivanchenko, N. Kotilainen, J. Vuori. Optimal Resource Discovery Paths of Gnutella2. *Proceedings of the IEEE* 22nd *International Conference on Advanced Information Networking and Applications, pp. 546-553,* 2008.

1 INTRODUCTION

The development of technology has put peer-to-peer networks (P2P) back on the map in last ten years. Although P2P networks have gained a lot of publicity since 2000, this technology is not new. In the peer-to-peer network all nodes have equal roles: nodes both provide resources to other nodes and consume resources from other nodes. This idea of equality was already behind the ARPANET, which was invented in the late '60s. The emergence of applications such as e-mail and the WWW in the '90s changed the network to client–server architecture comprised of a small amount of powerful and dedicated servers and a huge amount of clients using services from these servers. Since then, the capacity of low-cost home computers and telecommunication has grown to enable the development of different P2P applications.

The best known application area in P2P networks is content distribution. The file distribution systems, such as Napster and Gnutella, brought P2P into the public eye. In addition to file services, the applications may utilize the unused CPU time to perform tasks that would otherwise require expensive supercomputers. The P2P networks can be used also for communications; for example the popular internet phone application Skype is based on the peer-to-peer model. The most popular and widely-used networks and applications have been developed for open and voluntary use and require no service commitment from individual nodes, only the installation of the application. However, P2P systems can also be made available to restricted environments and be developed for the specific needs of organizations, particularly for applications where resilience and scalability are needed.

P2P network consists of computers, called nodes or peers, and the connections between them. The nodes and connections together form the logical to-

pology of the P2P network on top of the physical network. The P2P networks can be divided into several groups based on the logical topology of the network. In unstructured P2P networks, the node's position in the network is not predefined, whereas structured P2P networks have specific rules dictating where the node joining the network is placed. Furthermore, unstructured networks can be pure or hybrid and nodes may construct one or more layers in the topology. The hybrid network has a special server called a broker, which includes the index of the resources.

The different topologies have their own advantages and disadvantages and properties affecting their efficiency. In particular, the topology affects the usability of different search mechanisms, and searching for the resources is considered to be the main bottleneck in the scalability of unstructured P2P networks.

This thesis concentrates on the unstructured pure P2P networks where the simplest search algorithm is flooding breadth-first search (BFS), where nodes forward the query to their neighbors and possible replies are sent back to the querier along the same route by which the queries were sent.

The performance analysis of P2P networks is far from simple. The real networks are hard to monitor as they have no central control, the networks are big, and the traffic and resources are quite diverse. Also, the expectations for performance may differ even within the same network. Design and optimization of networks is further complicated by the fact that the observable properties of the networks are mainly emergent and global whereas the design and control is local. So, it is quite understandable that the literature on the analysis of P2P networks is quite diverse and lacking commonly-used benchmarks.

The big question driving this work has been to understand how the topology of P2P networks can be adapted to the needs of the users of the network. For such a broad question no general solution is to be expected. So our focus is in studying the related sub-problems and tools that will be needed when answering this question in a concrete situation. The essential sub-problems consist of the following:

1. How to make controlled experiments on P2P networks and protocols?
2. How far is the real observed search performance of simple P2P protocols from theoretical limits?
3. How much the performance can be improved by careful tuning of the parameters?
4. How to analyze systematically the performance of a topology management algorithm for a given setup?

1.1 Structure of the Thesis

The thesis is structured so that the first five chapters present background information on the study of topology management in unstructured P2P networks,

with a focus on the first three research questions. The actual contribution related to these first chapters is documented in the attached papers I through VI. As the papers have been written to obey strict page limits, they cannot cover all aspects of a systematic analysis. Chapter 6 aims to expand on this by documenting a detailed analysis of a simple experiment.

Chapter 2 includes definitions of peer-to-peer networks and presents how they are classified based on the level of centralization and the overlay structure. The developed tools and topology management algorithms presented in this thesis are designed for pure unstructured P2P networks, which are introduced in greater detail by using the Gnutella protocol, for example.

Chapter 3 and Chapter 4 concentrate on two aspects affecting the efficiency of pure unstructured peer-to-peer networks, namely: search algorithm and topology. Chapter 3 presents the problem of the resource discovery in pure unstructured networks and the most common search algorithms, which are used when studying the search and topology management algorithms. Typically, search algorithms depend on rules and parameters defined in the implementation phase. The NeuroSearch algorithm presented in paper PI uses neural networks to decide which neighboring parameters are relevant when forwarding the query. Neither NeuroSearch nor any other developed search algorithm using local information has optimal performance. To facilitate the efficiency evaluation of the algorithms, paper VII presents an upper bound for efficiency to which the algorithms can be compared to.

In the unstructured P2P network, nodes may manage their neighborhood by adding or dropping the neighbors to improve efficiency. Chapter 4 introduces the principle of topology management and its different aspects. Also, the solutions developed to improve the topology are presented and compared to the ones developed in this thesis. The topology management may be based on the physical parameters of the network or the information collected from the logical networks. The two algorithms presented in papers PII and PV use local information about neighbors that nodes have collected. These algorithms improve the efficiency of the search algorithms by organizing the topology in such a way that nodes with similar interests are situated close to one another. The information used in the management is rather complex and has several parameter values that impact the results and performance.

As the real networks cannot be analyzed and optimized easily, some artificial models must be used. It is common practice for a preliminary evaluation of a technology to explore its behavior under well-understood conditions and simple models. After that large scale simulation with more realistic conditions can be designed and performed. Thus, a well-designed simulator can be a useful tool. The P2PRealm simulator presented in the paper PIV was developed for the study of search and topology management algorithms, especially those using neural networks. Systematic simulation of P2P networks is a challenge due to the lack of well-defined benchmarks, test-suites, and scenarios. The parameters defining the simulation environment and their possible values are introduced in Chapter 5.

For any algorithm, the performance depends not only on the values of the algorithm's own parameters, but also on the properties and parameters of the networks used. Chapter 6 focuses on making simulation studies of the algorithms in simple and controlled networks rather than in snapshots of real-world P2P networks to be better able to control the effects of algorithms on the behavior of the networks. Thanks to simple and well-controlled setup new information about the qualitative behavior of the studied topology management algorithm could be obtained. The tendency to create star-like topologies and to create unconnected networks while optimizing the short range search efficiency was observed and could be partially controlled. The managed network provided also a better environment to adaptive search algorithms but this effect may be largely related to the artificial, evenly distributed resource allocation used in the simulation. Hence no strong conclusions about the performance in real networks can be made. As the setup is fully artificial and has little in common with any real use case of P2P networks, the most valuable findings are not related to the actual performance of different methods but to various aspects and features of the methods and simulation experiments that are revealed through systematic analysis in a simple setup.

Chapter 7 summarizes the contribution of the thesis. The appendix documents the results of different simulation tests in greater detail.

2 PEER-TO-PEER NETWORKS

The peer-to-peer networks (P2P) are a class of distributed systems. The P2P networks can be further divided into several subclasses based on the properties of their logical topologies. This chapter introduces different P2P networks and discusses advantages and disadvantages of peer-to-peer architecture.

2.1 Distributed Systems

In the literature, several definitions are presented for a distributed system. (Coulouris, et al, 2005) define it from a system infrastructure view by saying that a "distributed system is a system in which hardware and software components located at networked computers communicate and coordinate their actions only by passing messages". (Tanenbaum, et al, 2002) has included a user's view into their definition of the distributed system. According to their definition, "a distributed system is a collection of independent computers that appears to its users as a single coherent system." Thus the system consists of several computers that cooperate together with messages for providing users a service. In spite of being distributed, the system functions as if it was running on one computer, and the user does not have to be aware of the details of distribution.

Currently, the most common distributed network architecture is the client-server architecture. The system consists of one server and several clients. The architecture is centralized: the server's role is to provide a service to clients, who request services from the server and wait for replies from it. Usually the

server has high performance compared to the clients. The system based on client-server architecture is easy to manage because of the centralized server, but at the same time, the server is also a possible bottleneck of the system and the biggest obstacle to scalability of the system. Expensive clusters, high-capacity communication channels, and storage are typically needed to maintain good service in client-server architecture.

Master-slave is a distributed architecture that is used mostly for computational tasks. The architecture consists of one master unit and several other units called slaves. The master node has some computational task, which it divides into independent subtasks. The master sends subtasks to slaves that process the data. After processing the data, the slaves send results back to the master. In the master-slave architecture, the master controls the commands. Thus the architecture has a centralized unit that may become the bottleneck of the system. The architecture is well suited for situations where the tasks can be easily divided into smaller tasks and the amount of slaves does not increase infinitely.

The third common architecture for distributed systems is a peer-to-peer (P2P) architecture that in the purest form is totally decentralized. In the P2P architecture, all computers can take on either role, working as masters or clients giving out tasks, or as slaves or servers providing work and resources for the others. Thus the architecture is scalable in terms of amount of computers, but finding a certain resource and controlling the system is a challenge.

These three architectures differ from each other with respect to the centralization of resources or commands. In the client-server architecture, resources are centralized and provided by a server, but commands are decentralized and executed by clients. In the master-slave architecture, commands are centralized, coming from the master, but the resources are decentralized in slaves. Peer-to-peer architecture is totally decentralized: both resources and commands are decentralized. The client-server and P2P architectures differ also with respect to the location of service providers. In the client-server network, the servers are situated in the middle of the network whereas in peer-to-peer applications the utilized resources are available at the edges of the Internet (Coulouris, et al, 2005).

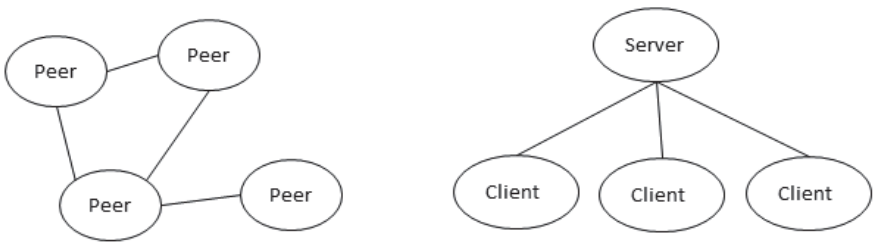

FIGURE 1 Peer-to-peer and traditional client-server architectures.

2.2 P2P Network

In the peer-to-peer network, all computers or peers (also called nodes) in the network have equal roles. The main idea is that each peer in the peer-to-peer network may both provide resources to other peers in the network and request resources from the others. According to (Schollmeier, 2001) "the peer-to-peer network is a distributed network architecture where the participants, i.e. peers, share a part of their own hardware resources which are necessary to provide service and content offered by the network. Participants are accessible by other peers directly, without passing intermediary entities. The participants of such a network are thus resource providers as well as resource requestors." In the peer-to-peer networks, peers communicate by message passing and thus it fits to the definition of distributed systems presented in the previous chapter (Ciglaric & Vidram, 2002).

(Androutsellis-Theotokis & Spinellis, 2004) add the concept of self-organizing into the definition. The nodes self-organize into network topologies to share resources, but those topologies are also "adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority." Thus in a peer-to-peer network, dynamicity of the networks is considered normal behavior.

Schollmeier's definition of peer-to-peer networks can be expanded for defining a pure peer-to-peer network. In a pure peer-to-peer network, any single, arbitrary chosen peer can be removed from the network without affecting the service (Schollmeier, 2001). Gnutella is the best known and most used pure peer-to-peer protocol.

Another category of peer-to-peer networks is the hybrid peer-to-peer networks (Schollmeier, 2001). In the hybrid peer-to-peer network, the resource requests are sent to a central entity usually called an intermediate or broker, which replies by sending information about nodes possessing the requested resource. The requesting peer selects from the reply message a node that it wants to utilize as a resource and establishes a connection to that node. Thus in the hybrid peer-to-peer network, the searching is like client-server, but the actual service, for example downloading a file, is pure peer-to-peer. The popular music file-sharing application Napster used hybrid peer-to-peer architecture. Hybrid peer-to-peer systems are easy to implement and searching is simple, but they have the same disadvantages as the client-server system due to centralized architecture. The intermediate may become a bottleneck of the system, restricting the scalability.

In addition to the traditional pure and hybrid peer-to-peer networks, there is a third class called partially centralized architecture. In a partially centralized system, all peers are not equal, but there is a subset of peers that are called supernodes or ultrapeers. Other participating peers are called leaf peers. Ultrapeers and leaf peers form a two-tier topology so that ultapeers form a top-

level topology to which leaf peers are connected through ultrapeers. An ultrapeer knows the resources of the leaf peers connected to it, so it can process the resource search queries on behalf of its leaf peers. Ultrapeers have to have sufficient capacity, but they are dynamically replaceable. (Androutsellis-Theotokis & Spinellis, 2004), (Stutzbach & Rejaie, 2008), (Wang, et al, 2007), (Yang & Garcia-Molina, 2003). The Kazaa file-sharing application has the two-tier topology.

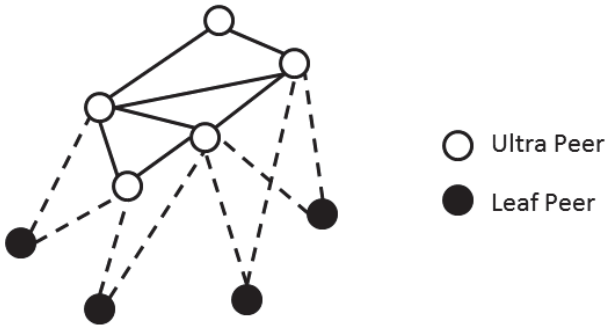

FIGURE 2 Topology of partially centralized architecture.

2.3 Unstructured and Structured P2P

Peer-to-peer networks can also be categorized into unstructured and structured peer-to-peer networks, based on the evolution of logical topology.

In a structured network, the overlay topology is controlled and there is mapping between resources and the peers providing them. Thus the resource queries can be routed efficiently and it is guaranteed that the resource can be found if it exists in the network. The disadvantage of this overlay architecture is that maintaining the topology needs work and it is not suitable for networks where nodes are joining and leaving the network at a very high rate (Lv, et al, 2002B). Examples of structured peer-to-peer systems are Freenet, Chord (Stoica, et al, 2001), CAN (Ratsanamy, et al, 2001) and Tapestry (Zhao, et al, 2004).

A P2P network is unstructured when a peer's position in the network is not predefined. A peer can join the network by connecting to any node in the network and freely select its neighboring peers, so the overlay topology is totally nondeterministic (Androutsellis-Theotokis & Spinellis, 2004). In an unstructured P2P network, resource discovery burdens the network because it has to be done by propagating a resource query from peer to peer. An advantage is that the search key is not restricted to the predefined keywords used on the structured networks. Unstructured P2P is also suitable for networks where the net-

work is highly transient because maintaining the topology does not need extra work.

This thesis concentrates on pure unstructured peer-to-peer networks because of their flexibility for different kinds of applications and ability to scale.

2.4 Gnutella Protocol

The most traditional pure unstructured peer-to-peer architecture uses the so-called Gnutella protocol. Gnutella protocol was introduced in 2000 and since that it has been used for several file sharing applications.

Because of the pure nature, the joining to the network is challenging. When the peer wants to join the Gnutella network it needs to locate one node in the network. The protocol does not define how this should be done, but it can be done for example through a specific web page listing nodes. Rejoining the network is easier because each node keeps an index of the neighbors it has. When a node wants to rejoin the network after disconnection, it checks its index of neighbors from a previous session and tries to reestablish connections.

Gnutella uses a propagation mechanism called flooding for searching nodes and resources. Gnutella uses two types of messages. Ping/Pong messages are for finding new nodes and Query/Query Hits messages for locating resources. Gnutella node sends a Ping message to the node to which it is connected. The node replies to the message by a Pong message and includes in the message its connection information, IP and port number, and information about its resources. The node also forwards the Ping message to all of its neighbors. Using the Ping message, the entering node gains knowledge of the nodes close to it and thus finds new nodes to connect with.

Locating resources with a Query message is also implemented with a flooding algorithm. A node sends a Query message to all its neighbors, who then forward the Query message to their neighbors. If the node receiving the Query message has the queried resource, it sends back as a response the so-called Query Hit message. A Query Hit message includes IP and port number of the sender and the number of resources that match the query. All response messages, Pong and Query Hits are propagated back to the initiator by the same route that the Ping or Query message came to the responder.

Gnutella protocol has two ways to restrict the flooding algorithms. The Ping and Query messages have a time-to-live (TTL) value that is used to limit the searchable area. Each copy of the messages has a TTL value that is decreased by one every time a query arrives at a node. When the value of TTL is zero, the message is not forwarded but discarded. The nodes also keep a history of the messages they have forwarded. When a node initiates a query, it assigns to it a unique ID. All nodes propagating queries add the copy of the ID and the node from where the query arrived into its local history. If a node receives a query already included in its history, it discards it. The information

about the sender is used when the response message is delivered back to the initiator.

When the node that initiated a Query message receives Query Hit replies, it selects the nodes from which it wants to use the resource. The initiator establishes a connection to the replier and uses the resource. In the case of file distribution the peers replicate files when downloading them from a peer. Thus the principle is that all downloaded resources will be available also in the peer that has downloaded them.

2.4.1 Two-Tier Gnutella

Modern Gnutella utilizes a two-tier overlay topology, described in chapter 2.3, which enables more-efficient search mechanisms. When connecting to ultrapeers, a leaf peer uploads a set of hashes of its resource keys (filenames) to the ultrapeer. Ultrapeers manage the queries for their leaves so that queries are propagated in top-level topology only. (Stutzbach & Rejaie, 2008), (Rasti, et al, 2006), (Wang, et al, 2007)

Modern Gnutella uses Dynamic Query Protocol, which is totally controlled by ultrapeers (Fisk 2003). The challenge of the unstructured P2P network is that the popularity of specific resources in the network is not known and it is hard to select a proper TTL value. In the Dynamic Query Protocol, a node uses a probe query to gather information about popularity of the searched resource. The probe queries are sent to specific amount of neighbors and the horizon is varied based on the received replies.

In Gnutella protocol, a leaf node is allowed to become an ultrapeer if it cannot find enough ultrapeers than can accept an additional leaf. Proper balance between ultrapeers and leaf peers and well-connected top-level overlay is important in order to provide scaling and short pair-wise distances. Two popular implementations of two-tier Gnutella protocol are LimeWare and BearShare.

2.5 Advantages and Disadvantages of Pure Unstructured P2P Networks

The pure unstructured peer-to-peer system has several advantages when compared to the traditional client-server architecture, such as high availability, scalability and fault-tolerance (Oram, 2001). The lack of central management is a consequence of the decentralized architecture. The scalability of search algorithm and freeriding are the most recognized problems in the P2P networks.

(DePaoli & Mariani, 2004) define scalability as the degree of adaptability a system exhibits with respect to increasing load situations. P2P systems are highly dynamic and unpredictable in size, topology and activity. Size of the network may grow or shrink due to connecting and disconnecting peers, which also affects the topology of the system. As (Ge, et al, 2003) states, the increasing population of peers not only increases workload, but also increases the capacity

to serve the workload. Peer-to-peer networks are scalable regarding amount of peers and resources, but the used flooding search algorithm is not scalable. This barrier should be overcome by using more efficient search algorithms.

P2P has also lower costs because peer-to-peer networks typically utilize unused capacity of the computers and resources already available in the network, i.e. the disk space of home computers (Coulouris, et al, 2005). Thus the system does not need any dedicated, expensive servers, space or administrators.

The fault-tolerance of the P2P network can be derived from the decentralized nature and equality of the nodes. The availability of a P2P system does not depend on any specific node, and leaving of a node is assumed to be a normal behavior of a P2P system. The homogenous distribution of connections provides no reference points for attackers, but if the nodes are organizing as in the small-world network model, the attacks can be targeted to peers with numerous connections. Even in this case, elimination of one node is not enough to paralyze the system, but the attacking needs to be more systematic. (DePaoli & Mariani, 2004)

P2P networks provide users high availability through replication. Because peers may join and leave the network whenever they want, availability of a certain resource in a certain node is not guaranteed. But since a downloaded resource is usually replicated into the downloading peer, popular data can be found from several peers in the networks. There are also peers that do not replicate data or do not share any resources with other peers but instead search and download resources that others are sharing. Those peers are called freeriders, or freeloaders, and there exists several studies concerning impact of freeriding to the P2P systems performance.

(Adar & Huberman, 2000) present that if a system demands that a peer has to share a resource to be able to download a resource, this may cause peers to share randomly generated files or other data that has no value to the system. The authors have studied the user traffic on Gnutella and found the significant amount of freeriding in the system. According to the authors, approximately 63% of the peers share no files and the top 20 % shares 98% of files. A result of significant freeriding might be that decentralized peer-to-peer network becomes more centralized if there are just few peers that provide resources to others acting just as clients. Those peers sharing popular content might become overloaded and thus become bottlenecks of the networks (Ramaswamy & Liu, 2003B). Significant freeriding also affects the amount of files in the system and the number of popular files may even decrease, which reduces a user's interest in using the system (Ramaswany & Liu, 2003). Because of the freeriders, the search horizon should be expanded so that it reaches enough nodes providing resources, which generates more traffic in the network.

Generally freeriding is thought as a disadvantage of the decentralized P2P systems but according to a study of (Ge, et al, 2003) pure unstructured networks, the limited flooding algorithm can tolerate some number for freeloaders without much degradation in the query success probability and the overall system

throughput may increase, but performance drops sharply if this number is too large.

In addition to the scalability of the used search algorithm and freeriders, the lack of global knowledge of the network is a recognized problem in P2P networks. In the client-server system, a resource placement is easy to manage, but in the P2P system it is difficult to place data across peers so that workload would be balanced in the network. Lack of global knowledge causes also challenges for management of replicas and resource discovery.

3 RESOURCE DISCOVERY IN UNSTRUCTURED P2P

Resource discovery is a challenge in the pure unstructured network where resources are distributed across the network and location of the resources is not known by the nodes. The use of flooding mechanisms for resource discovery is the main barrier for the scalability of unstructured peer-to-peer networks.

(Tsoumakos & Roussopoulos, 2003A) categorize search methods as either blind or informed methods. Blind search algorithms function without any information about resource location whereas informed methods use some kind of index to assist with the search. Blind methods are neither accurate nor efficient. So the goal is to maximize the number of found resources and minimize the number of messages needed to achieve that.

This chapter briefly describes the most common blind and informed search methods.

3.1 Evaluating the Search

(Lv, et al, 2002A) have categorized metrics used in the evaluation of search algorithms for user aspects and load aspects. User aspects include success, i.e. the probability of finding the queried object before the search terminates, and number of hops needed to find the resource. Load aspects include for example number of search messages per node, number of visited nodes, and percentage of message duplication.

Search algorithms are evaluated mainly based on the success and efficiency of the query. The search is successful if the node receives a reply or replies to

the request. One common criterion is that a search is successful if the request results to at least one reply, but the criterion may demand also more replies. Success rate describes the proportion of successful searches to total searches, and it is used as a measure of estimating the performance of the search algorithm. Efficiency presents proportion of received replies to the generated queries. Usually efficiency of the search algorithm is compared to the efficiencies of other search algorithms, but it can also be evaluated against theoretical target value as presented in the paper VII.

Other metrics used when estimating resource discovery algorithms are cost and quality of results (Yang & Garcia-Molina, 2002). The used processing power and bandwidth are main costs when processing the query. When estimating quality of the results, the number of results, satisfaction, and time to satisfaction are taken into account. The query is satisfied if at the very least a defined amount of results is found. Time to satisfaction is the time that has passed since the user sent the query until he got the last result needed to achieve the satisfaction.

3.2 Blind Search Methods

The most common blind search algorithm is flooding Breadth first search (BFS) algorithm, which is described in chapter 2.4. Advantages of BFS are that it is easy to implement and is able to find a queried resource if it exists in the horizon defined by the TTL value. So if only the number of results is used as the quality metric, the BFS is ideal. The disadvantage is, as mentioned earlier, the scalability problem. Because the replication ratio of the resource is not known, the TTL value has to be high to ensure success. At the same time, the number of duplicate queries also increases. Those produce extra work in the network and decrease the effectiveness. (Tsoumakos & Roussopoulos, 2003A), (Tsoumakos & Roussopoulos, 2006), (Yang & Garcia-Molina, 2002), (Lv, et al, 2002A)

There are blind algorithms derived or extended from the BFS algorithm, for example iterative deepening. The Iterative Deepening algorithm consecutively sends BFS searches and increases the depth of the query in each iteration until the resource is found or the defined maximum depth is reached. (Yang and Garcia-Molina, 2002) suggest that when satisfaction is the used metric, the iterative deepening algorithm should be used. The disadvantage of the search algorithm is that each time the depth is increased, the same query message is also forwarded again to the same neighbors.

The random walk algorithm randomly selects a neighbor node to which the query is sent. This is repeated in every node receiving a query message until the query is satisfied. When only one walker is used, the user may notice delay. This can be avoided by using multiple walkers. The initiator of the query sends k query messages, which utilize random walk to k randomly selected neighbors. Usually k is between 16 and 64. The random walk algorithm needs a TTL value and periodical checking from the initiator to terminate the query. Compared to

BFS, the random walk algorithm increases the amount of hops a bit but reduces message overhead significantly, and it is a more scalable search algorithm (Lv, et al, 2002A). The disadvantage of the algorithm is variability of success rates and number of hits, which depend on topology. (Lv, et al, 2002A), (Tsoumakos & Roussopoulos, 2006). One specific random walk algorithm is Highest Degree Search, which selects the neighbor that has the highest degree, i.e., highest amount of neighbors, and that has not yet been visited (Adamic, 2001). The algorithm is especially suitable for the networks with power-law distribution.

3.3 Informed Search Methods

Informed search algorithms utilize the same kind of information as the interest-based topology management. There are several algorithms derived or extended from the BFS algorithm that can be classified as informed search methods (Tsoumakos & Roussopoulos, 2006), (Kalogeraki, et al, 2002). Intelligent-BFS algorithm is an extension of Modified-BFS, where nodes select part of the neighbors to forward the query to. The benefit is that it reduces the amount of generated messages but covers still a large number of peers. When using Intelligent-BFS search, each node stores information about recent answers for sent resource queries and uses that information when deciding where to forward the resource query. The node keeps a profile for each of its neighbor nodes and stores in the profile the resource replies returned by the neighbor. When deciding where to forward a query message, the node uses a query similarity metric to find out the similar queries in profiles and rank the neighbors. Intelligent-BFS has a high success rate and it does not produce any overhead when neighbors are joining or leaving. The disadvantage is that intelligent-BFS still produces a large amount of messages. (Kalogeraki, et al, 2002)

Adaptive Probabilistic Search (APS) collects information about previous searches to define forwarding probabilities for its neighbors. Searching is done by using k random walkers. The local index is updated based on the success of a walker. If the walker succeeds, the relative probabilities of the nodes on the walker's path are increased. If the walker fails, the probabilities are decreased. The advantage of APS algorithm is bandwidth-efficiency. (Zhang et al, 2007), (Tsoumakos & Roussopoulos, 2006)

In Routing Indices (RI) method, each node keeps an index for every neighbor it has. The index contains information how many resources belonging to each resource categories can be found from that neighbor direction. The information is not gathered from the replies, but the nodes provide that information to each other. There is no information on which node will provide a resource, just a direction. Searching using Routing Indices is very bandwidth-efficient, but maintenance of routing indices uses flooding, which creates load in dynamic networks. (Crespo & Garcia-Molina, 2002), (Tsoumakos & Roussopoulos, 2003A), (Tsoumakos & Roussopoulos, 2006)

(Crespo & Garcia-Molina, 2002) have presented the compound and the hop-count routing indices. In compound RI (CRI) the neighbors are ranked based on their goodness, i.e. the estimated number of resources that may be found from the neighbor's direction. The query is forwarded to the neighbor with the highest goodness value. If a node has no neighbors, the query is forwarded back to the neighbor from which the node received the query, which in turn selects the second best neighbor and forwards the query to that one. In hop-count Routing Indices, aggregated RIs are saved for each hop up to the maximum number of hops. Goodness is defined as the ratio between the number of resources available through that neighbor and the number of messages required to get those resources. This method has higher storage and transmission cost than the CRI has.

Directed BFS (DBFS) algorithm (Yang & Garcia-Molina, 2002) uses also information on neighbors and replies. The querier sends query only to a subset of neighbors and selects the neighbors based on the defined heuristic. The criterion may be for example the amount of replies or closeness of replier. The nodes receiving the query forward it using the BFS algorithm. The algorithm has lower cost than BFS without significant loss in quality of results.

4 TOPOLOGY MANAGEMENT IN UNSTRUCTURED P2P

This chapter presents how the efficiency and scalability of unstructured peer-to-peer networks can be improved by reconstructing the overlay topology. The topology can be presented as a graph, and concepts and parameters from the graph can be used to characterize P2P network topologies. The concept of topology management is defined and the various methods to optimize the topology are presented.

4.1 Characteristics of Overlay Topologies

Overlay topology of the peer-to-peer network can be presented as an undirected graph G = < V, E >, where V is a set of nodes in the network and E is a set of edges representing connections between the nodes. The topology management affects connections in the overlay by changing them, and the effect of the management can be evaluated using properties derived from the graph of the overlay. Networks have some simple global properties that can be used to characterize the topology. The global parameters that affect the search performance the most are the diameter of the networks, the average distance between the nodes, and the average degree of the nodes. All of those have influence on the needed TTL value, the amount of hops, i.e., the traffic the query is causing.

The distance of two nodes i and j, d_{ij}, is the number of edges (i.e. connections) of the shortest path between the nodes. Diameter of the network is the length of the maximum distance in the network. Characteristic path length, L,

measures the typical distance in the network, i.e., the average shortest path lengths from a node to all other nodes in the network. $L = 1/n \sum_{i}^{n} 1/(n-1) \sum_{j}^{n} d_{ij}$ (Watts & Strogatz, 1998), (Newman, 2003), (Xie, et al, 2007). Using diameter of the graph the maximum number of hops needed for reach all the nodes in the network can be defined and thus the maximum search path can be calculated. The node degree, *k,* is the number of connections the node is maintaining, i.e., the number of the node's neighbors. A more informative way to characterize node degrees is node degree distribution, which can be formed by using Probability Distribution Function (PDF) *p(k) = n(k)/n*, where *n(k)* is the total number of nodes with degree value *k* (Xie, et al, 2007).

Peer-to-peer networks can be categorized according to graph properties such as grid, random, scale-free or small-world networks. Grids are always constructed and not suitable to be maintained in the dynamic environments where nodes are joining and leaving the networks whenever they want. A two-dimensional grid graph is an *m*n* graph that is the graph cartesian product of path graphs on *m* and *n* vertices. In the regular *m*m* grid, the nodes have an almost equal position and the degree distribution is uniform. The regular torus is a periodic grid, where all nodes have the same degree of 4 and the diameter is *m/2*. Hypercube is a special grid used usually in parallel computing. It is a graph, joining the vertices of a *n*-dimensional hypercube along the edges. It has 2^n nodes of degree *n* and the diameter is *n*.

Random graph is a graph in which edges are placed between nodes randomly. The number of possible edges is *n(n-1)/2*, where *n* is the number of nodes. Each pair of nodes is connected with probability *p*. In random graphs, the degree of the node follows binomial distribution, or a Poisson distribution, in the limit of large *n*. (Newman, 2003)

In real networks, also in Internet or P2P networks, degree distribution does not usually follow the Poisson distribution but a power law $P(k) \sim k^{-\gamma}$. Networks with power law degree distribution are called scale-free networks. According to (Albert & Barabasi, 2002) power law is based on growth and preferential attachment. Preferential attachment means that probability of connecting to a node depends on node's degree. Power law distributed network can be formed by algorithm of the Barabasi-Albert method: In each time step, one new node is added to network. Probability that node will be connected to node *i*: $\Pi(k_i) = \frac{k_i}{\sum_j k_j}$, where k_i is degree of node *i*. (Albert & Barabasi, 2002), (Newman, 2003)

Interconnectivity of the node's neighbors is measured by a local clustering coefficient, which is the average probability for any two nodes sharing a neighbor to be connected. The local clustering coefficient is calculated by $C_i = \frac{2E_i}{k_i(k_i - 1)}$ where k_i is degree of node *i* and E_i denote actually existed edges between the node's neighbors. The clustering coefficient of the network, *C*, is

the average of the local clustering coefficients $C = \frac{1}{n}\sum_{1}^{n} C_i$. In random graph $C = p$. (Albert & Barabasi, 2002), (Xie, et al, 2007). Networks, which have small characteristic path lengths similar to random graphs, but a larger clustering coefficient, are called small-world networks. (Watts & Strogatz, 1998)

4.2 Topology Management

Topology management involves constructing the overlay topology in a self-organizing way. It affects the overlay topology of the network by defining principles for nodes choosing their neighbors and thus making the network more efficient for the given purpose. Topology management includes two processes that together determine the topology. First is the process of inserting new nodes to the network. The second process includes methods to define when and how to make changes, and add or remove connections. The purpose of the topology management is to maintain the neighborhood of the node so that neighbors are the best nodes available to a specific node according to some defined criteria. The rules may be defined by the developer of the algorithm or the algorithms may also adapt some principles from the existing models, such as Schelling's model (Singh & Haahr, 2007), club concept (Asvanund, et al, 2003), (Idris & Altmann, 2006), prisoner's dilemma (Hales, 2005), (Lai, et al, 2003) or social networks (Yang, et al, 2008).

4.2.1 Characteristics of Neighbors

Because there is no global knowledge of the network, nodes make decisions regarding their neighbors based on local knowledge only. Nodes to which a node is connected are called neighbors. Other nodes, that a node is aware of, are called candidate nodes. Nodes decide on taking candidate nodes as their neighbors based on their knowledge about some property or properties of the candidates. Nodes either collect this information locally from the candidate nodes or utilize information that the neighbors or the potential neighbors are providing. The topology management approaches are more or less characterized by these choices: what information to collect, what criteria to apply to the information, and what action to take.

One popular criterion is that the node's neighbors should be the nodes which are the closest nodes in the physical network. The purpose is to match logical topology to the underlying physical topology and thus prevent a situation where the flooded message goes through the same path several times in the physical level, although it is handled at most once by the node in the application level. Thus the purpose is to decrease the amount of traffic in the physical network and to decrease the delay. However, in the logical topology, this solution may increase the amount of hops needed to find a searched resource. If the used time-to-live values of the flooded messages need to be increased to find a

certain amount of resources, it increases the traffic both on logical and physical levels. The methods work well in the network where data is largely replicated. (Agrawal & Casanova, 2003), (Alima, et al, 2002), (Hu and Sereviratne, 2003), (Liu, et al, 2003), (Liu, et al, 2004A), (Liu, et al, 2004B), (Liu, et al, 2005A), (Liu, et al, 2005B), (Lu, et al, 2005), (Massoulie, et al, 2003), (Ni & Liu, 2004), (Ratsanamy, et al, 2002), (Wan, et al, 2005), (Xiao, et al, 2005), (Zhang, et al, 2004)

A good neighbor can be defined also as a node providing service that the other node needs. This service may be the capacity to handle the received messages, it may be the amount of resources or replies the node is providing, or it may be the quality of the provided resources. This criterion for goodness is very similar to the one used in node selection, i.e., when a node is selecting where it will finally download the found resource from. Issues taken into account may be physical properties of the connection or historical information about the node and resources it has provided (Abraham, et al, 2007), (Berstein, et al, 2003), (Habib and Chuang, 2006), (Liu, et al, 2008), (Liu, et al, 2010), (Sun, et al, 2007).

When capacity information, delivered usually by the candidate node, is used as criteria, the purpose is to manage the load in the network. Good neighbor is a node which has the capacity to handle the messages it receives from the node. This prevents nodes from overloading and decreases the processing delay in the network. (Lv, et al, 2002B), (Chawathe, et al, 2003)

The criterion may also be the similarity of interests. When the good node is defined as a node providing replies for the resource requests, the purpose is to put nodes with a similar interest close to each other in the logical network. This clustering may be established based on the information gathered from the received replies for the sent queries or it may involve semantic metadata describing resources or interests. In the first case, the idea is that nodes that have provided resources to the node will provide results also for the future queries. In the second case, there have to be some global and predefined rules for describing, classifying, and matching resources and measuring similarities (Handurukande, et al, 2004), (Khambatti, et al, 2004), (Broekstra, et al, 2003), (Sakaryan & Unger, 2003A and 2003B), (Asvanund, et al, 2003), (Idris & Altmann, 2006), (Voulgaris, et al, 2004), (Ng & Sia, 2002), (Crespo & Garcia-Molina, 2005), (Kojima, 2003). This is difficult in the system with a distributed nature, but when a node has neighbors with similar interests, it receives required resources closer and thus the value of time-to-live in the queries may be decreased. This decreases traffic in the network.

When the information used for topology management is delivered by the neighbors, or neighbor candidates, the advantage is that all nodes have the same information about the same neighbor. The disadvantage is that nodes are not making decisions based on the experience or observation about candidates but they have to rely on the information delivered by the neighbors and trust that information. Nodes may promise resource replies for the queries, but the actual received service does not match with the promise or has low quality. Thus one criterion for the goodness of the neighbor is trust, which evaluates also the received service (Niu, et al, 2007).

The age of the node may be used as criteria when selecting the neighbors. The goodness depends on the lifetime of the nodes (Bustamante & Qiao, 2004). The underlying assumption is that the longer the node has been in the network, the longer it will be in the network and provide resources, and thus the better neighbor it is. When neighbors are staying in the network, there is no need for searching and establishing new connections, which decreases the traffic generated by connection management.

4.2.2 Topology Management Methods

Topology management has to define rules for when and how the topology is managed to optimize the node's neighborhood. The methods rank the nodes based on the defined criteria and use ranking information when selecting new neighbors or removing existing neighbors. A topology management method includes the actions to reconstruct the overlay, initiator of the actions, and the extent/scope of the actions.

The methods may use one or several criteria mentioned in the previous chapter to rank the neighbor candidates. Most methods have a fixed optimization target, such as to decrease the delay, and thus they are utilizing a predefined criterion, but there are also more general solutions without set criteria. When the criteria is not defined in the algorithm, the user/programmer may select one or several properties of the node as criteria depending on the application area (Alima, et al, 2002), (Singh & Haahr, 2007), (Ramaswamy, et al, 2005), or the general method is adapting into the scenario and selects the important properties of the node as criteria (Iles & Deugo, 2002), (Iles & Deugo, 2003).

Topology management method needs an initiator of the management actions. Usual triggers are situations such as when a node is rejoining the network, when a node is overloading (Cooper & Garcia-Molina, 2005), or when the goodness of the neighborhood is below the limit, like the searching time is longer than expected (Sakaryan, et al, 2003A), (Sakaryan, et al, 2003B). The node may also have some extra capacity to deliver and thus apply for topology management.

Topology management is usually clustering nodes based on the defined criteria, for example nodes with a similar interest should be in the same cluster and thus close to each other. In a pure P2P network, all nodes are in principle equal, and it is up to the topology management method to self-organize nodes as clusters and create the cluster structure (Singh & Haahr, 2007), (Crespo & Garcia-Molina, 2005). In two-tier networks, there is a predefined hierarchical structure, i.e., super-peer architecture, and thus there already exist the super-peers or cluster heads which are defining the characteristic of a cluster belonging to it (Ramaswamy, et al, 2003A), (Ramaswamy, et al, 2005). The latter case is very common in the networks utilizing semantic information (Löser, et al, 2003), (Nejdl, et al, 2003), (Airiau, et al, 2006), (Asvanund, et al, 2003), (Idris & Altmann, 2006). If the topology is defined as hierarchical, the purpose of the topology management is to define how nodes select the super-node which to connect

to (Lo, et al, 2005), or to define how a node will become a super-node (Zheng, et al, 2005), (Lo, et al, 2005).

Hierarchical structure may also have several tiers, i.e., topology has several layers and the cluster head in the upper layer serves one or more clusters in the lower layer (Srivatsa, et al, 2006), (Yang & Chen, 2008). Thus topology management methods may reconstruct the connections in the current overlay network or construct another overlay or overlays on top of the original and use those for different purposes (Ng & Sia2002), (Crespo & Garcia-Molina, 2005) or define different type of connections for different purpose (Cooper & Garcia-Molina, 2005).

Thus when topology management achieves cluster structure, the question is whether there exists the cluster heads. If the originators exist, the topology management determines to which originator the node should connect, and how the originator accepts a request or prevents overloading. The big question concerning the topology management is then whether the originator has criteria for the nodes belonging to that cluster or whether those criteria are evolving based on the nodes joining the cluster. This also affects the adaptation of the method. If it demands some special structure of the network, it is not easily included in the existing system. The developed method may be independent of the used search algorithm or it might be developed for the certain search algorithm, which is common in the semantic P2P systems. (Nejdl, et al, 2003), (Voulgaris, et al, 2004)

Actual changing of topology consists of adding and removing connections. Additions and removals of connections are naturally needed when peers join or leave the network. When a node joins the network, it needs to add a connection to a node in the network. When a node leaves the network, it may inform the neighbors of it and disconnect all connections to neighbors. The topology adaptation can appear also in other situations when a node wants to change its neighborhood to a better one. The node may also replace the connection when it adds a new connection to a node and removes one existing connection. Thus some developed methods have only heuristics for situations when a node joins or disconnects, but some have heuristics to adapt the network more actively.

Topology management produces load in the network because establishing and dropping connections requires messages to be sent. Thus the produced load should be taken into account when evaluating the gained efficiency. In the ideal situation, the network changes a lot only in the beginning and reaches a stable state where just a few changes take place. Nodes in the network can also have constraints concerning the capacity that topology management should take into account to guarantee the functionality of the network. Some developed topology management methods control the degree of a node by defining a maximum number for neighbors, but this is not an optimal solution. The amount of neighbors should be restricted based on the real capacity of a node, and the node adjusts its neighbors based on its capacity to handle the messages the neighbors are sending to it.

The scope of the topology management is usually the node's neighborhood, i.e., nodes one-hop or two-hops away. Thus a node constructs its own local connections using information on the neighbors and their neighbors. There are also the topology management methods where changing connections has larger impact, for example when two nodes are replacing one of the neighbors by connecting to each other. In this situation, the dropped neighbors of the nodes are connected to each other (Sun & Garcia-Molina, 2004). Thus a node's decision to replace a node with another also affects other nodes' neighborhoods and it causes the nodes in these other neighborhoods to connect "against their will".

4.3 Interest-Based Topology Management Approaches

Interest-based topology management methods cluster nodes according to interests. The goal is that nodes with a similar interest will be close to each other and thus a node will receive the resources it requires close to it and the search path can therefore be decreased, which also decreases the load to the network. The characteristic used to illustrate the interest is usually the amount of resource replies the node has received to the queries it has sent to the network (Ramanathan, et al, 2002), (Ghanea-Hercock, et al, 2006), (Ng, et al, 2002), (Sripanidkulchai, et al, 2003). Some studies use information about satisfactory transactions and unsatisfactory transactions for interest-based topology management (Condie, et al, 2004). A node sending a reply to the query shares the same interest with a querying node. Thus the node uses history information about the queries and replies to predict the need in the future. The expectation is that the nodes providing results are most likely providing results also for the subsequent queries.

When a node uses resource replies as criteria, it collects information locally. So another node's information on the same node is not identical. Thus a node rates other nodes from its point of view, i.e., how beneficial the other nodes are to it. Topology management methods need also to take care that decisions based on local goals and locally collected information are also beneficial to the P2P network in general.

From resource replies the node collects information about neighbors, neighbors' neighbors and the indirect nodes, which sent resource replies further than two hops away from the node. Neighbors' neighbors and the indirect nodes construct the group of candidate nodes. The node saves the amount of replies the nodes are providing to it, and in some methods also the amount of replies the neighbors (Ramanathan, et al, 2002) and neighbors' neighbors are forwarding to the node.

Information about interests can be used in several ways to manage the topology. When a node is rejoining the network, it may try to connect to the nodes it has found useful in the past. Information can be used also when the node has extra capacity, i.e., too few connections, or when it is overloading, i.e.,

it has too many connections. In the first case, the node adds a new neighbor with similar interest, and in the latter case the node removes a neighbor that seems to be less relevant. There might also be a trigger that activates the topology management in the case when a node wants to improve its neighborhood. Usually this is related to the time, and nodes periodically evaluate the goodness of their neighborhood (Ramanathan, et al, 2002).

Basic actions applied in the interest-based methods are adding and removing nodes. A node adds as a new neighbor the node which has the highest reply value. It removes the neighbor that has the lowest value. All other methods are variations of the basic actions. If a node has extra capacity left, it might just add a neighbor without any special conditions, or for example, if percentage of reply messages of a known candidate is greater than its neighbor with the smallest value, a connection to that candidate node is established (Ramanathan, et al, 2002). The method may include several additions or removals in one operation. A node may for example remove all neighbors which have a value lower than some defined limit (Ghanea-Hercock, et al, 2006).

A topology method may also include both adding a neighbor and removing a neighbor and thus replacing some existing neighbor by some candidate node. Replacing is utilized when a node has no extra capacity, but it finds a better neighbor for it. A node may replace a neighbor if a candidate node has provided more replies than it, or it may replace the neighbor with lowest value with a candidate that has more replies than any of the neighbors (Condie, et al, 2004). Overtaking [PII] is one special case of replacing - a node replaces a neighbor by a neighbor's neighbor and thus overtakes an existing neighbor. In addition to constructing the existing connections in the overlay network, the method may create and utilize a different kind of shortcut connections on top of the logical topology and use those connections for special purposes (Sripanidkulchai, et al, 2003). Using shortcut connections between nodes with similar interests, the node keeps the heterogeneity of its neighborhood, but the searching can be focused using the shortcuts and does not load the whole neighborhood. Topology algorithms manage the shortcut connections between nodes.

Most of the interest-based methods using resource reply information do not specify the used search algorithm and are thus independent of the used search algorithm and easily adapted.

TABLE 1 Interest-based topology management algorithms.

Algorithm	Used Information	Methods	Search Algorithm
Ramanathan, et al, 2002	Received resource replies, both sent and forwarded by neighbors, collected locally	Adding	Any
Condie, et al, 2004	Received resource replies, collected locally	Adding replacing	BFS, Any
Ghanea-Hercock, et al, 2006		Adding removing	
Ng, et al, 2002		Adding	Any
Sripanidkulchai, et al, 2003		Shortcut links on top of the overlay	Algorithm using shortcut links
Auvinen	Received resource replies, both sent and forwarded by neighbors, collected locally	Adding removing overtaking	Any

The interest-based method developed by the author uses received resource replies when evaluating the neighborhood and consists of four algorithms for adding a neighbor, removing a neighbor, traffic estimation, and overtaking. A node in the network evaluates its neighbors based on the resource replies received from and relayed by them. These replies together form the goodness value of the neighbor. A node saves information about neighbors and neighbor candidates, which are neighbors' neighbors and other nodes which have replied to the node's queries. A good neighbor is a neighbor that provides or delivers resource replies to the node.

Based on the capacity of the node, it adds or removes the neighbors. If the node has free capacity available, it tries to add a new neighbor, and if it has too much traffic it drops one neighbor. If the node wants to add a new neighbor, it searches potential candidates from the history information where it saves information about all candidates. If the node succeeds in establishing a new connection, it is satisfied, otherwise it expands the search area. The node has two main sources for information when it searches and selects a node to reconnect: how many resources a node has provided (or forwarded) in the past and how long time ago there has been a connection, if ever. First, the node searches candidates among the nodes which have delivered reply messages to it but which have not been its neighbor in the defined time. Thus the node does not add connection to a node that it has just dropped. Then it expands the search to the candidates that do not have any goodness value. Second expansion is done to the nodes that have no information about replies and that have not been the node's neighbors before. In practice, this means that candidates have been neighbor's neighbors that have not replied or relayed any replies to the node. Finally, if the node does not have any neighbor, it searches a node to connect among the candidates, which have information about replies. Then, it might add a connection to a node it has just dropped. If a node needs to drop a neigh-

bor it selects the worst neighbor, i.e., a neighbor that has the lowest goodness value.

In addition to the traffic estimation algorithm that utilizes adding and removing algorithms, there is also an algorithm for overtaking. Overtaking algorithm moves, step by step, a node closer to those nodes that provide replies to it. When the querier node receives replies to a query, it calculates for each neighbor and neighbor's neighbors the amount of replies that these have relayed to it. Then, it checks whether there is a neighbor's neighbor, whose proportion of the total amount of replies received through the neighbor is more than a defined overtaking threshold. If that kind of neighbor's neighbor is found, the node tries to establish a connection to it and removes the connection to the current neighbor.

5 SIMULATION OF PEER-TO-PEER NETWORKS

This chapter summarizes the components that need to be modeled when simulating topology management in unstructured peer-to-peer networks.

When studying the effect of the topology management or search algorithms, the peer-to-peer network where the algorithms can be evaluated is required. Setting up the real peer-to-peer network is expensive and it is hard to control when the size of the network increases. Because of the distributed nature of the P2P networks, individual nodes do not have global knowledge of the networks and thus information about the real networks has to be collected separately, typically using crawlers. A crawler is a real implementation of a P2P node, which is put into an existing network to collect data. Each crawler has only a local view of the network, but several crawlers in the same network can provide a larger view (Stutzbach & Rejaie, 2005A), (Stutzbach & Rejaie, 2005B). In the case of studying topology management algorithms under development, the use of crawlers is not appropriate because a network containing the studied algorithm is needed. The only methods providing a global view of the network are emulators and simulators.

Another possibility to study P2P networks is to use emulators. An emulator contains the implementation of a single node, but one computer may contain several copies of the emulator. The message passing is done using the network layer, which increases the time needed to run the test cases but gives detailed data on the actual traffic in the network.

The third method is a simulator. Simulation provides a reliable, easily managed, and repeatable environment for the topology management research, and it is the most used method in P2P studies. When simulating the P2P network, the whole network is modeled in a single computer. Thus the simulation

imitates the real P2P network without investments for large amount of computers and network connections. Messages between the simulated nodes are handled with local data structures and do not have to be delivered on the network layer, which consequently increases the speed. The drawback of emulators and simulators is that the real delays originating from the distances of the P2P nodes and effects of users actions cannot be taken into account as such but have to be modeled.

Parameters needed in the peer-to-peer simulator models are related to the nodes, resources, and queries. These parameters affect also the simulation results of the topology management algorithms. Starting topology defines how many nodes there are in the network and how they are connected to each other. The total amount of resources, resource popularity, and distribution of resources among the nodes need to be defined. The third group of parameters forms a query model describing the amount of queries and distribution of queries into nodes. Node distribution, resource distribution, and query distribution can be correlated or independent of each other. The simulation may also include the dynamicity of the nodes, i.e., nodes are leaving and entering the network.

5.1 Initial Network

The emergence of the network can be modeled by some mechanism of adding or removing the nodes, or the simulation may utilize typical existing networks that are generated either randomly or deterministically to the specific form. The information of typical networks can be collected from the existing P2P networks or the network used in simulation can be constructed based on artificial algorithms. In the first case the data is usually collected with crawlers which attain only a partial view of the network and because of the dynamic nature of P2P there are errors in the collected data.

The parameters defining the initial peer-to-peer network in the simulation are the number of nodes and distribution of connections between the nodes. The most common networks in the P2P studies are power-law networks, random networks, regular networks or GnutellaII-like two tier networks. Realistic simulations consist of tens of thousands of nodes, but small simulations are run in networks with only hundreds or tens of nodes. In the papers PV, PVI and PVII the simulations were run in random networks generated by Erdös-Renyi model (Albert & Barabasi, 2002). Power-law networks using Barabasi-Albert method (Albert & Barabasi, 2002) was used as initial network in PI, PVI and PVII. The topology studies in papers PVI used also grid as initial topology and the latest paper PVII studied search also in GnutellaII network.

5.2 Resources

In addition to the starting network topology, the resources also have significant influence on the simulation results and search efficiency. The resources may be anything that can be distributed in the peer-to-peer network. It may be service, which the nodes are providing to other nodes, or it may be a concrete downloadable item, i.e., a file, which is copied and transferred from a node to another. The service distributed in the peer-to-peer network may be for example computing power, file storage or printer service which is located in the providing node where nodes utilize this service.

There are three aspects concerning resources. The first aspect is the popularity of resources, i.e., amount of specific resource in the network. The second aspect is how resources are distributed to the nodes in the network. Of course, the total amount of resources in proportion to the amount of nodes is one parameter in the simulation. The third aspect is whether the resources can be classified into interest groups or not.

These three aspects have to be modeled in the simulations. Each aspect needs to have a defined distribution from where the values are randomly selected. The simplest distribution is uniform distribution. The distributions derived from the existing peer-to-peer simulations are usually non-uniform. The real resources are not uniformly distributed: some resources are more popular than others and some nodes provide more resources than other nodes. The distributions in real content sets vary in different applications and networks and may be available only as inaccurate snapshots (Cooper, 2004). Thus, there do not exist any specific general and justifiable distribution for peer-to-peer simulations or not even a generally accepted benchmark data.

The amount of different resources has to be modeled in the simulation. If popularity distribution is uniform, every resource has the same number of copies and every resource has same probability to be found. The distribution may also be non-uniform when some resources are more popular than others and those resources are easier to find than the less popular resources. The most used non-uniform distribution in peer-to-peer simulations is Zipf (Lv, et al, 2002A), (Tsoumakos & Roussopoulos 2003A), (Tsoumakos & Roussopoulos 2003B), (Liu, et al, 2005B), (Kojima, 2003), (Cooper, 2004),(Schlosser, et al, 2002) (Condie, et al, 2004), (Srivatsa, et al, 2006), which is a power-law type distribution and derived from the Gnutella simulations. Although the accuracy of those simulations may be questioned, the distribution of resource popularity in real networks is non-uniform rather than uniform.

In addition to the amount of different resources, the distribution of resources in each node also has to be modeled. The simplest case is that every node has the same amount of resources when the used distribution is uniform. In the real public peer-to-peer networks, this is rare and usually the resources are not uniformly distributed, but some nodes provide more resources than others. For example, the 80/20 biased distribution derived from the Gnutella

simulations distributes resources in a way that 80% of the document results arrive from the 20% of the nodes (Crespo & Garcia-Molina, 2002), (Srivatsa, et al, 2006). Another solution for uniform distribution is to take into account the capacities of the nodes and distribute the resources proportional to those (Lv, et al, 2002B). Thus, the node which has more capacity has also more resources to share. Simulation may also include nodes without any resources to provide. This situation is obvious in the open file sharing networks where freeriding is a common aspect and it can be observed in the real statistics from the Gnutella studies.

When a resource is downloaded from a node to another, the copy of the resource is always created. Depending on the decision of the downloading node or the application, the copied item may be put on the node and thus the number of copies of the certain resource is increased. This is called replication. The replication distribution, decision whether the resources are replicated during the simulation and whether there exists correlation between the node degree distribution and resource distribution may be varied in the simulation cases. Both popularity distribution and replication distribution may evolve in the file-sharing simulations if the nodes are providing the downloaded resources to other nodes. Thus, the more popular the file is, the more it is replicated and the easier it is to find in the future.

When topology management is interest-based, an important aspect is how interest can be imitated and simulated. Normally, a user's behavior affects the distributed resources and queries and thus naturally generates interest-based queries and resources. When there is no special information, such as metadata, describing the resources and interests, the interest information needs to be derived from the resource. Thus, either the node has some specific interest, which affect to the resources set to the node (Schlosser, et al, 2002) or node has some resources which specify the interest of the group. It is also possible to use interest categories in the simulation even if those are not needed in the real world application if those are just used to imitate the interest, not to describe interest in the system.

Resources may be distributed to the nodes randomly or so that nodes have only resources belonging to their specific interests. As the studies of (Meng, et al, 2006) and (Shao, et al, 2005) have found, the most peers have interests, and thus the principle to distribute resources to nodes needs to be defined in the simulation.

The topology and search studies in the PI, PV, PVI and PVII utilize non-uniform resource distributions. In PI and PVII, the resources were distributed based on the number of the neighbors. The more neighbors the node had, the more resources it also had. PVII also used resource distribution derived from the GnutellaII. The topology studies used in PVII the capacity of the nodes as criterion of resource amounts, and PV also divided the nodes into two groups; one provided more resources than the other.

5.3 Queries

The query model defines the amount of queries sent to the network along with the distribution of resources queried and the nodes sending the queries.

The total amount of queries depends on the number of nodes and resources and study case.

Queried resource is selected from the distribution describing the probabilities of resources. Usually the resource is selected randomly in peer-to-peer simulations (Cooper, 2004), (Crespo & Garcia-Molina, 2005), (Liu, et al, 2005B). Distributions from where the resource is selected can be uniform when all resources have same probability to be selected or it may be proportional for example to the amount of specific resource in the network. Then the more popular the resource is, the more it is queried. Most used non-uniform query distribution in peer-to-peer simulators is the Zipf mentioned already in previous chapter (Tsoumakos & Roussopoulos, 2003B), (Klemm, et al, 2004), (Sripanidkulchai, 2003), (Liu, et al, 2005B).

The other distribution is needed for the selection of a querying node. The node sending a query may be selected from the uniform distribution in which case all nodes have equal probability to be selected as a source of a query. Other option is that probability of querying node is proportional for example to the resource amounts the node is providing or capacity of the node. So, the more node is providing resources to other nodes, the more it is also querying resources in the network. The distribution may also be non-uniform but not proportional to any specific feature of the nodes. The source node may be selected for example from the power-law distribution, thus there are few nodes which are making huge amount of queries and a lot of nodes which are making small amount of queries.

The random selections of a querier node and a queried resource from the uniform distributions were used in PI, PVI and PVII. The PV used non-uniform distribution for selection of querier node.

In the case of interest-based topology management, the interest of the nodes also needs to be simulated. The nodes request with in certain probability the resources which they are interested in. The modeling of the interest of the node depends on the selected method to describe the interest of the node when setting the resources. Besides the resources, also the interest may have different probabilities to be queried and, further, resources in different interest areas may have different probabilities to be selected.

In simulation, events are created in the network and their consequences are studied. There are two types of peer-to-peer simulations: time based or query based simulations. In the first case, time is involved in the simulation and events, such as sending a query, are related to the time. When the simulation is based on a query cycle, each query (sending request getting replies) is considered as an isolated event. The next query is processed only after the previous has been finished unlike in time based case where several queries may be active

simultaneously. In both cases the simulation requires iterations. In query cycle based simulations number of sent queries is amount of iterations. The decision of initiation of a query can be done in the node level, i.e. each node makes decision independently of each other, which is simpler to execute in a time based simulations, or all aspects of a query are defined and controlled in the upper level, which is well-suited in a query based simulations.

The topology management operations usually depend on the queries because the interest-based topology methods utilize the information collected from the received replies and thus some operations may be activated when a node receives replies to the query it has send. If the operations require an estimate for the loading of the node, this can be checked either after a given amount of sent queries or as a function of received replies.

Concerning the outcomes of the queries, it depends on the application and its needs as to which features are important. One may be interested in the total number of replies, the proportion of replies to the (estimated) amount of possible resources in the network, or just in the fact of finding or not finding at least one resource.

5.4 Running and Monitoring P2P Simulations

The simulation of the peer-to-peer networks may focus on the transient phase or on the equilibrium. In the first case one studies the behavior of the short time average values of relevant parameters, such as replies and topology changes, during the period of change in the network. When the goal is to achieve equilibrium, long enough simulations are needed so that it can be reliably concluded that the network is no longer changing or that the changes are just small scale fluctuations around equilibrium.

The simulator has to save or report the information needed to evaluate the studied algorithm. The needed information depends on the goal of the simulation. If the physical properties of the network are an issue, then the delays and thus the physical time is needed. This case usually requires the use of an emulator for simulation. In the case of topology management algorithms the interest is on the quantitative and qualitative properties in the overlay level and observed values during the simulation may include the final topology, sent queries and received replies, lost queries and topology changes. When studying the topology management algorithms, the used search algorithm and its parameters are also needed in the simulation.

The topology management aims to make the network more efficient in a way that the required resources can be found closer or with less traffic. As mentioned earlier, the topology changes also generate additional load in the network because of messages related to creating and managing connections and computations related to analyzing the traffic. Thus the ideal case would be that the algorithms create a network which will find equilibrium. The amount of

topology changes should be important only in the beginning and once the network has reached the equilibrium state the changes should be rare.

When a network is simulated with search algorithms with a limited scope, all resources are not necessarily found. Thus one has to define a criterion for the success of a query. The criterion of success rate may define that it is enough to find one resource or the search algorithm should find some predefined amount of all possible resources. The topology management algorithms may change the topology so that the network is partitioned and disconnected. Once again, it depends on the application, whether this partition is acceptable, if the criterion for success rate is fulfilled.

As it is can be seen, to set up a simulation of P2P network and its topology management, a great number of probability distributions and their parameters have to be defined, together with other parameters specifying the configuration. To cover systematically all combinations or modeling realistically, dependencies between different distributions is clearly too complex and quite impossible to execute. Thus the next chapter describes simulation of topology algorithms in a situation where the distributions and parameters have been chosen to keep the simulation simple.

6 SIMULATING TOPOLOGY MANAGEMENT AND DBFS

The efficiency of P2P networks, especially the efficiency of the searching, depends on the topology of the network and the used search algorithm. One search algorithm can function well in a certain type of network but give poor results in another kind of network. To avoid this situation, in addition to the topology management algorithms, adaptive search algorithms have also been developed. These utilize similar information than interest-based topology management and are easy to adapt to a system where nodes are gathering information about received replies.

In a P2P network where nodes have interests, the efficiency can be improved utilizing either an interest-based topology management with some traditional flooding search algorithm or an adaptive search algorithm. The behavior of these two approaches has not been compared systematically to conclude which gives better results for a given P2P network or to find out whether adaptive search can improve the efficiency even for a network that has been optimized with a topology management algorithm. In what follows, we study simple "static" P2P networks (that is, networks where the nodes and their resources do not change during the simulation) with topology management and adaptive search (DBFS) to find characteristics which define, when adapting the logical topology is the optimal operation, and in such cases, it is more beneficial to just concentrate on adapting the search.

6.1 Simulated Situation

The previously mentioned methods are compared in a static P2P network with predefined amount of nodes. The number of nodes is not varied during the simulation. So new nodes are not added into the network and leaving the network is also excluded in the simulation. The purpose of this is to keep the simulation simple and focused only to changes in the topology that are due to the topology management algorithms. The P2P simulation requires also resources which can be searched. To simplify the simulation and interpretation of results every node has the same amount of resources which are not replicated during the simulation. Each resource in the network has a unique identifier. The queries are designed so that the number of possible replies is constant.

The nodes are divided into eight non-overlapping interest groups based on the resources the node has. Because resources are not added or destroyed during the simulation, the defined interests will also remain unchanged. The interests are also taken into account in queries. The nodes are searching more frequently resources which belong to the same interest group as the querying node and less frequently random resources which may belong to any of the interest groups. The probabilities to the interest-based query and random query are different and constant during the whole simulation.

The resources are set randomly to the nodes so that the interest groups do not depend on the initial topology of the created network. For each query, the possible amount of resources matching the query is the same, but depending on the used time-to-live value of the search algorithm and placement of the resources, all of them are not necessarily found. The number of resources per node matching a query varies, and consequently also the amount of nodes giving replies varies from query to query.

The most important results of the simulation are the amount of received replies in relation to the sent queries and the traffic the queries are causing. These together define the efficiency of the network and these are observed in every simulation. Besides the efficiency of the network, we monitor the behavior of topology management algorithms: how fast they are changing the network and how fast the topology is stabilized. The search in the managed topology is compared to the DBFS algorithm which is simulated both in initial and managed topologies to see whether topology management will improve the performance of directed search.

6.1.1 Simulated Networks

We simulated P2P networks with 256 and 1024 nodes. The initial topologies were torus and random. The average degrees of the nodes were 4 and the number of links was the same for both topologies: $2*N$, where N is number of nodes. Random networks were created by using Erdös-Renyi model (Albert & Barabasi, 2002). Number of resources was 4096 and these were divided into 8 interest groups. One resource may belong to only one group. Queries were formed so

that 80% of queries were interest-based targeting to resources within the same group where the querying node belongs. Respectively, 20% of queries were randomly generated, but also these may have the same interest than the node. Thus, 82,5% of the queries were made to the interest group 17,5% to the rest if the network. For each query there were 32 resources fitting the specification. The setup is summarized in the Appendix.

6.1.2 Simulated Algorithms and Their Parameters

Topology management algorithms manage the overlay topology of the P2P network using the local information the nodes are gathering. Each node manages its local connections and favors the nodes which are providing resources to it. The simulated topology management algorithm from PV consists of four different algorithms. The first algorithm, called overtaking, is replacing neighbor by neighbor's neighbor, if that one is better for the node than original neighbor. The second algorithm uses two other algorithms for taking care of the node's load by adding new connections or removing existing connections. The algorithms were simulated together and the effect of traffic estimation was simulated also without the overtaking.

The goal of the overtaking algorithm is to move the node in the logical network closer to the nodes, which are providing resources to it. The overtaking algorithm calculates the received replies that its neighbors and neighbors' neighbors have sent or forwarded to it. Then it checks whether there is a neighbor's neighbor, whose proportion of all replies is more than a predefined value, for example 80%, and replaces the neighbor by that.

The traffic estimation algorithm adds or removes neighbors based on the traffic load of the node. Traffic is estimated by the amount of the query messages the nodes are receiving. There are lower and upper limits in which the nodes' traffic should remain. When the traffic of a node is between lower and upper limits, it may accept new neighbors, but it will not initiate any topology changes except the overtaking. When the traffic is higher than upper traffic limit, the node will disconnect the connection which has provided or forwarded least resource replies to it and will not accept any connection requests. If the traffic is less than lower traffic limit, the node has extra capacity and it will add a new connection. By using two limit values the algorithm avoids the situation where the nodes are constantly adding and removing their neighbors.

In the simulations, each node has an equal traffic capacity. The nodes are checking their traffic load after a given period, which is a varied parameter in the simulations. Because the simulation is not time based, the period is measured by sent queries. Each node sent in average 2, 4 or 6 queries between checking the traffic load. The maximum allowed traffic during the observation period was another simulation parameter. Values of 40% and 60% of $k*N$ were used as maximum traffic limits, where k is number of queries per node between checks and N is number of nodes in the network. The lower traffic limit value was defined as 20% of the maximum traffic limit value.

One initiated query in the network is one cycle and the frequency of the topology management is defined with help of expected number of initiated queries per node, which is a varied parameter in the simulation. The number of elapsed cycles after node's previous topology changes affects to the information available for the node. The longer the period between changes is, the more information the node has collected as a basis for deciding on topology changes.

When nodes are using BFS algorithm, they forward the query message to all neighbors except the one, from which they received the message. The only restricting parameter in BFS search is a time-to-live (TTL) value, which defines the amount of hops after which the query is no more forwarded. If the node has a queried resource, it will send a reply to the node from where the query came and if the TTL value is not zero, it forwards the query to other neighbors. When a querying node is using DBFS algorithm, the initiator of the query is forwarding a query to the limited part of the neighbors, which have replied to the node most. The algorithm has two parameters, one defining the number of neighbors to which the query is forwarded and the other defining how many hops the query is forwarded to the selected amount of neighbors after which the search is using normal BFS algorithm. The simulation used 1 or 2 neighbors with 1-2 DBFS hops. This means that the node and its neighbors sent the query to the small set of neighbors and after that the query was forwarded as the BFS query. This reduces not only the amount of nodes that will receive the query, but also the traffic in the network and because the neighbors are selected by using the amount of resource replies they have provided, it is supposedly more efficient than BFS search.

Both DBFS and BFS algorithms use TTL value to limit the horizon of the query. TTL affects to the amount of nodes the queries are reaching and thus to the amount of replies they are receiving. Further, in the interest-based search and topology management it also affects to the decisions the nodes are making because it affects to the information available to topology management. The bigger the TTL is, the further in the network potential neighbors can be found. Both algorithms are simulated with several TTL values: 3, 5 and 7.

6.2 Simulator

The interest-based topology management does not consider properties of the physical network and physical time. Thus this study uses a simulator that does not model the elapsed physical time but is based on the sent queries. The simulator uses query cycles to present the logical time. Each sent query is one cycle in the simulator.

The simulator starts by reading the configuration file and initiates the networks used in the simulations. The configuration file defines the topologies of the simulated networks and amount of nodes, resources, and queries. All parameters that the algorithms utilize and other simulation parameters are also read from the configuration file. The generated random values, such as re-

sources, queried resources and queriers, use seed values. These seed values are defined in the configuration file to enable controlled repetitions of simulation experiments. Also the topology management policy is initialized. In practice each node is given a starting time and interval for topology updates that prevents the situation that all nodes are making changes at the same time.

After creating a topology and setting the resources to the nodes, the simulation is executed one query at a time. For each query, the simulator takes one random node as an initiator of the query. The simulator also draws the queried resources from the given distribution. The simulator uses the search algorithm defined in the configuration file and sends the query into the network according to the algorithm. The nodes receiving the query will handle it and create replies, if they have resources matching the queried one. After the query has been executed possible topology updates are done according to the prescribed schedule.

The simulator saves information about networks, nodes, queries and topology changes into files. The simulator saves the topologies and neighbor distributions of the networks in the beginning and at the end of the simulations. The saved parameters about nodes are: number of neighbors, number of resources and amount of sent replies. About each forwarded query are saved the queried resource, the initiator node of the query, the used algorithm and TTL value. The replying node, distance of the replying node and the found resource are saved on each resource reply. To keep the statistics about totally failed queries, the simulator saves information about the queries, which did not receive any matching results.

If the topology is optimized in the simulation, the simulator also saves information about topology changes in the network. In each round, it saves the amount of additions, removals and overtakings which took place in the network.

6.3 Simulation Tests

The simulator's correct behavior was tested by running the BFS algorithm in the different networks used in the simulations. The simulation results of topology management algorithms and DBFS algorithm were also compared to the results attained with BFS algorithm. Because resources and queries were distributed equally, the expected values for the studied parameters were easy to calculate for comparison. Each simulation consisted of 20 trials and each trial used same parameters but different random numbers.

The simulator uses the queries as a pseudo time and the number of resource queries defines the duration of the simulation. The duration chosen was long enough to reach the state where the topology does not change anymore. This depends on the size of the network, the TTL value, and used policy. When the topology is modified based on the sent queries and received replies it is essential that the simulation time is long enough so that the network has a possi-

bility to attain a balance. In addition to the properties of the network at equilibrium, the rate of the changes in the topology is studied too.

FIGURE 3 presents the average amount of replies in proportion to the average amount of queries per hop until TTL seven using the BFS algorithm in the torus and random networks of 256 nodes. The corresponding results of BFS algorithm in the torus and random network of 1024 nodes are presented in the FIGURE 4. The efficiency values of the simulations in different networks are illustrated in the FIGURE 5. The efficiency value is calculated by R/Q, where R is amount of received replies and Q is amount of generated query messages. The averages of queries, replies and efficiency values are calculated over the all simulation cases and presented per an initiated query. The numerical data and the ranges are presented in the appendix. Success rates of different BFS simulations are presented in FIGURE 6. The query is considered successful if at least one reply is received. Thus the success rate charts illustrate the proportion of successful queries per all queries.

When results of different networks of 256 nodes are compared, the random network with TTL 4 achieves more replies than the torus network with TTL 7. The total average of found resources was over 17 replies more than in the torus network.

In the network of 256 nodes, each node has 16 resources. Corresponding resource amount per node in the bigger networks is 4. Thus in the bigger network, each node has 25% of the resources the nodes in the smaller network have. This can be observed also directly from average reply amounts in the different torus networks. The amounts of replies in larger network are slightly better because the querier node has also smaller amount of resources in the smaller network than the bigger one.

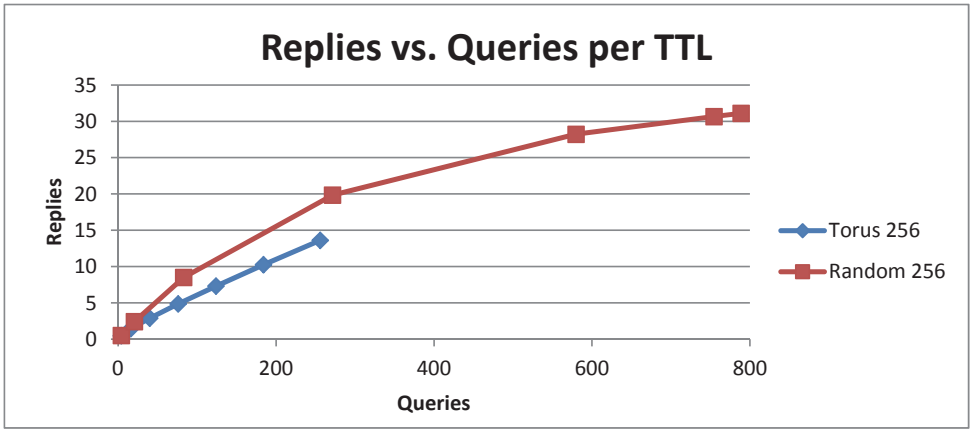

FIGURE 3 The average amount of replies in proportion to the amount of query messages in networks of 256 nodes with BFS algorithm.

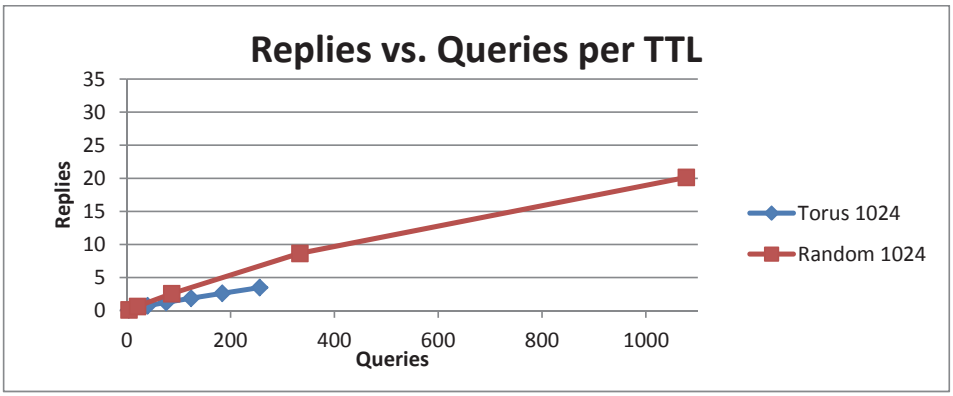

FIGURE 4 The average amount of replies in proportion to the amount of query messages in networks of 1024 nodes with BFS algorithm.

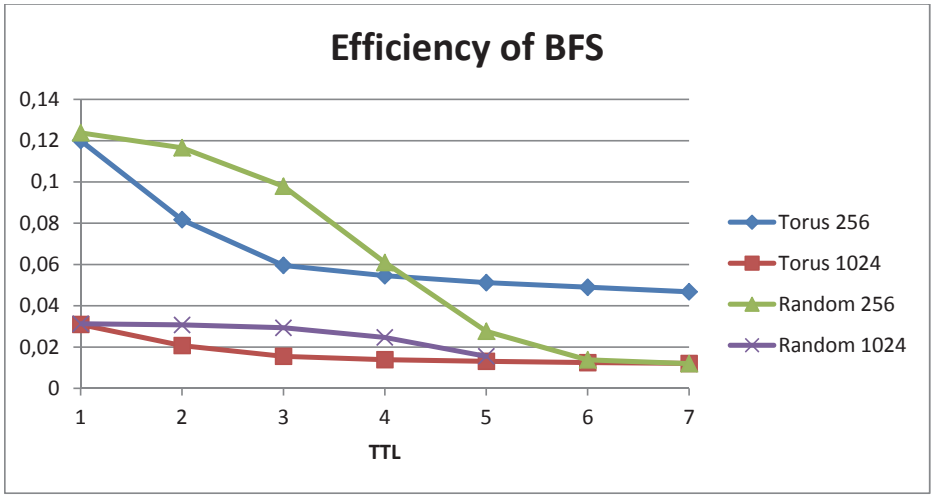

FIGURE 5 Efficiency of BFS algorithm per TTL values in different networks.

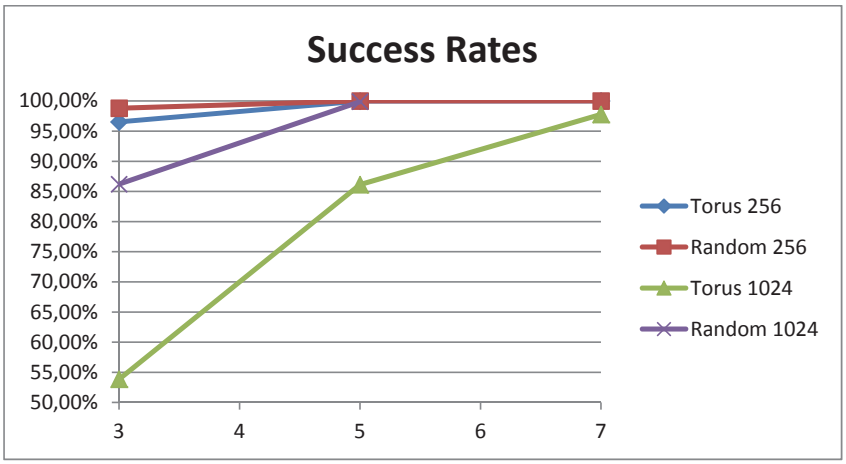

FIGURE 6 Success rates of BFS algorithm per TTL values in different networks.

Based on the success rates, it can be derived that even TTL 3 is large enough for torus and random networks of 256 nodes. TTL 7 is not giving any more extra value compared to the huge amount of queries it is causing in hops six and seven. The average amount of queries without any replies was in the torus network of 256 nodes with TTL 3, 3,5% while in random networks, the value was 1,2%. This can be explained with the shorter diameter of the random network compared to the torus network. With TTL value 3 the query will reach more nodes in random network than in the torus network.

Because of the long diameter of the torus network of 1024 nodes, the success rate with TTL 7 is close to the success rate of the same size random network with TTL 5. TTL 3 is too small for torus network of 1024 nodes as almost half of queries remains without any reply.

6.4 Simulation Cases

Simulation experiments studied the behavior of the DBFS algorithm, the topology management algorithms and the combination of the topology management and DBFS algorithms. The DBFS algorithm was simulated in static networks with different TTL values. The topology management algorithms were studied in the same networks with the same TTL values. The behavior of traffic estimation and overtaking was studied with different topology change frequencies. The obtained results were compared to the results of DBFS algorithm to find out the parameters which define, when the DBFS with static network achieved better results than topology management and when managing the topology and use of BFS algorithm is more efficient. Both cases were compared also to the static network with BFS algorithm. Finally, the DBFS algorithm was simulated

also in the networks, which resulted from using the topology management algorithm.

6.4.1 DBFS

DBFS algorithm has two main parameters: the amount of selected neighbors and the amount of directed hops the algorithm uses. To guarantee the comparability, the simulations used similar initiation rounds as topology management algorithms. In initialization rounds, the nodes searched with BFS algorithm and saved information about received replies. DBFS was studied with 2, 4 and 6 initiation rounds, for 1 and 2 selected neighbors with 1 and 2 hops. The networks were torus and random network of 256 and 1024 nodes. As assumed, DBFS gave better success rates with all used parameters than BFS. Because the simulated networks and amount of neighbors were quite small, the results of 1 selected neighbor in 1 hop are presented. The results of DBFS algorithm in networks of 256 nodes are presented in FIGURE 7 and FIGURE 8. The corresponding results of larger networks are in FIGURE 9 and FIGURE 10 and success rates in different networks in FIGURE 11. The cumulative values of replies and queries are presented only for TTL 7, as behavior is similar for smaller values of TTL.

As the figures illustrating the efficiency values show, the results are better in the first hops the smaller the TTL value is. Neither the equal distribution of resources nor the torus network are suitable for the DBFS, especially with large TTL value, because in the static network all directions are equally good for the node.

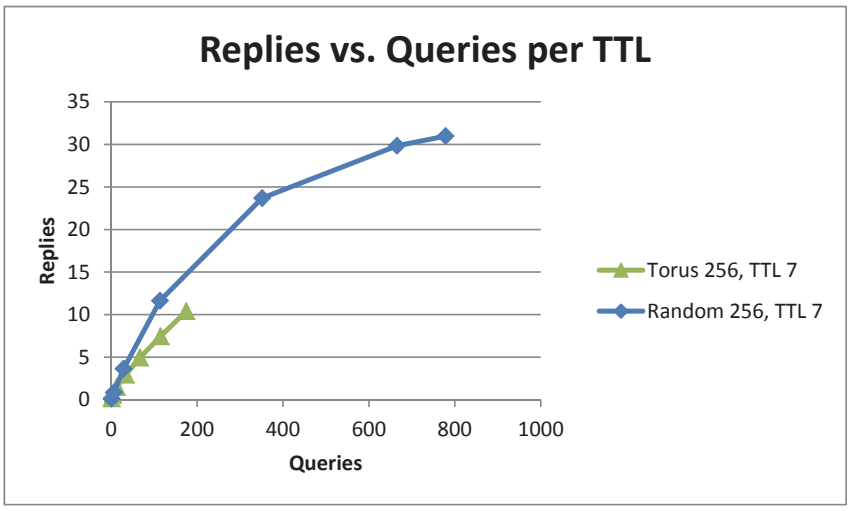

FIGURE 7 The average amount of replies in proportion to the amount of query messages in networks of 256 nodes with DBFS algorithm.

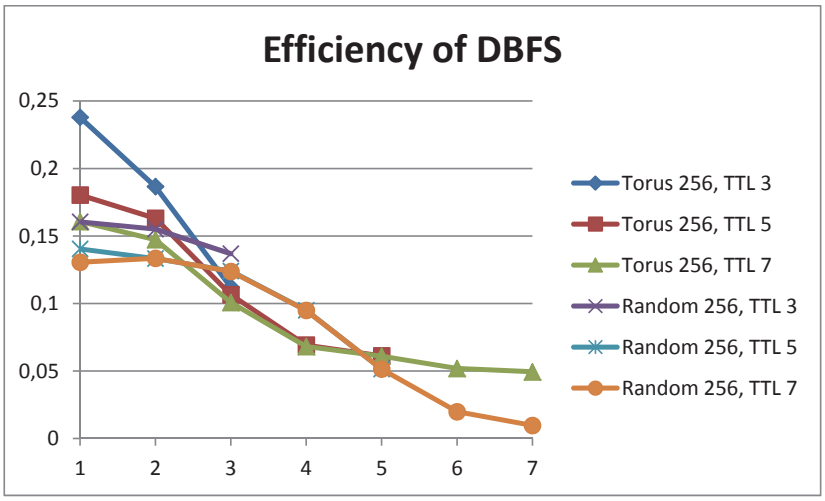

FIGURE 8 Efficiency of DBFS algorithm per TTL values in different networks of 256 nodes.

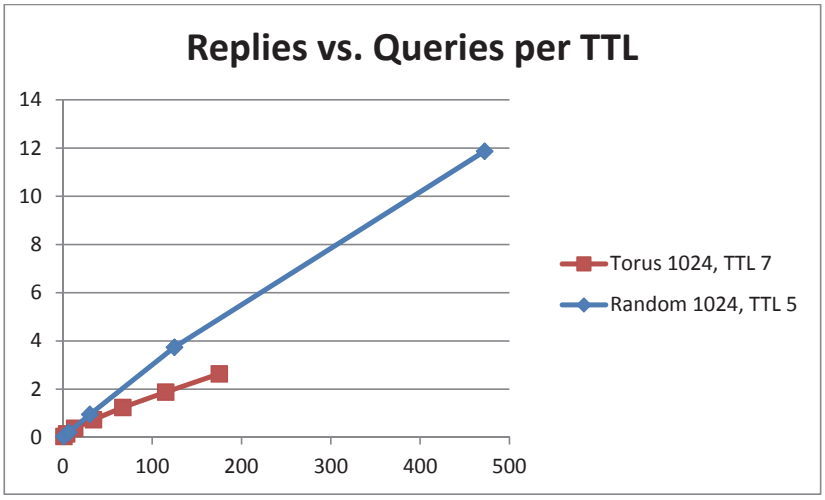

FIGURE 9 The average amount of replies in proportion to the amount of query messages in networks of 1024 nodes with DBFS algorithm.

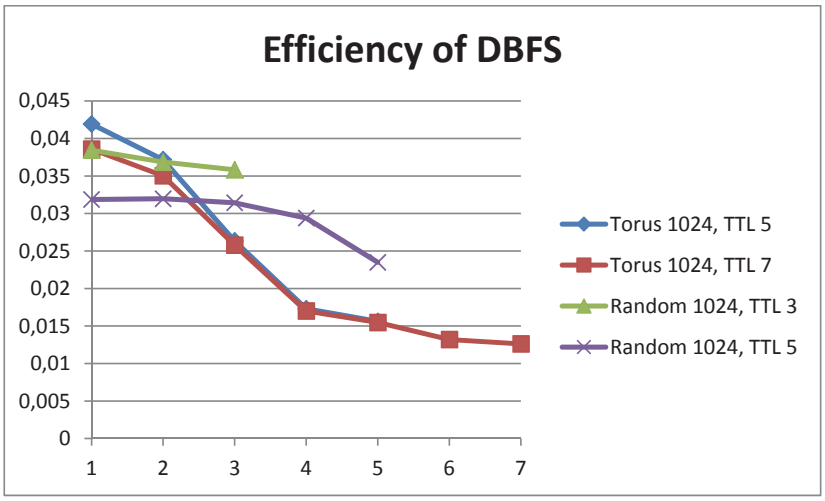

FIGURE 10 Efficiency of DBFS algorithm per TTL values in different networks of 1024 nodes.

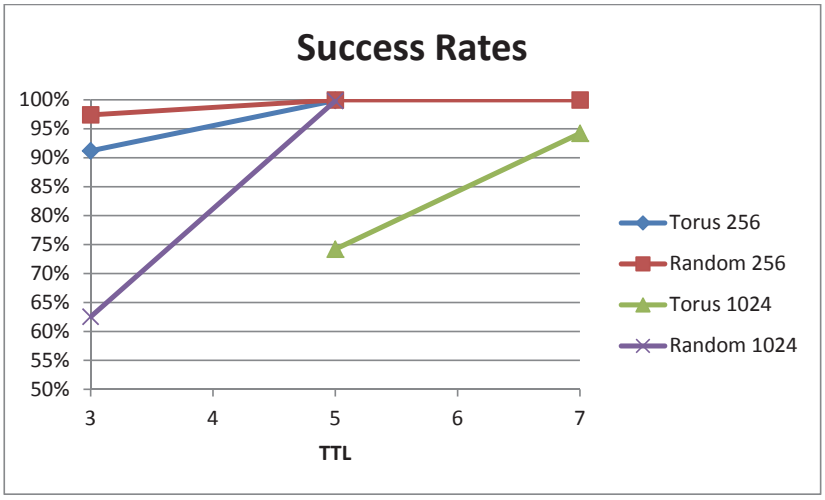

FIGURE 11 Average of success rates in different networks with DBFS using different TTL values.

6.4.2 Topology Management Algorithms

To study the impact of topology management algorithms, traffic estimation and overtaking, the simulations were performed both without overtaking and with different overtaking parameters. Without overtaking, the topology changes are based on the traffic measurements and the decisions of the traffic estimation algorithm only. The simulations used upper traffic limit (*UTL*) values 40% and 60% and interval of the traffic checkings (*k*) was 2, 4 or 6. This means that the maximum traffic of a node was *k*N*UTL*, where *N* is amount of nodes.

When the TTL is three, the horizon is small and traffic estimation with used parameters does nothing. The topology remains unchanged during the simulations and the results are the same as presented in the section 6.3. The results of the TTL values five and seven are presented in the TABLE 2. The topology changes and success rates are presented as average values over 20 replications. Topologies do change but most of the changes are neighbor removals and the frequencies of traffic estimation and upper traffic limit values have influence on the changes. The amount of removals decreases as the interval lengthens or the upper traffic limit increases. When TTL is five, the amount of removals is significantly larger with upper traffic limit 40% than 60%.

TABLE 2 Amount of topology changes and success rates in torus network of 256 nodes without overtaking.

TTL	Interval of Traffic Checkings	Upper Traffic Limit	Amount of Additions	Amount of Removals	Success Rate
5	2	40	0,1	162,45	98,10%
		60	0	33,5	99,98%
	4	40	0	118,8	99,73%
		60	0	0,95	99,98%
	6	40	0	111,15	99,84%
		60	0	0,05	99,98%
7	2	40	2,25	258,75	78,44%
		60	0,95	225,9	92,35%
	4	40	3,85	232,15	92,34%
		60	0,55	180,4	99,43%
	6	40	2,4	290,85	98,13%
		60	0,4	160,85	99,82%

The efficiency values achieved with TTL five, with upper traffic limit 60% are almost equal to the efficiency of BFS algorithm in the static environment. By using upper traffic limit 40% the efficiency values are slightly better in every hop. With TTL 7 the efficiency values are significantly better if compared to BFS in the static network, especially when the upper traffic limit was 40%. The improvement of efficiency values is a result of the decrease of redundant connections.

If only the efficiency values would be studied, the received results with TTL 7 would be good. Even if there are a lot of changes in the network, most changes happen in the early cycles in the simulation. However, the percentages of the lost queries are indicating that the topology is not ideal. Even if the percentages are remaining reasonable, they are measuring the queries that are not receiving any replies. Even in a disconnected network, it is rare that a node does not have path to any node that has the queried resource.

When the topologies are studied, it is observed that, especially when using the TTL 7 with small intervals and upper traffic limit 40%, the networks are

fragmented. The most connected network is achieved with parameters interval 6 and upper traffic limit 60%. The results of this case and best case of TTL 5 with frequency 6 and upper traffic limit 40% are presented in FIGURE 12 and FIGURE 13. As FIGURE 12 shows, with TTL 7 the average amount of received replies is significantly smaller than with BFS in static network. This is explained by the neighbor removals the topology management generated. Because amount of connections is decreased and thus the distances are not any more equal, the query is reaching fewer nodes than in static torus network with the same TTL. However, it finds the same amount of replies with fewer query amounts than BFS. Thus the (unwanted) fragmentation of the network seems to happen mainly in between the different interest groups. The 20% of random queries is not enough to maintain overall connectivity.

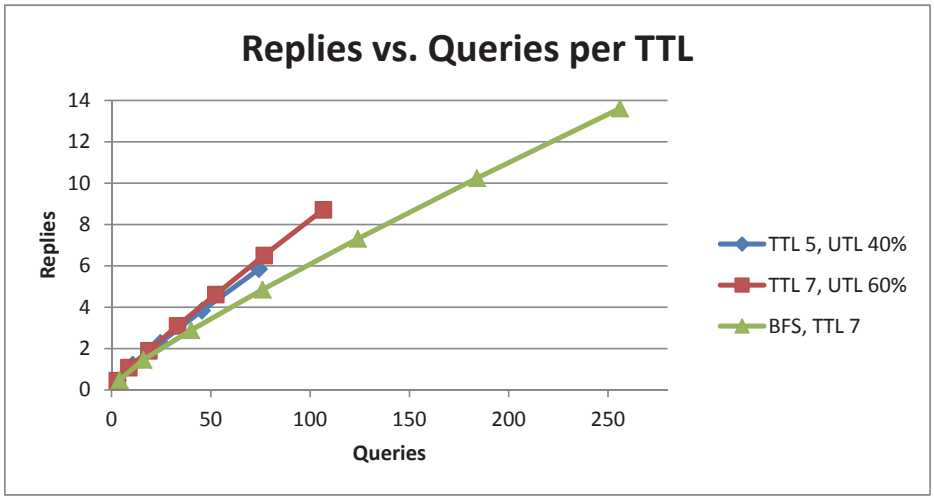

FIGURE 12 The average amount of replies in proportion to the amount of query messages in torus network of 256 nodes without overtaking.

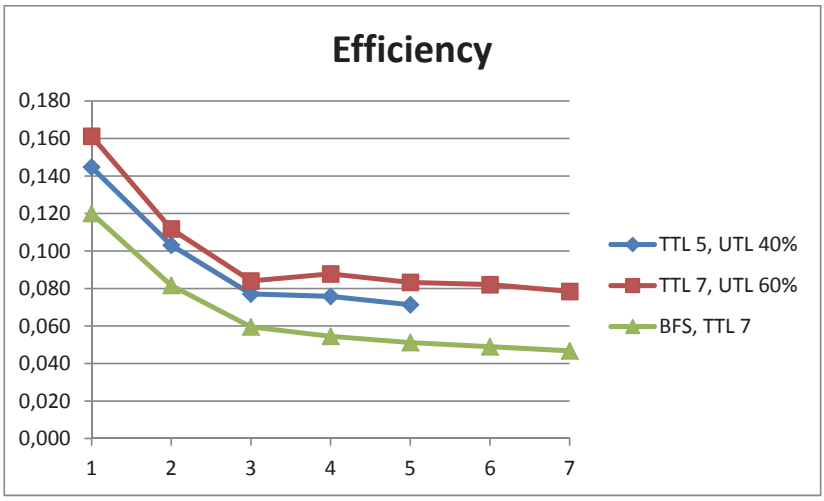

FIGURE 13 Efficiency per TTL values in torus network of 256 nodes without overtaking.

With overtaking the networks were fragmented. The larger the TTL, the more disconnected the network. FIGURE 14 presents a sample result topology with TTL 7 with overtaking applied. The network is decomposed to several components: one large cluster with few nodes with high degree and several small clusters. The effect of the frequency or upper traffic limit was not as clear as in the tests without overtaking.

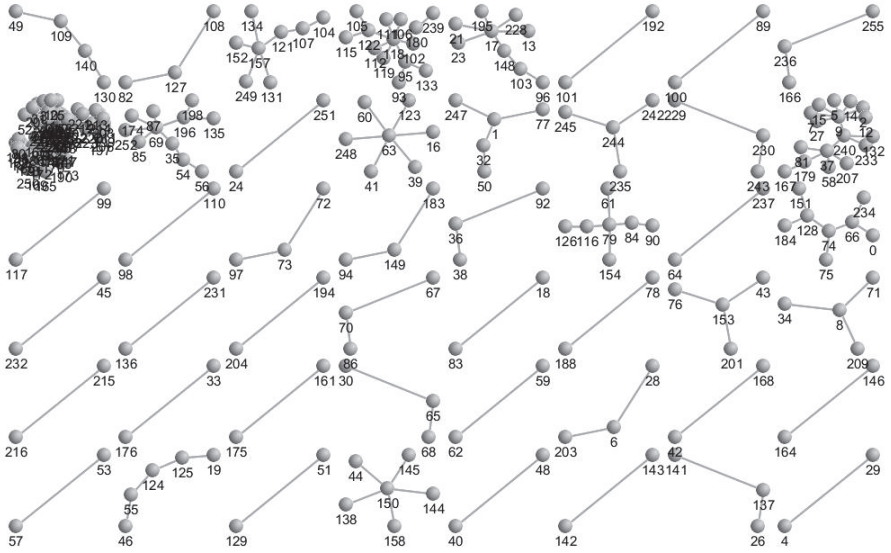

FIGURE 14 Topology of the network in the test case, where TTL was 7, overtaking percent 80, upper traffic limit 60% and frequency 6.

With the TTL value 3, the half of topologies stayed connected and fragmented networks had only few nodes separated from the others. One generated topology is presented in FIGURE 15.

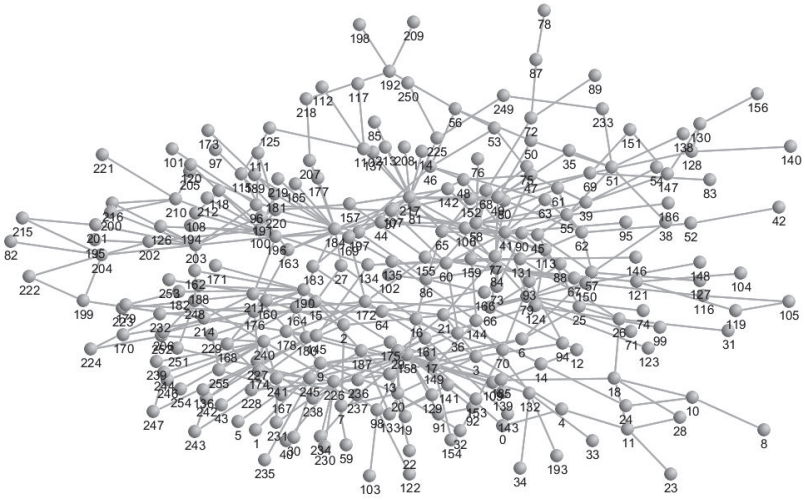

FIGURE 15 Topology of the network in the test case, where TTL was 3, overtaking percent 80, upper traffic limit 60% and frequency 6.

As FIGURE 16, FIGURE 17 and TABLE 3 show, the results are clearly better compared to the BFS in static network, but the percentage of the lost messages is significantly high with larger TTL values. So, opposite to the BFS in static network, the success rates are decreasing as the TTL value is increased because of the fragmentation of the network.

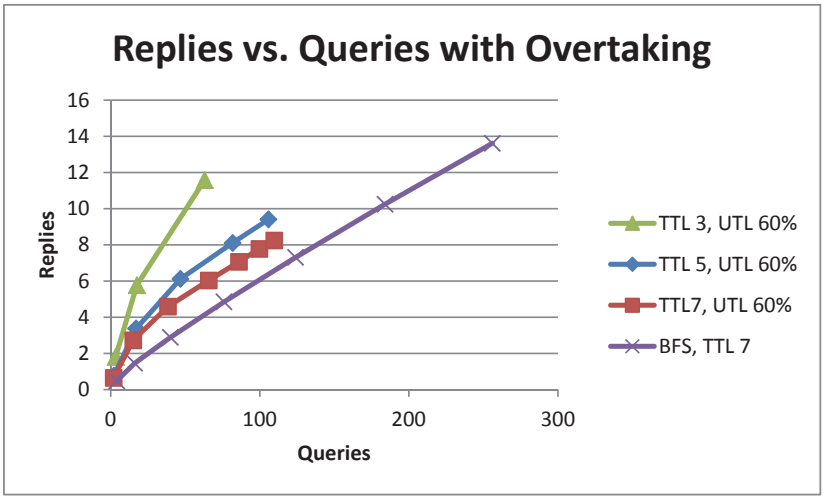

FIGURE 16 The average amount of replies in proportion to the amount of query messages in torus network of 256 nodes with overtaking percent 80.

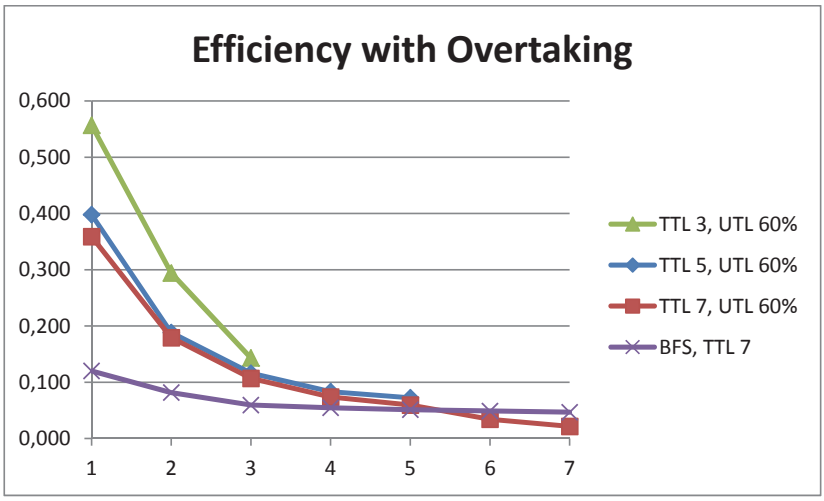

FIGURE 17 Efficiency per TTL values in torus network of 256 nodes with overtaking percent 80.

TABLE 3 Success rates without and with overtaking.

	Without Overtaking	**With Overtaking**
TTL 3, UTL 60%	96,52%	95,57%
TTL 5, UTL 40%	99,84%	84,95% (UTL60)
TTL 7, UTL 60%	99,82%	75,87%

The improvement in the efficiency is due to the star-like topologies of the connected parts of the topology. Thus if only the ratio of replies to queries would have been studied, it could have been concluded that topology algorithms give better results than just using BFS in static network. The networks achieved equilibrium quite fast and the efficiency values were improved compared to the initial networks. However, the networks were mostly disconnected and results cannot be considered satisfactory.

The decomposition of the networks to possibly star-like components was evaluated to be a consequence of the traffic estimation being initiated too infrequently and the overtaking algorithm initiated too often. The traffic estimation should work in such a way that it prevents the nodes from overloading, but if several nodes have a chance to make changes to the connections between the traffic checks, a node may accept several new connections before checking its traffic. Once the node checks the traffic, it is already overloaded. As the neighbors are then dropped one by one, it requires several cycles to disconnect extra nodes. Moreover, if the neighborhood of an overloaded node contains another high degree node, the traffic in other neighboring nodes is also high, and they do not accept new neighbors easily. Thus the nodes that are dropped are not accepted as neighbors by any node in the network, and the network will be disconnected. Finally, a node with several neighbors is also the direction from where other nodes are receiving replies and thus they want to overtake in that

direction, which leads to the star-like topologies. In small torus networks, most of the changes are done in a first possible cycles, so the traffic estimation algorithm should check the possible overload situation more actively.

A node should make an overtaking decision based on the received replies, but if the situation is checked too often, the available information is incomplete. If a node has a new neighbor, it does not have a chance to compete with the older neighbors. Even if the overtaking algorithm had an initialization phase when nodes were just collecting information and did not make any overtaking decisions, the latter checkings were done after each query. After the initialization phase, each node checked the situation after each sent query and received replies. This behavior might cause the situation where nodes were overtaken too fast and based on information of one query.

The conclusion of the previous analysis was to make the traffic estimation algorithm more active and the overtaking algorithm less active. The aim of this was to decrease the amount of topology changes so that the information used in these changes is more complete. At first, the traffic estimation algorithm was changed so that nodes check the possible overload situation more often. The original traffic checking rate was $k*N$ cycles and it was kept the same but one additional check for overloading was added after each 16 cycles. If the node is overloaded, it does not accept any new connections to the node until the situation is changed and traffic is under the upper limit value.

Another improvement was made in the overtaking algorithm. An additional parameter controlling the rate of the overtakings was included in the simulator. The parameter defines how many queries the node should wait before checking the overtaking situation. This same parameter defines also the length of the previously used initialization phase. The new algorithms were tested with the overtaking periods 10, 15 and 20. In addition, different values were tested for the overtaking threshold (80% and 90% of replies).

The results of the tests were evaluated based on the topology of the constructed network, and values of calculated functions are presented in TABLE 4. For each test configuration, all 20 samples of constructed topologies were checked to find out whether they are connected or disconnected. Also, the structures of the topologies were studied and average values of leaf nodes and maximum degrees of the nodes were calculated. Other studied parameters were the efficiency values and success rates.

The simulation tests with traffic limit value 60% gave best results when overtaking percent was 90. With 80% overtaking, the average efficiency values were better, but the networks were less stable and more centralized and amount of lost queries was higher. The best combinations of results were received with 90% overtaking threshold, i.e., only with 90%; all the 20 cases formed topologies which were connected and the amount of leaf nodes was realistic (approximately between 20 and 40), and the largest node or nodes have 9-13 neighbors. The more detailed statistics about selected test cases are presented in TABLE 4.

TABLE 4 Selected test cases with traffic limit 60% and with TTL3.

OT Percent	OT Period	Checking Interval	Disconnected Networks	Leaf Nodes	Max Degree	Efficiency			Success Rate
						TTL 1	TTL 2	TTL 3	
80	10	6	2	50-84	11-38	0,396	0,174	0,090	97,02%
80	15	6	0	45-67	11-28	0,353	0,149	0,081	97,54%
80	20	6	0	40-67	11-26	0,330	0,139	0,077	97,73%
90	10	2	2	34-56	9-22	0,314	0,132	0,077	97,95%
90	10	4	1	28-55	10-21	0,294	0,125	0,074	97,85%
90	10	6	0	25-42	9-13	0,265	0,116	0,070	98,20%
90	15	2	0	30-45	9-13	0,277	0,117	0,071	98,18%
90	15	4	1	26-37	9-12	0,262	0,113	0,069	98,15%
90	15	6	1	16-32	9-12	0,240	0,107	0,067	98,34%
90	20	2	0	23-45	10-15	0,255	0,112	0,068	98,26%
90	20	4	0	22-37	10-14	0,239	0,105	0,066	98,25%
90	20	6	0	15-28	9-12	0,228	0,103	0,065	98,18%
Results of static network with BFS						0,120	0,082	0,060	96,52%

It can be observed that overtaking percent has the biggest influence on the amount of leaf nodes and the degree of the central node (i.e., the node with the highest degree). With 80 percent, the amount of leaf nodes is higher than with 90% for all used traffic checking or overtaking frequencies. With the traffic limit 60%, the success rates were also better with overtaking percent 90. The traffic checking frequency does not seem to have impact on the degree size of the central node, but shorter OT period seems to add the amount of leaf nodes.

The test case using overtaking percent 90, OT period 20 and traffic checking period 6 generated the network with good success rate, significantly better efficiency compared to the BFS in static network and the topology, where size of the central node and amount of leaf nodes were reasonable taken into account the small network size, 256 nodes. When compared to the total amount of average reply amounts, there was no significant difference with BFS in static network, but the amount of queries decreased.

The same parameter values were tested also with larger TTL values in torus network and also in random network with and without overtaking. The results are presented in the FIGURE 18, FIGURE 19, TABLE 5 and TABLE 6. With TTL 5 in torus network two cases were disconnected, but in both cases only couple of nodes was outside. With TTL 7 almost all, 19 networks were disconnected. The disconnecting of network can be observed for TTL 7 already without overtaking but in significantly lesser amount: three cases were disconnected in a way that in each one cluster of two nodes was separated. With smaller TTL values the networks remained unchanged without overtaking. With TTL 7, using overtaking, the average amount of queries was half of the queries in static network, but reply amount was just a few less than in static. The efficiency of the search is a consequence of the partition, which occurred in such way that nodes are still connected to the nodes that have resources with similar interests. The same reason is behind the huge amount of overtakings with TTL 7.

In the random networks of 256 nodes, the horizon of TTL 3 is quite similar to the TTL 5 in torus and thus the TTL 5 and 7 are not presented. (With larger TTL values networks are disconnected with or without the overtaking.)

The efficiency values are presented in FIGURE 19, where also the efficiency values of static torus and random networks are shown for comparison. In all cases, the efficiency values were improved especially in the first two hops of the query.

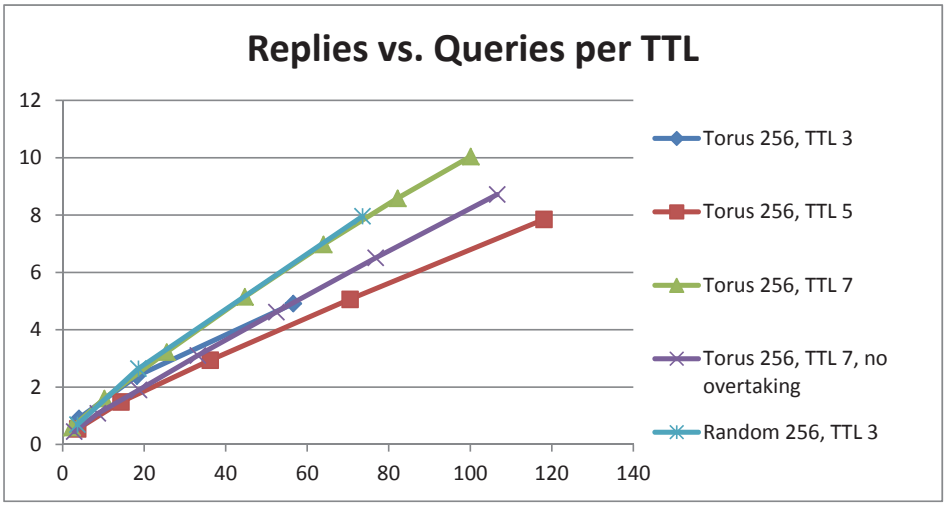

FIGURE 18 The average amount of replies in proportion to the amount of query messages in networks of 256 nodes with upper traffic limit 60, interval of traffic checkings 6, overtaking period 20 and overtaking percent 90.

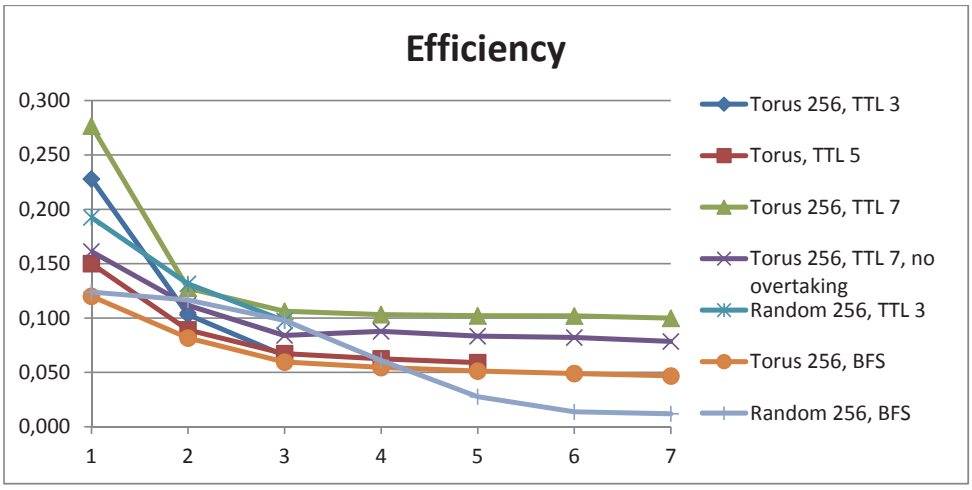

FIGURE 19 Efficiency per TTL values in networks of 256 nodes with upper traffic limit 60, interval of traffic checkings 6, overtaking period 20 and overtaking percent 90. For the comparison, efficiency of BFS in static network is also shown.

TABLE 5 Success rates without and with overtaking in torus and random networks of 256 nodes.

Network	TTL	Without Overtaking	With Overtaking
Torus	3	96,52%	98,18%
	5	99,98%	99,64%
	7	99,82%	97,24%
Random	3	98,67%	97,31%

TABLE 6 Amount of changes without and with overtaking in torus and random networks of 256 nodes.

Network	TTL	Without Overtaking		With Overtaking		
		Additions	Removals	Additions	Removals	Overtakings
Torus	3	0	0	0,15	3,45	260,15
	5	0	0,05	0,8	90,9	118,6
	7	0,4	160,85	12,95	262,35	409,6
Random	3	0,15	21,25	4,8	85,55	185,85

Simulation results of larger networks are presented in FIGURE 20, FIGURE 21 and TABLE 7. Only results with the overtaking are shown because without overtaking the networks did not change. The networks were well-connected; with TTL 5 in torus network only one case was disconnected and 2 nodes were in separate network. Also in the random network with TTL 3 one case was disconnected leaving network of 3 nodes outside of the connected part, but with TTL 5 almost all networks were partitioned in a way that couple of nodes were left outside of the larger cluster.

The efficiency values are improved especially in first hops. The efficiency of TTL 7 in torus network stays similar to the efficiency of static network because the topology does not change much. Also with TTL5, the amount of changes is so small that the improvement of efficiency is small. In the torus network, the smaller the TTL, the larger is the number of overtakings. This is as expected as the resources are distributed randomly and all directions should be equally good on the average. With small TTL, the overtaking mechanism can adapt to local fluctuations that smooth out when the search neighborhood is larger.

A clear improvement can be seen in success rates of the torus and random networks with TTL 3 and torus with TTL 5.

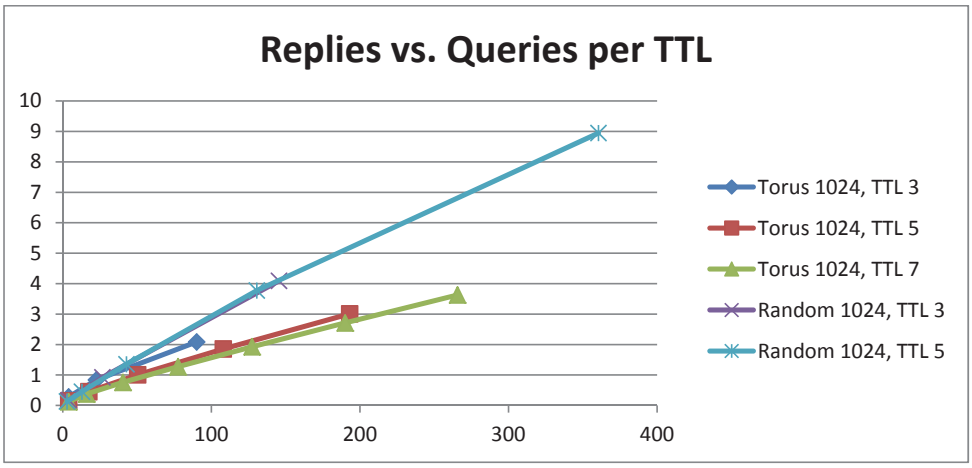

FIGURE 20 The average amount of replies in proportion to the amount of query messages in networks of 1024 nodes with upper traffic limit 60, interval of traffic checkings 6, overtaking period 20 and overtaking percent 90.

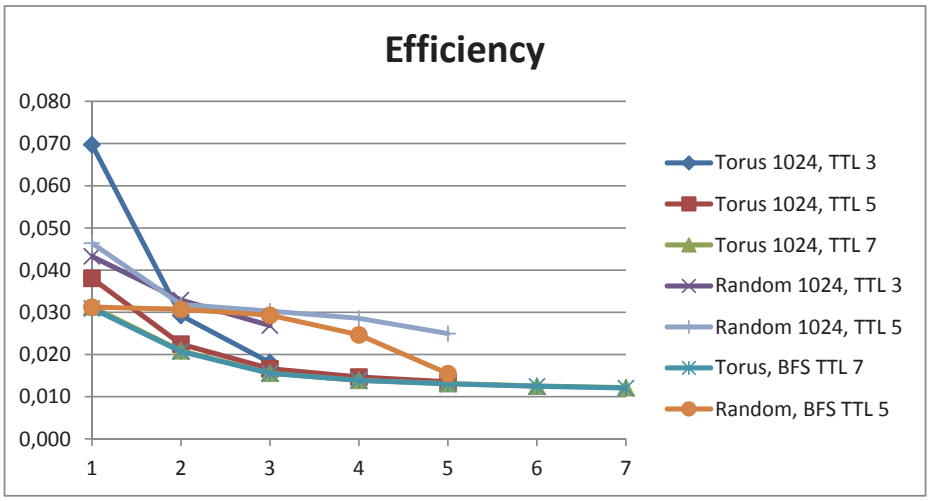

FIGURE 21 Efficiency per TTL values in networks of 1024 nodes with upper traffic limit 60, interval of traffic checkings 6, overtaking period 20 and overtaking percent 90. For the comparison, efficiency of BFS in static networks is also shown.

TABLE 7 Success rates and amount of changes overtaking in torus and random networks of 1024 nodes.

Network	TTL	Success Rate	Changes		
			Additions	Removals	Overtakings
Torus	3	80,41%	0,35	6,05	2419,85
	5	94,52%	0,5	8,95	628,60
	7	98,15%	0	0	28,25
Random	3	88,69%	2,55	18,55	999,10
	5	98,16%	50,50	743,20	654,05

In addition to the total amount of topology changes, the time evolution of changes during the simulation was studied. FIGURE 22 shows changes in a random network of 256 nodes (the corresponding graphs of other networks are included in the APPENDIX). As shown in FIGURE 22, the topology changes occur at the early cycles after, which the topology changes are rare.

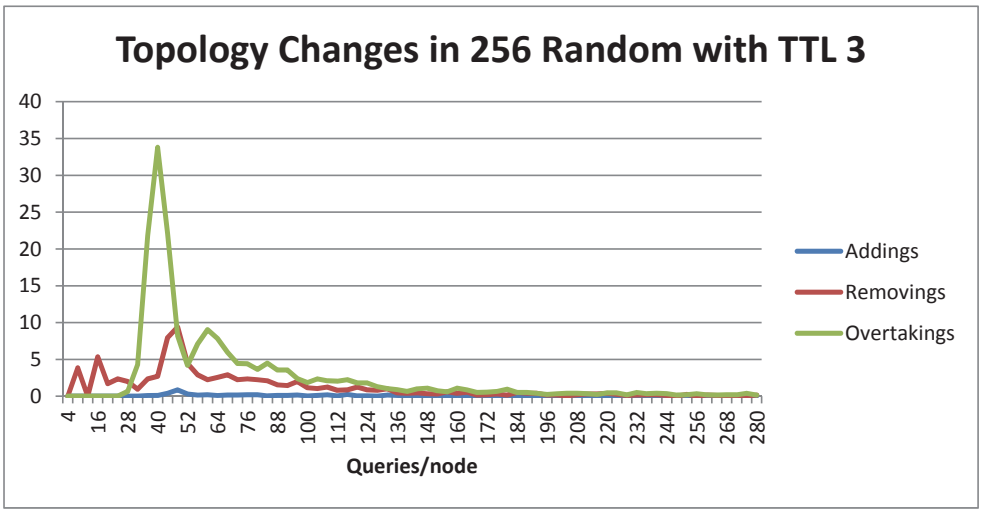

FIGURE 22 The amount of topology changes during the simulation of topology management algorithms in random network of 256 nodes with TTL 3.

The results presented in this chapter include also the initiation phase and the cycles when the network is changing a lot. To study the resulted topology (i.e., the outcome of the algorithms), the values at the equilibrium were also studied and results are presented in FIGURE 23, FIGURE 24, and TABLE 8. As the FIGURE 23 shows, the first two hops give quite similar results in the both smaller networks but in the third hop the random is achieving better results compared to the torus. Thus the initial topology can still be seen in the performance, but not until the third hop. If the results are compared to the results of BFS algorithm presented in the FIGURE 3, the improvement in torus network can be clearly seen: the amount of replies is increased and amount of queries is decreased. Thus the topology management algorithm has made the two different

initial topologies to perform qualitatively similarly, which was the intended purpose.

The notable improvement is in success rates of torus and random networks of 1024 nodes, the success rates are close to 100% whereas in a static network, these rates were 54% and 86%.

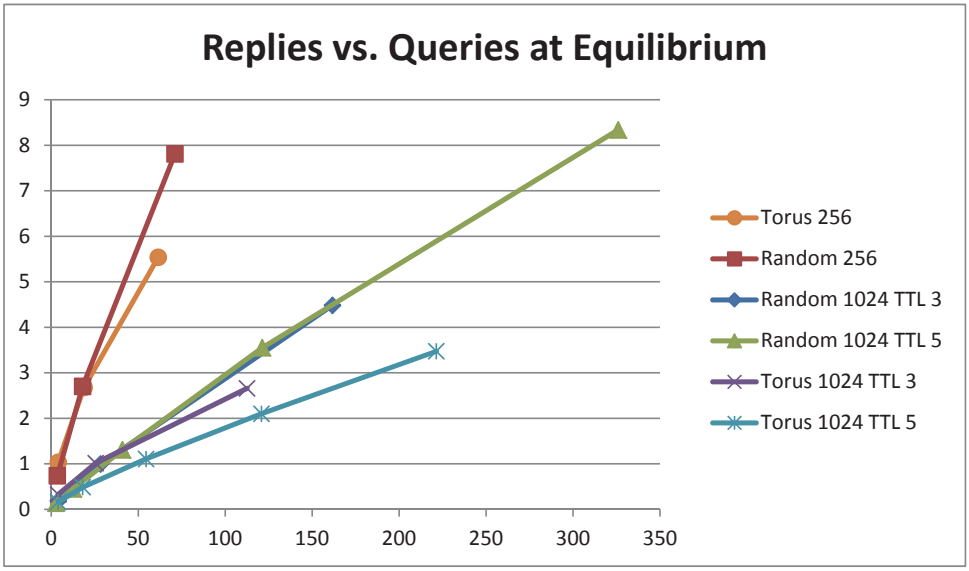

FIGURE 23 The average amount of replies in proportion to the amount of query messages in different networks after equilibrium.

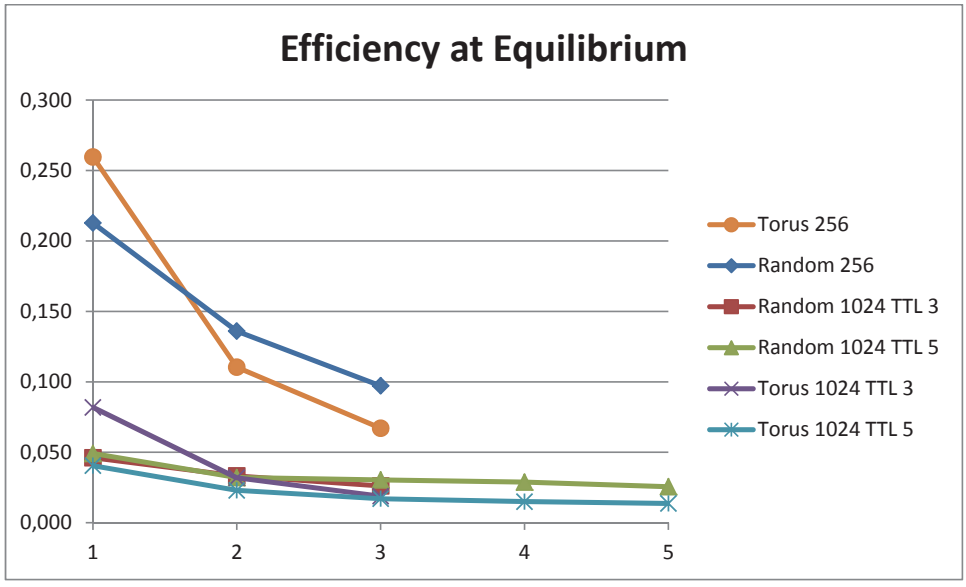

FIGURE 24 Efficiency per TTL values with overtaking after equilibrium.

TABLE 8 Success rates of networks at equilibrium.

Network	Size	TTL	Success Rate	Success Rate of BFS
Torus	256	3	99,64%	96,52%
Random	256	3	99,28%	98,80%
Torus	1024	3	96,67%	53,85%
		5	99,12%	86,12%
Random	1024	3	97,26%	86,20%
		5	99,53%	99,88%

6.4.3 Comparison of Topology Management Algorithms and DBFS Algorithm

The topology management algorithms with BFS search were compared to the DBFS in static networks. Moreover the DBFS algorithm was studied in the networks reconstructed by the topology management. The results are presented in FIGURE 25, FIGURE 26 and TABLE 9. As DBFS and BFS produce significantly different amounts of forwarded queries for a given TTL level, efficiency values are compared as a function of the amount of query messages.

The main observation is that the topology management clearly changes the immediate neighborhood of the nodes. The efficiency values for TTL 1 in DBFS show that the first neighbor in the selected preferred direction is in the same interest group in half of the cases (giving close to 4 times more replies than average node, especially for topologies derived from the torus - for modified random grids the effect is smaller). Also, the TTL 2 level for DBFS is clearly above average efficiency. On TTL 3, the effect disappears, but the performance is still better than in the original topology.

The above observation does not surprise. As the original topologies and resource distributions did not have any clustering of interests, there was not much to be achieved by directed search. The moderate clustering that emerged from the use of topology management improves the performance of the directed search but only locally.

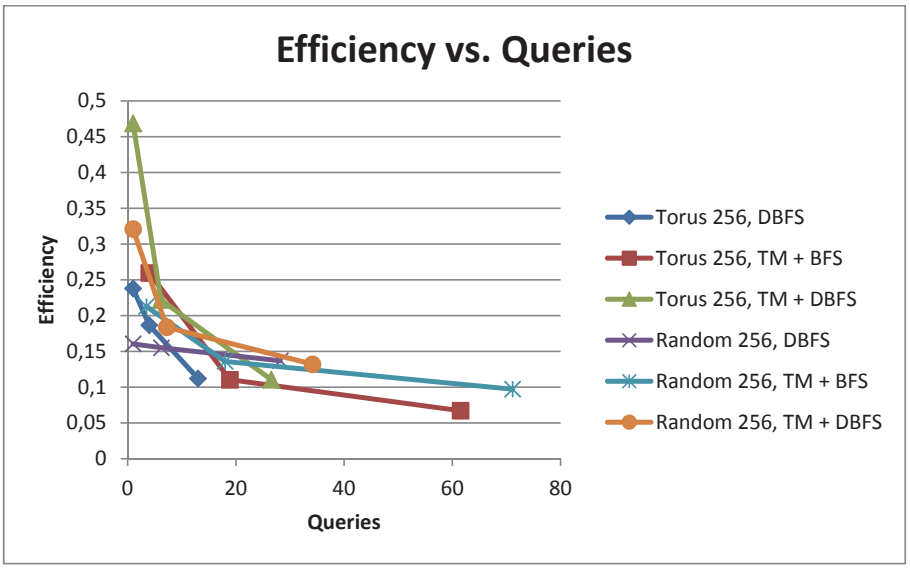

FIGURE 25 Efficiency vs. query amount of DBFS algorithm in static networks and BFS and DBFS algorithms in the networks generated by topology management algorithms.

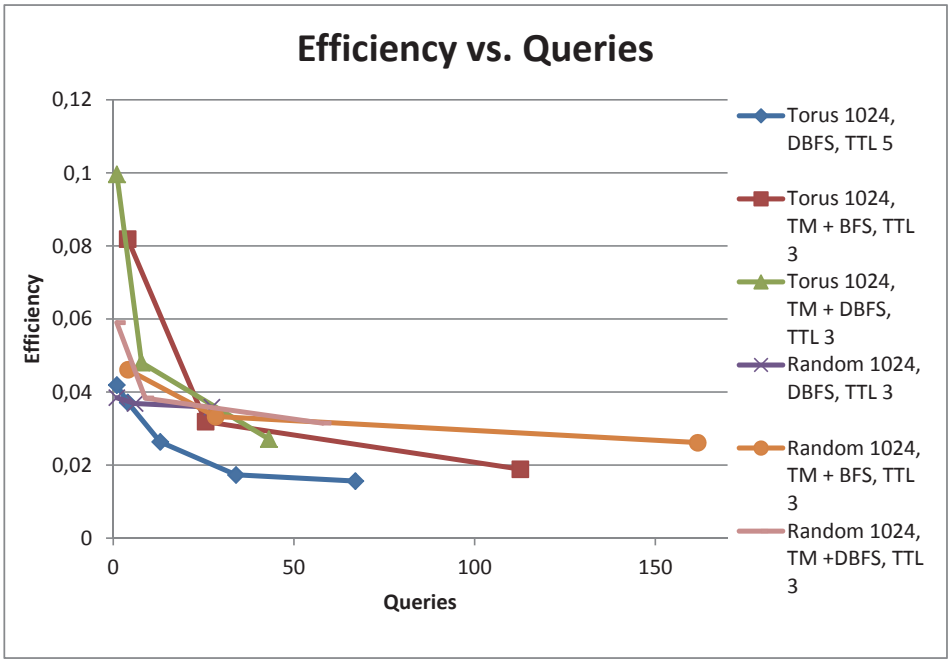

FIGURE 26 Efficiency vs. query amounts of DBFS algorithm in static networks and BFS and DBFS algorithms in the networks generated by topology management algorithms.

TABLE 9 The success rates of DBFS algorithm in static networks and BFS and DBFS algorithms in the networks generated by topology management (TM) algorithms.

Network	Size	TTL	Success Rate of TM & DBFS	Success Rate of TM & BFS	Success Rate of DBFS in Static Network
Torus	256	3	96,66%	99,64%	91,19%
Random	256	3	97,58%	99,28%	97,41%
Torus	1024	3	83,38%	96,67%	
		5	96,09%	99,12%	74,20%
Random	1024	3	88,25%	97,26%	62,53%
		5	96,53%	99,53%	99,82%

6.5 Conclusions

The simulation of peer-to-peer networks and algorithms managing the network is complex and based on several parameters, which should be defined. The essential part of simulation is to keep it simple so it can be controlled and the parameters affecting the results can be found. Before understanding thoroughly the performance of the algorithms in a simple setup, it is not advisable to try them on real, measurement-based data whose properties are only partially understood and controllable.

The original purpose of the simulation was to study the topologies that emerge from the use of the developed topology management algorithms and compare the BFS search in managed networks with DBFS in static networks to find out when the optimization of topology is delivering better results than DBFS. The generated topologies, amount of queries and replies, efficiency values and success rates were studied. The simulations used two different topologies, namely, torus and random of 256 and 1024 number of nodes. All distributions of resources and queries were uniform to keep the simulation and its analysis simple.

Several surprises appeared along the way. The most serious was the tendency of the topology management to produce fragmented networks. As these were not acceptable for later trials with DBFS a lot of attention was needed to tame the topology management algorithm. Understanding the joint effects of simultaneous overtakings might have been more difficult in more complex networks.

An outcome of this analysis is the impact that the overtaking on the performance. The overtaking should be “tuned” with caution, since it can have a more dramatic impact on the network topology than the simple additions or deletions of connections. In particular, in some cases, it results on “star” topologies, if the interest data is clustered.

Another major problem arose with the traffic estimation algorithm. For large networks with large TTL values, it was difficult to define traffic limit in a way that would keep the connections of a node within a reasonable amount but would not partition the network. This was also seen in the amount of neighbor additions compared to the amount of the neighbor removals; the amount of removals was always more than amount of additions.

The managed topologies give rather similar search performances in the close neighborhood (up to two hops away) independently of the initial topology. This shows that topology management is able (in the given setup) to make systematic changes. However, for larger neighborhoods the effect is lost. This implies also that even if the search performance improves from the original torus topology, the resulting performance is not really better than that for the original random grid for higher values of TTL. Obviously, one would like the topology management to perform better than simply random networks. A positive result was that even if the efficiency did not improve the amount of received replied close to the node and success rates increased. Thus the used TTL value could be decreased after topology management algorithms and node would still receive more replies.

It seems that the topology management cannot, in the current form and setup, do much more than create contacts to the neighbor's neighbor once. After that, the used thresholds for different operations are too tight to allow further progress. A small scale sensitivity analysis was made to test the effect of different traffic limits, but this did not change the situation. Presumably, the 80-90% threshold for overtaking is too high to allow iterated changes in the topology.

The parameters used in the simulation of topology management algorithms were selected so that they would work as well as possible for (almost) any test case. For that reason, for example, the initialization period before topology changes was kept quite long.

It is bit doubtful to generalize the findings from one simple setup, but at least the following observations may be valid also in a more general context:

- overtaking has a tendency to create star-like topologies (neighbor's neighbor of a star center node aims to joining the center node)
- simultaneous overtaking in the same neighborhood should be avoided as they may create severe traffic overloading (two stars coming to contact)

When studying topology management and adaptive search together it could be observed that topology management, even with its limited performance, was able to bring some clustering to the network to be exploited by the search. So, rather than competitive approaches, these should be seen as collaborators. This of course on precondition that topology management can be modified to exploit the biased data from directed searches.

7 CONCLUSIONS AND CONTRIBUTION OF THESIS

The peer-to-peer technology is utilized already in 1960s but in the last decade it has attained more publicity and it has challenged the traditional client-server architecture especially in the file-sharing applications. The main principle of the P2P networks is equal role of the nodes: all nodes may act as servers and clients. The pure unstructured P2P network is simple to implement and provides good scalability, but the resource discovery using flooding algorithms restricts this scalability. Resource discovery and topology are the main aspects affecting the efficiency. The efficiency of the search can be improved by changing the topology in a way that the resources can be found closer or by using non-flooding search algorithms. The concentration of this thesis is to study the performance of developed topology management algorithms [PV, PVI] and search algorithms [PI, PVII], but it also presents tools [PII, PIII, PIV] developed for the study.

The paper [PV] proposes a method combining four interest-based topology management algorithms for self-organizing the overlay topology in the pure unstructured peer-to-peer networks. The algorithms utilize the local information that the nodes collect about both their neighbors and also other nodes they know in the network. All the algorithms aim for nodes providing needed resources in the close neighborhood of the node that executes the algorithms. The simulations showed that this can be reached with certain parameters without significant load in the network or partition of the topology.

Solving the problem of a large amount of varied parameters was attempted with the use of neural networks. The topology management algorithms were further developed and the used parameters were tuned by using neural networks. The paper [PVI] introduces NeuroTopology algorithm. The algorithm

organizing the overlay topology of the peer-to-peer network is constructed by neural networks. Each node in the network defines by using neural network whether or not it establishes a connection to the node it knows. The local information the node knows about its potential neighbor is taken as input to the neural network, and the output defines whether a connection is to be established or not. As NeuroTopology is an interest-based algorithm, the local information that a node collects from other nodes includes the amount of resource replies arrived from the node or relayed by the node. NeuroTopology was trained using HDS algorithm and efficiency of the topology the NeuroTopology generated was tested in grid, power-law and random graph topologies. The results show that topologies generated by NeuroTopology are more efficient than other tested topologies. The challenge of the study was the complexity of the neural network.

The developed resource discovery algorithm called NeuroSearch is presented in the paper [PI]. NeuroSearch uses neural networks for generating a search algorithm similarly as NeuroTopology is used for constructing a topology algorithm. The algorithm decides to which neighbor nodes the resource discovery query is forwarded. NeuroSearch was simulated and tested with power-law graph networks and results were compared with BFS algorithm. It could be observed that by adapting the search with neural networks, about 50% more resources could be found with the same efficiency than with BFS for TTL2.

An algorithm was constructed to estimate the performance of resource discovery algorithms from theoretical upper limits. The algorithm uses k-Steiner minimum tree and is presented in the paper [PVII]. Five resource discovery algorithms, Breadth-First Search, Self-avoiding Random Walker, Highest-Degree Search, Dynamic Query Protocol and k-Steiner Minimum Tree, were analyzed by using power-law, random, and Gnutella2 topology. It was observed that the studied resource discovery algorithms performed by one to two orders of magnitude less efficiently than optimal search that was fully aware of the network topology and all the resources - in particular if a big fraction of the resources was to be found. For finding one sample of a popular resource, the difference was much smaller.

The first developed tool was a peer-to-peer middleware called Chedar which is based on pure peer-to-peer architecture and presented in paper [II]. The middleware is implemented in Java programming language and thus designed for heterogeneous environments. It offers an application interface for peer-to-peer applications. Chedar uses the TCP/IP protocol for communication and algorithm guaranteeing that the resource reply is forwarded back to the initiator of the query message if it still exists in the network. Chedar contains also topology management algorithms, which are used for self-organizing the topology and implements several search algorithms.

The ensure the reproducibility of the studies and avoid the problems the delays in real TCP connections and load of CPUs caused, all topology and resource discovery algorithms were studied in the P2PReal simulator described in paper [PIV]. P2PRealm is specially designed for optimizing neural networks

used in peer-to-peer networks. The user can define by input parameters the structure of the peer-to-peer networks and resource distribution and query patterns. Also requirements for algorithms trained by neural networks can be defined. The simulator provides communication by message passing, supports parallel computing and dynamic network.

The third tool is P2PStudio [PIII], which can be used to monitor, control, and visualize the peer-to-peer networks. P2PStudio is composed of a user interface (UI) and a server. The server handles the communication between user interface and peer-to-peer nodes. With P2PStudio, the user can send commands to a node or several nodes, modify resources, modify connections, and view a logical topology of the network, information about the last resource query, a node's parameter values, neighborhood distribution graph of the network, and a log of events in the network.

As the previous studies of topology management algorithms concentrated on the effect of few varied parameters, the main focus of the sixth chapter of this thesis was to define controlled simulations and analyze more systematically the performance of the generated networks. Simulations for a simple setup were run to study the effect of different parameters and to compare the topology management algorithms with the search algorithm using similar information.

The simulated setup showed it was impossible to define the parameters in a way that the developed algorithms would have improved all topologies and kept the networks connected. The partition of the network was varied from the few disconnected nodes to the several disconnected clusters. This problem arose in particular with larger networks (1024 nodes) and higher TTL values, which explains why the feature was not observed in the original paper [PV].

On the other hand, algorithms improved the efficiency, success rate, and especially the amount of received replies from the nodes close to the querier. This simple simulation kept all distribution of variables equal because different distributions of resources, interests, and queries tested with different topologies of varied node amounts, and with replication and dynamicity of the nodes might have given better results in some cases but would have done it impossible to make conclusions.

7.1 Contributions of the Author

Publications are produced in collaboration with other authors in a peer-to-peer research group. In the paper [PII], the author has implemented the middleware, designed and implemented topology algorithms along with the routing algorithm the middleware includes. The author has further developed topology algorithms and implemented and tested those in simulator [PV]. The NeuroTopology algorithm trained by neural networks [PVI] is defined by the author.

Author's contribution to P2PReal simulator [PIV] concerns defining the requirements for the simulator, and the author has implemented algorithms to the simulator and modified the simulator. In P2PStudio tool [PIII], the author

has participated in design of the tool and particularly defined the interface between the tool and peer-to-peer middleware.

The author participated in defining the neural network and inputs for NeuroSearch [PI] and resource discovery as a Steiner tree problem [PVII].

YHTEENVETO (FINNISH SUMMARY)

Hajautetut järjestelmät ovat muutaman vuosikymmenen jälkeen tekemässä paluuta asiakas/palvelin–arkkitehtuurista takaisin vertaisverkkoihin, joissa periaatteena on verkossa mukana olevien koneiden eli vertaisten tasa-arvoisuus. Vertaisverkossa mikä tahansa vertainen voi tarjota resursseja muille verkon vertaisille. Käyttämällä täysin keskittämätöntä vertaisverkkoarkkitehtuuria saavutetaan monia etuja, kuten skaalautuvuus ja alhaiset hankintakustannukset, mutta kohdataan myös haasteita. Suurin haaste liittyy käytettyihin ns. tulviviin (kaikkiin suuntiin lähettäviin) hakualgoritmeihin, jotka luovat verkkoon paljon kyselyviesteistä johtuvaa liikennettä. Hakualgoritmi luokin yleensä suurimman esteen vertaisverkon skaalautuvuudelle.

Haun tehokkuuteen vaikuttaa käytetyn hakualgoritmin lisäksi myös verkon topologia. Tutkimusten mukaan verkossa olevilta solmuilta löytyy kiinnostuksen kohteita, joten haun suhteen olisi tehokasta, jos samasta asiasta kiinnostuneet solmut olisivat lähellä toisiaan, jolloin kyselyviestien kulkemaa matkaa voitaisiin rajoittaa.

Tässä väitöskirjassa, jonka otsikko on ”Kiinnostuksiin pohjautuva topologian hallinta järjestämättömissä vertaisverkoissa”, esitellään topologian hallintaan kehitettyjä algoritmeja, joilla vertaiset voivat hallita yhteyksiään naapurustoonsa käyttäen hyväkseen tietoa, jota ne keräävät hakiessaan verkosta resursseja. Topologian hallinta-algoritmien lisäksi työssä tarkastellaan räätälöityjä hakualgoritmeja ja niiden sovittamista topologian hallintaan. Edellä mainittujen menetelmien tutkimukseen soveltuvia simulointityökaluja esitellään sekä yleisesti että käymällä läpi esimerkinomainen simulointikoesarja.

REFERENCES

Abraham, A., Ye, B., Xian, C., Liu, H. & Pant, M. 2007. Multi-objective Peer-to-Peer Neighbor-Selection Strategy Using Genetic Algorithm. Lecture Notes in Computer Science, Vol. 4873, pp. 443-451.

Adamic, L. A., Lukose, R. M., Puniyani, A. R. & Huberman, B. A. 2001. Search in power-law networks. Physical Review E, Vol. 64, Is. 4.

Adar, E. & Huberman, B. A. 2000. Free Riding on Gnutella. First Monday, Vol. 5, No. 10.

Agrawal, A. & Casanova, H. 2003. Clustering Hosts in P2P and Global Computing Platforms. In Proceedings of the 3rd International Symposium on Cluster Computing and the Grid, pp. 367-373.

Airiau, S., Sen, S. & Dasgupta, P. 2006. Effect of Joining Decisions on Peer Clusters. In Proceedings of the Fifth international Joint Conference on Autonomous Agents and Multiagent Systems, pp. 609-615.

Albert, R. & Barabasi, A-L. 2002. Statistical Mechanics of Complex Networks. Reviews of Modern Physics, Vol. 74.

Alima, L. O., Mesaros, V., Van Roy, P. & Haridi, S. 2002. NetProber: A Component for Enhancing Efficiency of Overlay Networks in P2P Systems. In Proceedings of the Second international Conference on Peer-To-Peer Computing, pp. 25-32.

Androutsellis-Theotokis, S. & Spinellis., D. 2004. A Survey of Peer-to-Peer Content Distribution Technologies. ACM Computing Surveys, Vol. 36, No. 4 (December 2005), pp. 335-371.

Asvanund, A., Bagla, S., Kapadia, M. H., Krishnan, R., Smith, M. D. & Telang, R. 2003. Intelligent Club Management in Peer-to-Peer Networks. 1st Workshop on Economics of Peer-to-Peer Systems.

Berstein, D. S., Feng, Z., Levine, B. N. & Zilberstein, S. 2003. Adaptive Peer Selection. Lecture Notes in Computer Science, Vol. 2735, pp. 237-246.

Broekstra, J., Ehrig, M., Haase, P., Van Harmelen, F., Kampman, A., Sabou, M., Siebes, R., Staab, S., Stuckenschmidt, H. & Tempich, C. 2003. A Metadata Model for Semantic-Based Peer-to-Peer Systems. In Proceedings of the WWW'03 Workshop on Semantics in Peer-to-Peer and Grid Computing.

Bustamante, F. E. & Qiao, Y. 2004. Friendships that last: Peer lifespan and its role in P2P protocols. In Web Content Caching and Distribution: Proceedings of the 8th international Workshop, pp. 233-246.

Chawathe, Y., Ratsanamy, S., Breslau, L., Lanham, N. & Shenker, S. 2003, Making Gnutella-like P2P Systems Scalable. In Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications, pp. 407-418.

Ciglaric., M. & Vidmar, T. 2002. Position of Modern Peer-to-Peer Systems in the Distributed Systems Architecture. 11th Mediterranean Electrotechnical Conference, pp. 341-345.

Cohen, E. & Shenker, S. 2002. Replication Strategies in Unstructured Peer-to-Peer Networks. In Proceedings of the 2002 conference on Applications,

technologies, architectures, and protocols for computer communications, August 2002, pp. 177-190.

Condie, T., Kamvar, S. D. & Garcia-Molina, H. 2004. Adaptive Peer-to-Peer Topologies. In Proceedings of the Fourth international Conference on Peer-To-Peer Computing, pp. 53-62.

Coulouris, G., Dollimore, J. & Kindberg., T. 2005. Distributes Systems: Concepts and Design. Pearson Education Limited, England.

Cooper, B. F. 2004. A Content Model for Evaluating Peer-to-Peer Searching Techniques. In Proceedings of the 5th ACM/IFIP/USENIX international Conference on Middleware, pp. 18-37.

Cooper, B. F. & Garcia-Molina, H. 2005. Ad Hoc, Self-Supervising Peer-to-Peer Search Networks. ACM Transactions on Information Systems, Vol. 23, No. 2 (April 2005).

Crespo, A. & Garcia-Molina, H. 2002, Routing Indices for Peer-to-Peer Systems. In Proceedings of 22nd International Conference on Distributed Computing Systems, pp. 23-32.

Crespo, A. & Garcia-Molina, H. 2005. Semantic Overlay Networks for P2P Systems. Lecture Notes in Computer Science, Volume 3601, pp. 1-13.

DePaoli, F. & Mariani, L. 2004. Dependability in Peer-to-Peer Systems. IEEE Internet Computing, Vol. 8, No. 4 (July-August 2004), pp. 54-61.

Fisk, A. 2003. Gnutella Dynamic Query Protocol v0.1. Gnutella Developer's Forum. http://www.ic.unicamp.br/~celio/peer2peer/gnutella-related/gnutella-dynamic-protocol.htm, 5.12.2012.

Ge, Z., Figueiredo, D. R., Jaiswal, S., Kurose, J. & Towsley, D. 2003. Modeling Peer-Peer File Sharing Systems. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications, Vol. 3, pp. 2188-2198.

Ghanea-Hercock, R. A., Wang, F. & Sun, Y. 2006. Self-Organizing and Adaptive Peer-to-Peer Network. IEEE Transactions on Systems, Man, and Cybernetics – part B: Cybernetics, Vol. 36, Nro. 6 (December 2006).

Habib, A. & Chuang, L. 2006. Service Differentiated Peer Selection: An Incentive Mechanism for Peer-to-Peer Streaming. IEEE Transactions on Multimedia, Vol. 8, No.3 (June 2006), pp. 610-621.

Hales, D. 2005. Self-organising, Open and Cooperative P2P Societies - From Tags to Networks. Lecture Notes in Computer Science, Vol. 3464, pp. 267-285.

Handurukande, S., Kermarrec, A-M., Fessant, F. L. & Massoulie, L. 2004. Exploiting Semantic Clustering in the eDonkey P2P Network. In Proceedings of the 11th Workshop on ACM SIGOPS European Workshop.

Hu, T. H. & Sereviratne, A. 2003. General Clusters in Peer-to-Peer Networks. The 11th IEEE International Conference on Networks, pp. 277-282.

Idris, T. & Altmann, J. 2006. A Market-Managed Topology Formation Algorithm for Peer-to-Peer File Sharing Networks. In Proceedings of 5th International Conference on Internet Charging and QoS Technologies, pp. 61-77.

Iles, M. & Deugo, M. 2003. Adaptive Resource Location in a Peer-to-Peer Network. In Proceedings of the 16th international Conference on Developments in Applied Artificial intelligence, pp. 604-613.

Iles, M. & Deugo, M. 2002. A Search for Routing Strategies in a Peer-to-Peer Network Using Genetic Programming. In Proceedings of 21st IEEE Symposium on Reliable Distributed Systems, pp. 341-346.

Kalogeraki, V., Gunopulos, D. & Zeinalipour-Yazti, D. 2002. A Local Search Mechanism for Peer-to-Peer Networks. In Proceedings of the Eleventh international Conference on Information and Knowledge Management, pp. 300-307.

Khambatti, M., Ryu, K. D. & Dasgupta, P. 2004. Structuring Peer-to-Peer Networks Using Interest-Based Communities. Lecture Notes in Computer Science, Vol. 294a, pp. 48-63.

Klemm, A., Lindemann, C., Vernon, M. K. & Waldhorst, O. P. 2004. Characterizing the Query Behavior in Peer-to-Peer File Sharing Systems. In Proceedings of the 4th ACM SIGCOMM Conference on internet Measurement, pp.55-67.

Kojima, K. 2003. Grouped Peer-to-Peer Networks and Self-Organization Algorithm. In Proceedings of IEEE International Conference on Systems, Man and Cybernetics, Vol. 3, pp.2970-2976.

Lai, K., Feldman, M., Stoica, I. & Chuang, J. 2003. Incentives for Cooperation in Peer-to-Peer Networks. In Workshop on Economics of Peer-toPeer Systems.

Liu, H., Abraham, A. & Badr, Y. 2010. Neighbor Selection in Peer-to-Peer Overlay Networks: A Swarm Intelligence Approach. Computer Communications and Networks, Part 4, pp. 405-431.

Liu, H., Abraham, A. & Xhafa, F. 2008. Peer-to-Peer Neighbor Selection Using Single and Multiobjective Population-based Metaheuristics, Metaheuristics for Scheduling: Distributed Computing Environments. Studies in Computational Intelligence, Springer Verlag, Germany, pp. 323-340.

Liu, Y., Liu, X., Xiao, L., Ni, L. M. & Zhang, X. 2004A. Location-Aware Topology Matching in P2P Systems. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies , Vol. 4, pp. 2220-2230.

Liu, Y., Xiao, L, Esfahanian, A-E. & Ni, L. M. 2005A. Approaching Optimal Peer-to-Peer Overlays. In Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 407-414.

Liu, Y., Xiao, L., Liu, X., Ni, L. M. & Zhang, X. 2005B. Location Awareness in Unstructured Peer-to-Peer Systems. IEEE Transactions on Parallel and Distributed Systems, Vol. 16, Nro 2 (February 2005).

Liu, Y., Zhuang, Z., Xiao, L. & Ni, L. M. 2004B. A Distributed Approach to Solving Overlay Mismatching Problem. In Proceedings of the 24th International Conference on Distributed Computing Systems, pp. 132-139.

Liu, Y., Zhuang, Z., Xiao, L. & Ni, L. M. 2003. AOTO: Adaptive Overlay Topology Optimization in Unstructured P2P Systems. IEEE Global Telecommunications Conference, Vol. 7, pp. 4186-4190.

Lo, V., Zhou, D., Liu, Y., GauthierDickey, C. & Li, J. 2005. Scalable Supernode Selection in Peer-to-Peer Overlay Networks. In Proceedings of the Second international Workshop on Hot Topics in Peer-To-Peer Systems, pp. 18-27.

Lu, T., Fang, B., Sun, Y., Cheng, X. & Guo, L. 2005. Building Scale-Free Overlay Mix Networks with Small-World Properties. In Proceedings of the Third international Conference on information Technology and Applications, pp. 529-534.

Lv, Q., Cao, P., Cohen, E., Li, K. & Shenker, S. 2002A. Search and Replication in Unstructured Peer-to-Peer Networks. In Proceedings of the 16th International Conference on Supercomputing, pp. 84-95.

Lv, Q., Ratnasamy, S. & Shenker, S. 2002B. Can Heterogeneity Make Gnutella Scalable?. In Proceedings of the first International Workshop on Peer-to-Peer Systems, pp. 94-103.

Löser, A., Naumann, F., Siberski, W., Nejdl, W. & Thaden, U. 2003. Semantic Overlay Clusters within Super-Peer Netwoks. In Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing, pp. 33-47.

Massoulie, L. & Kermarrec, A.-M. 2003. Network Awareness and Failure Resilience on Self-Organising Overlay Networks. In Proceedings of the 22nd International Symposium on Reliable Distributed Systems, pp. 47-55.

Meng, S., Shi, C., Han, D., Zhu, X. & Yu, Y. 2006. A Statistical Study of Today's Gnutella. Lecture Notes in Computer Science, Vol. 3841, pp. 189-200.

Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M., Brunkhorst, I. & Löser, A. 2003. In Proceedings of the 12th International Conference on World Wide Web, pp. 536-543.

Newman, M. E. J. 2003. The Structure and Function of Complex Networks. SIAM Review, Vol. 45, Issue 2, pp. 167-256.

Ng, C. H. & Sia, K. C. 2002. Peer Clustering and Firework Query Model. In Poster Proc. of the 11th International World Wide Web Conference.

Ng, W. S., Ooi; B. C. & Tan K. 2002. BestPeer: A Self-Configurable Peer-to-Peer System. In Proceedings of the 18th International Conference on Data Engineering, pp. 272.

Ni, L. M. & Liu, Y. 2004. Efficient Peer-to-Peer Overlay Construction. In Proceedings of the IEEE International Conference on E-Commerce Technology for Dynamic E-Business, pp. 314-317.

Niu, C., Wang, J. & Shen, R. 2007. A Trust-Enhanced Topology Adaptation Protocol for Unstructured P2P Overlays. Third International Conference on Semantics, Knowledge and Grid, pp. 200-205.

Oram, A. 2001. Peer-to-Peer: Harnessing the Power of Distributed Technologies. O'Reilly & Associates, Inc.

Ramanathan, M. K., Kalogeraki, V. & Pruyne, J. 2002. Finding Good Peers in Peer-to-Peer Networks. In Proceedings of the 16th International Symposium on Parallel and Distributed Processing, pp. 24-31.

Ramaswamy, L., Gedik, B. & Liu, L. 2005. A Distributed Approach to Node Clustering in Decentralized Peer-to-Peer Networks. IEEE Transactions on Parallel and Distributed Systems, Vol. 16, No. 9 (September 2005), pp. 814-829.

Ramaswamy, L., Gedik, B. & Liu, L. 2003A. Connectivity Based Node Clustering in Decentralized Peer-to-Peer Networks. In Proceedings of Third International Conference on Peer-to-Peer Computing, pp. 66-73.

Ramaswamy, L. & Liu, L. 2003B. Free Riding: A New Challenge to Peer-to-Peer File Sharing Systems. In Proceedings of the 36th Annual Hawaii International Conference on System Sciences, January 2003.

Rasti, A. H., Stutzbach, D. & Rejaie, R. 2006. On the Long-term Evolution of the Two-Tier Gnutella Overlay. In Proceedings of 25th IEEE International Conference on Computer Communications, pp. 1-6.

Ratsanamy, S., Francis, P., Handley, M., Karp, R. & Shenker, S. 2001. A scalable Content-addressable Network. In Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 161-172.

Ratnasamy, S., Handley, M., Karp, R. & Shenker, S. 2002. Topologically-Aware Overlay Construction and Server Selection. In Proceedings of Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 1190-1199.

Sakaryan, G. & Unger, H. 2003A. Topology Evolution in P2P Distributed Networks. In Proceedings of IASTED: Applied Informatics, Austria, pp. 791-796.

Sakaryan, G. & Unger, H. 2003B. Influence of the Decentralized Algorithms on Topology Evolution in P2P Distributed Networks. In Proceedings of Design, Analysis, and Simulation of Distributed Systems.

Schlosser, M. T., Condie, T. E. & Kamvar, S. D. 2002. Simulating a File-Sharing P2P Network. In First Workshop on Semantics in P2P and Grid Computing.

Schollmeier, R. 2001. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In Proceedings of the First International Conference on Peer-to-Peer Computing, pp. 101-102.

Shao, Y. & Wang, R. 2005. BuddyNet: History-Based P2P Search. In Proceedings of 27th European Conference on IR Research, pp. 23-37.

Singh, A. & Haahr, M. 2007. Decentralized Clustering In Pure P2P Overlay Networks Using Schelling's Model. IEEE International Conference on Communications, pp.1860-1866.

Sripanidkulchai, K., Maggs, B. & Zhang, H. 2003. Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems. Twenty-Second

Annual Joint Conference of the IEEE Computer and Communications., pp. 2166-2176 vol.3.

Srivatsa, M., Gedik, B. & Liu, L. 2006. Large Scaling Unstructured Peer-to-Peer Networks with Heterogeneity-Aware Topology and Routing. IEEE Transactions on Parallel and Distributed Systems, Vol. 17, Nro. 11 (November 2006).

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F. & Balakrishnan, H. 2001. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications.

Stutzbach., D. & Rejaie, R. 2005A. Capturing Accurate Snapshots of the Gnutella Network. In Proceeding of 24th Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 2825-2830.

Stutzbach, D., Rejaie, R. & Sen, S. 2008. Characterizing Unstructured Overlay Topologies in Modern P2P File-Sharing Systems. IEEE/ACM Transactions on Networking, Vol. 16, No. 2 (April 2008), pp. 267-280.

Stutzbach., D. & Rejaie, R. 2005B. Evaluating the Accuracy of Captured Snapshots by Peer-to-Peer Crawlers. Lecture Notes in Computer Science, Vol. 3431, pp. 353-357.

Sun, S., Abraham, A., Zhang, G. & Liu, H. 2007. A Particle Swarm Optimization Algorithm for Neighbor Selection in Peer-to-Peer Networks. In Proceedings of the 6th International Conference on Computer Information Systems and Industrial Management Applications, pp. 166-172.

Sun, Q. & Garcia-Molina, H. 2004. SLIC: A Selfish Link-Based Incentive Mechanism for Unstructured Peer-to-Peer Networks. In Proceedings of the 24th international Conference on Distributed Computing Systems, pp. 506–515.

Tanenbaum, A. S. & Van Steen, M. 2002. Distributes Systems: Principles and Paradigms. Prentice-Hall.

Tsoumakos, D. & Roussopoulos, N. 2003A. A Comparison of Peer-to-Peer Search Methods. In Proceedings of 6th International Workshop on Web and Databases, pp. 61-66.

Tsoumakos, D. & Roussopoulos, N. 2003B. Adaptive Probabilistic Search for Peer-to-Peer Networks. In Proceedings of the 3rd international Conference on Peer-To-Peer Computing.

Tsoumakos, D. & Roussopoulos, N. 2006. Analysis and Comparison of P2P Search Methods. In Proceedings of the 1st international conference on Scalable information system.

Voulgaris, S., Kermarrec, A.-M. & Massoulie, L. 2004. Exploiting semantic proximity in peer-to-peer content searching. In Proceedings of 10th IEEE International Workshop on Future Trends of Distributed Computing Systems, pp. 238-243.

Wan, H., Ishikawa, N. & Hjelm, J. 2005. Autonomous Topology Optimization for Unstructured Peer-to-Peer Networks. In Proceedings of the 2005 11th International Conference on Parallel and Distributed Systems, pp. 488-494.

Wang, Y., Yun, X. & Li, Y. 2007. Analyzing the Characteristics of Gnutella Overlays. In Proceedings of the International Conference on Information Technology, pp. 1095–1100.

Watts, D. J. & Strogatz, S. H. 1998. Collective dynamics of 'small-world' networks. Nature, Vol. 393 (June 1998), pp. 440-442.

Xiao, L., Liu, Y. & Ni, L. M. 2005. Improving Unstructured Peer-to-Peer Systems by Adaptive Connection Establishment. IEEE Transactions on Computers, Vol. 54, Nro. 9 (September 2005), pp. 1091-1103.

Xie. C., Guo, S., Rejaie, R. & Pan, Y. 2007. Examining Graph Properties of Unstructured Peer-to-Peer Overlay Topology. IEEE Global Internet Symposium, pp. 13-18.

Yang, B. & Garcia-Molina, H. 2003. Designing a Super-Peer Network. In Proceedings of the 19th International Conference on Data Engineering, pp. 49-60.

Yang, B. & Garcia-Molina, H. 2002B. Improving Search in Peer-to-Peer Networks. In Proceedings of 22nd International Conference on Distributed Computing Systems, pp. 5-14.

Yang, S. J. H. & Chen, I. Y. L. 2008. A social network-based system for supporting interactive collaboration in knowledge sharing over peer-to-peer network. International Journal of Human-Computer Studies, Vol. 66, Issue 1 (January 2008), pp. 36-50.

Zhang, H., Zhang, L., Shan, X. & Li, V. O. K. 2007. Probabilistic Search in P2P Networks with High Node Degree Variation. IEEE International Conference on Communications, pp. 1710-1715.

Zhang, X. Y., Zhang, Q., Zhang, Z., Song, G. & Zhu, W. 2004. A Construction of Locality-Aware Overlay Network: mOverlay and Its Performance. IEEE Journal on Selected Areas in Communications, Vol. 22, No. 1 (January 2004).

Zhao, B. Y., Ling, H., Stribling, J., Rhea, S. C., Joseph, A. D. & Kubiatowicz, J. D. 2004. Tapestry: a Resilient Global-scale Overlay for Service Deployment. IEEE Journal on Selected Areas in Communications, Vol. 22, Issue 1 (January 2004), pp. 41-53.

Zheng, W., Zhang, S., Ouyang, Y., Makedon, F. & Ford, J. 2005. Node Clustering Based on Link Delay in P2P Networks. In Proceedings of the 2005 ACM Symposium on Applied Computing, pp. 744 -749.

APPENDIX

This appendix presents the experiment setup and detailed data used in the graphs describing the results of the simulations in Chapter 6.

The simulation was set up as follows:

Nodes and resources

- Grids of N=2^l nodes were used (N=256, 1024, l=8, 10).
- There were M=2^12 resources in the network (identified with a 12 bit binary key). So each resource has a unique identifier, l first bits defining the node and the rest identifying the resource within the node. That is, there were 16 or 4 resources on each node.
- The nodes were divided to 8 interest groups using the first three leading bits in the resource/node key.
- The node numbering implied by the resource keys was independent of numberings used in construction of the grid topologies or in scheduling topology management operations.

Queries

- Each query was designed so that it matched to 2^5=32 resources distributed to varying amount of nodes. For each query a random mask was created that fixed 7 of the 12 bits (leaving 5 bits free). Thus the query matched to resources that were distributed to 2, 4, 8, 16 or 32 nodes (for N=256), (8, 16 or 32 for N=1024).
- For queries inside the interest group, the first three bits of the mask (defining the nodes in the interest group) were forced to be the same as for the querying node.

Simulation runs

- 20 independent replications were made of each simulation case.
- One simulation run consisted of 280*N queries (for N=256, 1024). For simulations of at equilibrium state for managed topologies the simulation consisted of 140*N queries. The equilibrium state was achieved with 140*n queries .When simulating DBFS, z*N queries were first made using BFS to collect data for deciding about the preferred directions (for z = 2, 4, 6).
- For topology management z*N queries were first made to collect data about the resources in the search neighborhood and the traffic at nodes (for z = 2, 4, 6).
- The traffic estimation period was z*N queries (for z = 2, 4, 6).
  - Each node was given random value within (1 - z*N) which defined the cycle when the node checked its traffic.
  - Overloading was checked more often: every 16th cycle the node was checking its overload situation i.e. whether its traffic was more than the upper traffic limit (u*z*N, for u = 0.4, 0.6, 0.8) allowed.
  - The lower traffic limit was 20% of upper traffic limit.

- Also for overtaking was defined the initiation rounds 2*z*N when the node is collecting data.
  - After that the node receiving replies checked if one of its neighbors' neighbor provides more than o percent of the replies (for o = 80, 90).
  - Since that the overtaking was checked after the node has sent k queries (k = 10, 20).

Observed variables

- The output of the simulation was monitored using results averaged over 1024 queries. From this data the following indicators were computed
  - Q= average number of query messages/node for each level of TTL
  - H= average number of found resources/query for each level of TTL
  - Efficiency E=H/Q
  - Number of queries without any replies (failed queries)
  - For topology management the amounts of additions, removals and overtakings per 1024 queries
- In addition to point estimates (sample averages), we report the standard deviations of Q and H on the node level. Confidence intervals can be obtained by multiplying the standard deviations by 1.96/sqrt(20*N*#query) where #query is the amount of queries/node in the run, , (i.e. by 0.0016 for N=256 and 0.0008 for N=1024).

TABLE 10 BFS in the static networks of 256 nodes.

	Torus 256					**Random 256**				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**	**Q**	**S_Q**	**H**	**S_H**	**E**
1	4,000	0,000	0,480	1,380	0,120	4,094	5,901	0,506	1,429	0,124
2	12,000	0,000	0,980	2,796	0,082	16,477	49,620	1,918	5,812	0,117
3	24,000	0,000	1,429	2,603	0,060	62,374	275,430	6,084	21,919	0,098
4	36,000	0,000	1,964	3,443	0,055	188,518	944,204	11,337	25,334	0,061
5	48,000	0,000	2,459	3,737	0,051	308,471	968,016	8,380	23,344	0,028
6	60,000	0,000	2,935	3,153	0,049	174,654	466,170	2,439	21,583	0,014
7	72,000	0,000	3,368	3,470	0,047	34,211	328,897	0,437	7,554	0,012

TABLE 11 BFS in the static torus network of 1024 nodes.

	Torus 1024				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	4,0000	0,0000	0,1238	0,3590	0,0309
2	12,0000	0,0000	0,2485	0,5425	0,0207
3	24,0000	0,0000	0,3720	0,6231	0,0155
4	36,0000	0,0000	0,4988	0,7687	0,0139
5	48,0000	0,0000	0,6266	0,8581	0,0131
6	60,0000	0,0000	0,7477	0,9052	0,0124
7	72,0000	0,0000	0,8661	1,0559	0,0120

TABLE 12 BFS in the static random network of 1024 nodes.

	Random 1024				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	4,1062	3,3708	0,1283	0,4052	0,0312
2	16,5716	24,5754	0,5085	1,0338	0,0307
3	65,9173	140,1060	1,9328	4,3911	0,0293
4	246,7508	658,6451	6,0812	13,2294	0,0247
5	744,4459	2064,2749	11,4857	14,8145	0,0155

The following result of DBFS algorithm use initialization round 4 and the algorithm selected one neighbor only in the first hop.

TABLE 13 DBFS in the torus network of 256 nodes.

	Torus 256, TTL 3			Torus 256, TTL 5			Torus 256, TTL 7		
Hops	**H**	**S_H**	**E**	**H**	**S_H**	**E**	**H**	**S_H**	**E**
1	0,238	0,848	0,238	0,180	0,989	0,180	0,161	0,757	0,161
2	0,560	1,579	0,187	0,489	1,618	0,163	0,442	1,653	0,147
3	1,009	1,993	0,112	0,959	1,837	0,107	0,907	1,911	0,101
4				1,453	3,224	0,069	1,432	2,903	0,068
5				2,021	2,294	0,061	2,012	2,822	0,061
6							2,493	2,769	0,052
7							2,964	3,959	0,049

TABLE 14 DBFS in the torus network of 1024 nodes.

	Torus 1024, TTL 5			Torus 1024, TTL 7		
Hops	**H**	**S_H**	**E**	**H**	**S_H**	**E**
1	0,0419	0,2132	0,0419	0,0387	0,2036	0,0386
2	0,1115	0,3556	0,0372	0,1051	0,3540	0,0350
3	0,2372	0,5309	0,0263	0,2322	0,4992	0,0258
4	0,3633	0,6740	0,0173	0,3571	0,6570	0,0170
5	0,5152	0,7560	0,0156	0,5100	0,7314	0,0155
6				0,6336	0,8509	0,0132
7				0,7572	0,9304	0,0126

TABLE 15 DBFS with TTL 3 in the random network of 256 nodes.

	Random 256, TTL 3				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	1,000	0,000	0,160	0,801	0,160
2	5,239	9,992	0,811	2,326	0,155
3	21,976	70,192	3,003	8,451	0,137

TABLE 16 DBFS with TTL 5 in the random network of 256 nodes.

	Random 256, TTL 5				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	1,000	0,000	0,140	0,494	0,140
2	5,353	9,240	0,712	2,064	0,133
3	22,717	66,310	2,815	7,834	0,124
4	85,093	364,731	8,044	26,233	0,095
5	238,303	1098,438	12,075	15,548	0,051

TABLE 17 DBFS with TTL 7 in the random network of 256 nodes.

	Random 256, TTL 7				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	1,000	0,000	0,131	0,686	0,131
2	5,345	9,060	0,713	2,184	0,133
3	22,654	68,135	2,802	8,310	0,124
4	84,837	374,991	8,004	25,293	0,095
5	237,821	1115,128	12,048	15,603	0,051
6	313,864	715,291	6,152	30,864	0,020
7	112,924	641,609	1,133	14,396	0,010

TABLE 18 DBFS with TTL 3 in the random network of 1024 nodes.

	Random 1024, TTL 3				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	1,0000	0,0000	0,0385	0,2079	0,0384
2	5,0587	5,4916	0,1864	0,5024	0,0369
3	21,2489	36,9272	0,7615	1,5643	0,0358

TABLE 19 DBFS with TTL 5 in the random network of 1024 nodes.

	Random 1024, TTL 5				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	1,0000	0,0000	0,0321	0,1865	0,0319
2	5,4971	5,9796	0,1756	0,4663	0,0320
3	23,6499	39,5265	0,7435	1,6260	0,0314
4	94,7559	216,9430	2,7842	5,8515	0,0294
5	347,2859	948,4843	8,1314	16,9088	0,0235

The following results of topology management algorithms use interval of traffic estimation value 6.

TABLE 20 Topology management in torus network of 256 nodes without overtaking, with TTL 5 and upper traffic limit 40%.

	Torus 256, TTL 5				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	3,194	7,752	0,460	1,393	0,145
2	7,464	43,336	0,760	2,773	0,103
3	13,876	96,560	1,055	4,754	0,077
4	20,997	143,391	1,561	4,704	0,076
5	28,699	185,116	2,009	4,826	0,071

TABLE 21 Topology management in torus network of 256 nodes without overtaking, with TTL 7 and upper traffic limit 60%.

	Torus 256, TTL 7				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	2,811	11,515	0,448	1,435	0,161
2	5,833	59,204	0,634	3,723	0,112
3	10,071	133,719	0,818	6,880	0,084
4	14,456	207,483	1,201	7,862	0,088
5	19,248	277,863	1,514	9,428	0,083
6	24,378	345,132	1,891	11,463	0,082
7	29,797	409,866	2,211	12,395	0,078

TABLE 22 Topology management in torus network of 256 nodes, with TTL 3, upper traffic limit 60% and overtaking 80%.

	Torus 256, TTL 3				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	3,294	10,471	1,806	8,006	0,557
2	14,233	156,165	3,967	27,211	0,294
3	45,432	977,407	5,811	45,013	0,143

TABLE 23 Topology management in torus network of 256 nodes, with TTL 5, upper traffic limit 60 and overtaking 80%.

	Torus 256, TTL 5				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	2,023	21,722	0,760	3,313	0,398
2	14,977	282,550	2,620	35,183	0,188
3	29,758	1474,052	2,741	72,536	0,116
4	35,026	2305,353	1,985	73,946	0,083
5	23,991	1382,668	1,303	65,194	0,072

TABLE 24 Topology management in torus network of 256 nodes, with TTL 7, upper traffic limit 60% and overtaking 80%.

	Torus 256, TTL 7				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	1,919	22,562	0,648	3,793	0,359
2	13,141	323,244	2,077	33,654	0,179
3	23,303	1463,376	1,869	71,443	0,107
4	27,401	2109,108	1,444	74,166	0,074
5	20,213	1387,215	1,025	64,725	0,060
6	13,737	1111,551	0,718	55,673	0,034
7	9,904	945,704	0,462	40,084	0,022

The following results of topology management algorithms are from simulations after improvement of the algorithms and the addition of a new parameter, the overtaking period. In the following tables, overtaking period value is 20, interval of traffic estimation is 6 and overtaking percent is 90%.

TABLE 25 Topology management in torus network of 256 nodes without overtaking, with TTL 3 and upper traffic limit 60%.

	Torus 256, TTL 3				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	3,991	1,582	0,909	6,286	0,228
2	14,237	33,619	1,478	7,885	0,103
3	38,355	207,663	2,526	16,418	0,065

TABLE 26 Topology management in torus network of 256 nodes, with TTL 3 and upper traffic limit 40%.

	Torus 256, TTL 3				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	3,905	2,836	0,908	6,283	0,233
2	13,336	20,042	1,402	6,528	0,105
3	34,365	137,636	2,273	12,105	0,066

TABLE 27 Topology management in torus network of 256 nodes, with TTL 5 and upper traffic limit 60%.

	Torus 256, TTL 5				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	3,662	9,075	0,543	1,966	0,150
2	10,624	38,143	0,938	2,653	0,089
3	21,887	78,570	1,457	3,450	0,067
4	34,301	101,352	2,123	4,506	0,062
5	47,589	118,397	2,789	6,736	0,059

TABLE 28 Topology management in torus network of 256 nodes, with TTL 7 and upper traffic limit 60%.

	Torus 256, TTL 7				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	2,271	18,869	0,589	2,699	0,277
2	7,961	93,217	1,012	13,257	0,127
3	15,291	201,125	1,625	23,988	0,106
4	19,162	230,368	1,923	21,638	0,103
5	19,305	299,223	1,828	14,318	0,102
6	18,196	450,941	1,612	21,906	0,102
7	17,926	610,133	1,455	32,482	0,100

TABLE 29 Topology management in torus network of 256 nodes without overtaking, with TTL 7 and upper traffic limit 60%.

	Torus 256, TTL 7				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	2,811	11,515	0,448	1,435	0,161
2	5,833	59,204	0,634	3,723	0,112
3	10,071	133,719	0,818	6,880	0,084
4	14,456	207,483	1,201	7,862	0,088
5	19,248	277,863	1,514	9,428	0,083
6	24,378	345,132	1,891	11,463	0,082
7	29,797	409,866	2,211	12,395	0,078

TABLE 30 Topology management in torus network of 1024 nodes, with TTL 3 and upper traffic limit 60%.

	Torus 1024, TTL 3				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	3,997	2,577	0,279	2,309	0,070
2	18,710	114,816	0,560	5,218	0,029
3	67,469	760,773	1,248	15,410	0,018

TABLE 31 Topology management in torus network of 1024 nodes, with TTL 5 and upper traffic limit 60%.

	Torus 1024, TTL 5				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	3,9957	1,2803	0,1521	0,5888	0,0381
2	13,5073	25,5516	0,3039	1,0695	0,0225
3	32,9542	144,7854	0,5511	3,0322	0,0166
4	57,5752	359,8522	0,8488	5,9871	0,0147
5	85,0290	632,1700	1,1534	9,0471	0,0135

TABLE 32 Topology management in torus network of 1024 nodes, with TTL 7 and upper traffic limit 60%.

	Torus 1024, TTL 7				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	3,9998	0,2406	0,1252	0,3635	0,0313
2	12,0587	1,4614	0,2518	0,5603	0,0209
3	24,4932	10,1391	0,3820	0,6652	0,0156
4	37,0757	22,1571	0,5151	0,8758	0,0139
5	49,7885	36,8094	0,6531	1,0348	0,0131
6	62,6500	54,3165	0,7843	1,2199	0,0125
7	75,6513	74,3509	0,9194	1,6766	0,0121

TABLE 33 Topology management in random network of 256 nodes, with TTL 3 and upper traffic limit 60%.

	Random 256, TTL 3				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	3,615	8,398	0,689	2,869	0,193
2	14,948	57,346	1,955	5,680	0,131
3	54,991	355,044	5,316	23,937	0,097

TABLE 34 Topology management in random network of 1024 nodes, with TTL 3 and upper traffic limit 60%.

	Random 1024, TTL 3				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	4,0901	4,1550	0,1768	0,7926	0,0433
2	22,6478	87,3742	0,7472	3,4540	0,0329
3	118,7900	757,4278	3,1659	17,5324	0,0269

TABLE 35 Topology management in random network of 1024 nodes, with TTL 5 and upper traffic limit 60%.

	Random 1024, TTL 5				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	2,8935	12,0626	0,1323	0,4271	0,0464
2	10,0275	70,8356	0,3188	2,1924	0,0319
3	29,9735	404,2282	0,9033	11,4347	0,0303
4	87,7331	1814,8314	2,4221	39,3838	0,0286
5	229,9101	5558,4151	5,1703	62,3458	0,0250

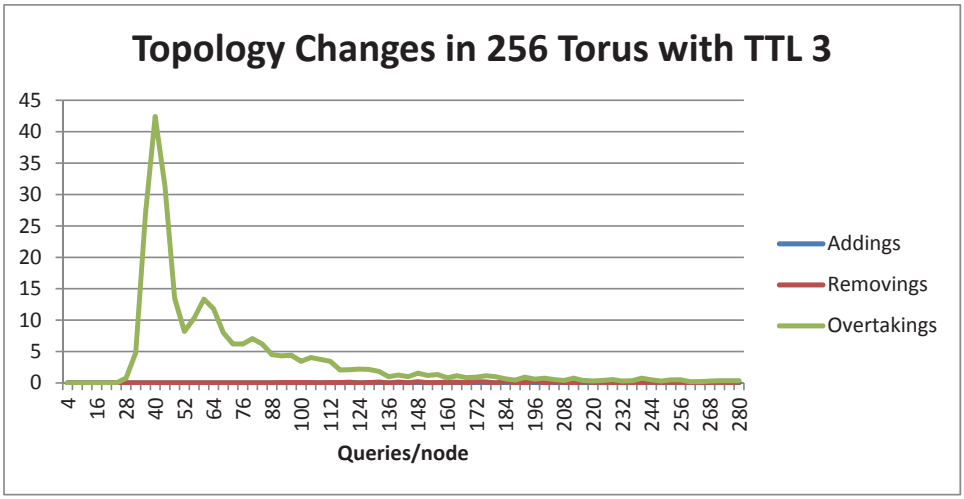

FIGURE 27 The amount of topology changes during the simulation of topology management algorithms in torus network of 256 nodes with TTL 3.

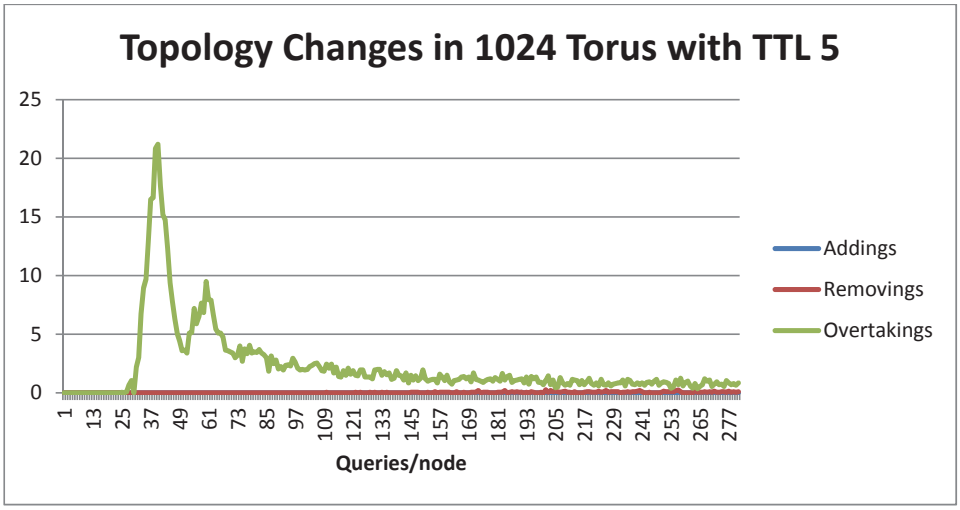

FIGURE 28 The amount of topology changes during the simulation of topology management algorithms in torus network of 1024 nodes with TTL 5.

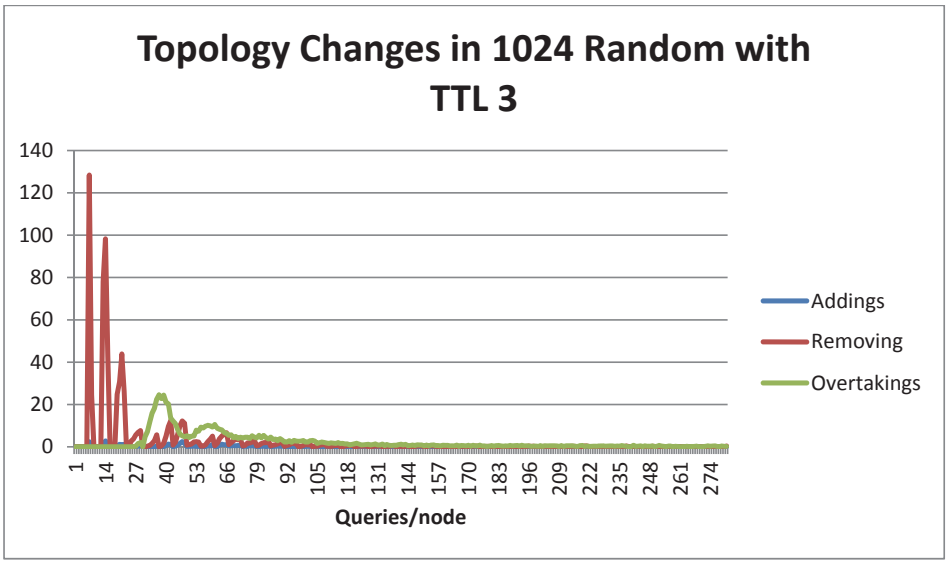

FIGURE 29 The amount of topology changes during the simulation of topology management algorithms in random network of 1024 nodes with TTL 3.

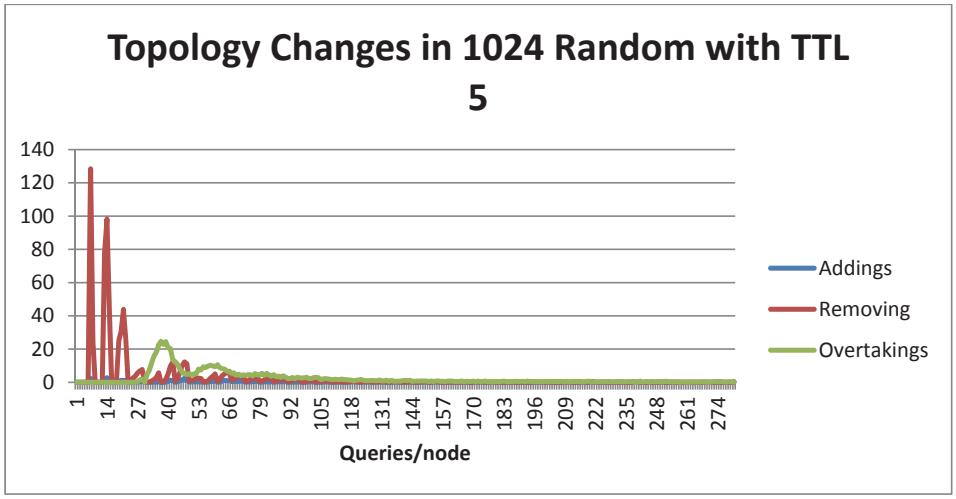

FIGURE 30 The amount of topology changes during the simulation of topology management algorithms in random network of 1024 nodes with TTL 5.

TABLE 36 The average queries, replies and efficiency values of torus network of 256 nodes at equilibrium.

	Torus 256, TTL 3				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	3,983	1,829	1,034	2,079	0,260
2	14,912	10,399	1,646	2,736	0,110
3	42,610	50,802	2,861	5,575	0,067

TABLE 37 The average queries, replies and efficiency values of random network of 256 nodes at equilibrium.

	Random 256, TTL 3				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	3,479	6,145	0,738	1,441	0,213
2	14,522	61,883	1,965	5,845	0,136
3	53,153	401,620	5,110	25,005	0,097

TABLE 38 The average queries, replies and efficiency values of torus network of 1024 nodes with TTL 3 at equilibrium.

	Torus 1024, TTL 3				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	3,9930	3,0967	0,3269	0,6664	0,0819
2	21,5672	29,4514	0,6877	1,3120	0,0319
3	87,0593	188,2515	1,6442	3,8631	0,0189

TABLE 39 The average queries, replies and efficiency values of torus network of 1024 nodes with TTL 5 at equilibrium.

	Torus 1024, TTL 5				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	3,9924	1,5330	0,1618	0,4295	0,0405
2	14,1151	9,7659	0,3245	0,7262	0,0230
3	36,4153	49,5177	0,6207	1,4156	0,0170
4	66,3347	135,5252	0,9933	2,5985	0,0150
5	100,5774	257,5062	1,3732	4,0019	0,0137

TABLE 40 The average queries, replies and efficiency values of random network of 1024 nodes with TTL 3 at equilibrium.

	Random 1024, TTL 3				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	4,0773	4,3302	0,1880	0,4882	0,0461
2	24,2453	27,2560	0,8071	1,3810	0,0333
3	133,4158	171,7463	3,4878	4,8733	0,0261

TABLE 41 The average queries, replies and efficiency values of random network of 1024 nodes with TTL 5 at equilibrium.

	Random 1024, TTL 5				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	2,7638	8,9277	0,1348	0,4063	0,0491
2	9,8103	61,4897	0,3140	1,9357	0,0321
3	28,3704	363,6612	0,8568	10,1725	0,0304
4	80,3731	1658,0642	2,2453	34,6015	0,0288
5	204,6774	4940,8829	4,7891	49,3008	0,0255

TABLE 42 The average queries, replies and efficiency values of reconstructed torus network of 256 nodes with DBFS.

	Torus 256, TTL 3				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	1,000	0,016	0,469	1,172	0,469
2	5,141	11,159	1,138	2,579	0,222
3	20,419	56,964	2,254	6,617	0,111

TABLE 43 The average queries, replies and efficiency values of reconstructed random network of 256 nodes with DBFS.

	Random 256, TTL 3				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	1,000	0,023	0,321	1,302	0,321
2	6,227	21,403	1,137	2,772	0,184
3	26,927	181,826	3,533	18,314	0,132

TABLE 44 The average queries, replies and efficiency values of reconstructed torus network of 1024 nodes with DBFS.

	Torus 1024, TTL 3				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	1,0000	0,0104	0,0996	0,3486	0,0996
2	6,8192	13,8961	0,3273	0,8763	0,0481
3	35,2546	93,7640	0,9561	2,4925	0,0271

TABLE 45 The average queries, replies and efficiency values of reconstructed torus network of 1024 nodes with DBFS.

	Torus 1024, TTL 5				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	1,0000	0,0090	0,0460	0,2236	0,0461
2	4,2264	5,6105	0,1410	0,4571	0,0333
3	16,1698	31,1055	0,4171	1,0166	0,0258
4	47,0932	111,8202	0,8818	2,3570	0,0187
5	94,3834	267,2742	1,4283	4,2738	0,0151

TABLE 46 The average queries, replies and efficiency values of reconstructed random network of 1024 nodes with DBFS.

	Random 1024, TTL 3				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	1,0000	0,0070	0,0589	0,2486	0,0590
2	7,8678	9,9336	0,3017	0,7402	0,0383
3	49,0531	86,8017	1,5445	2,9455	0,0315

TABLE 47 The average queries, replies and efficiency values of reconstructed random network of 1024 nodes with DBFS.

	Random 1024, TTL 5				
Hops	**Q**	**S_Q**	**H**	**S_H**	**E**
1	1,0000	0,0061	0,0545	0,2754	0,0546
2	5,4226	13,7511	0,1844	0,6819	0,0340
3	17,7369	102,7815	0,5608	3,2139	0,0316
4	57,8770	648,3989	1,7263	16,6835	0,0301
5	176,6154	2909,8513	4,5178	45,0982	0,0266

ORIGINAL PAPERS

PI

RESOURCE DISCOVERY IN P2P NETWORKS USING EVOLUTIONARY NEURAL NETWORKS

by

Mikko Vapa, Niko Kotilainen, Annemari Auvinen, Heikki Kainulainen & Jarkko Vuori 2004

IEEE International Conference on Advances in Intelligent Systems - Theory and Applications

Resource Discovery in P2P Networks Using Evolutionary Neural Networks

Mikko A. VAPA, Niko P. KOTILAINEN, Annemari K. AUVINEN,
Heikki M. KAINULAINEN, and Jarkko T. VUORI

***Abstract*-- Resource discovery is an essential problem in peer-to-peer networks since there is no centralized index in which to look for information about resources. One solution for the problem is to use a search algorithm that locates resources based on the local knowledge about the network. Traditionally, the search algorithms have been based on few simple rules, which often reduces the performance from optimal. In this paper, we describe the results of a process where evolutionary neural networks are used for finding an efficient search algorithm from a class of local search algorithms. The initial test results indicate that an evolutionary optimization process can produce search algorithm candidates that are competent compared to the breadth-first search algorithm (BFS) used in Gnutella peer-to-peer network.**

***Index Terms*-- resource discovery, peer-to-peer networks, multi-layer perceptrons, genetic algorithms.**

I. Introduction

In the *resource discovery problem*, any node can possess resources and query these resources from other nodes in the network. The problem consists of graph with nodes, links and resources. Resources are identified by unique IDs and nodes may contain any number of resources. One node knows only the resources it is currently hosting. Any node in the graph can start a query, which means that some of the links are traversed based on a local decision in the graph. Whenever the query reaches the node with the queried ID, the node replies. The goal is to locate a predetermined amount of resource instances with a given ID using as few query packets as possible.

One possible solution for the resource discovery problem is the breadth-first search algorithm (BFS) [1]. In BFS a node that starts a query passes the query to all its neighbors. When the neighbors receive the query, they pass it further to all their neighbors except the one from which the query was received. Nodes cache the messages that they have received and if the query has already been received from other neighbor then query is dropped. Time-to-Live (TTL) value is used to limit the number of hops the query can take by reducing TTL value each time a query is received. When TTL decreases to zero the query is dropped. The BFS algorithm ensures that if a resource is located in the network it can be found from the network if TTL is high enough. The downside of the algorithm, however, is that it uses many query packets to find the needed resources. Thus, we propose an alternative algorithm that is more efficient in face of used query packets and evaluate it using peer-to-peer scenario with power-law distributed topology [2].

The rest of this paper is organized as follows. The next section presents the references to related work done in P2P resource discovery. Section III describes the NeuroSearch algorithm as a solution for the resource discovery problem. Section IV describes the optimization process and Section V the test case used in the study. Section VI analyzes the simulation results and in Section VII the paper is concluded.

II. Related Work

Much research has been done regarding the resource discovery problem. Adamic et al. [3] and Kim et al. [4] propose a search strategy that utilizes the topological properties of a power-law network. The search strategy first proceeds towards highest-degree node, e.g. the node that has the highest number of neighbors, and then gradually moves to lower degree ones. The algorithm locates resources efficiently if they can be found from the core of the network, but the performance decreases when the central nodes are revisited in search for lower degree nodes.

Lv et al. [5] evaluate BFS, expanding ring and random walk search mechanisms with varying topologies, including random graphs [2], power-law graphs and a snapshot of the Gnutella network obtained in October 2000. These researchers find that BFS is not scalable and in particular on Gnutella and power-law graphs the effects of flooding are disastrous: the number of messages increases drastically when TTL is increased. Expanding ring, where TTL is extended gradually for BFS, is the first aid to the problem. However, because it forwards duplicate messages to the nodes that the query has already reached, a better solution to the problem using random walkers is proposed by the researchers. A search initiates multiple walkers and forwards them based on a random selection of a neighbor. In addition to the TTL as a termination condition for the walkers, Lv et al. use checking, where the random walkers periodically check from the query originator whether the

Manuscript received September 2, 2004. This work was supported in part by the Graduate School in Electronics, Telecommunications and Automation (GETA) and Innovations in Business, Communication and Technology (InBCT) –project of Agora Center.

M. A. Vapa, A. K. Auvinen, and J. T. Vuori are with Department of Mathematical Information Technology, University of Jyväskylä, Finland (e-mail: firstname.lastname@jyu.fi).

N. P. Kotilainen is with Agora Center, University of Jyväskylä, Finland (e-mail: niko.kotilainen@jyu.fi).

H. M. Kainulainen is with WTS Networks, Jyväskylä, Finland (e-mail: heikki.kainulainen@wts.fi)

walker should be terminated or not. While random walkers increase the number of hops and thus latency, they decrease the total traffic because the search proceeds in a depth-first manner.

Kalogeraki et al. [6] consider two search algorithms for the resource discovery problem. The Modified Random BFS Search behaves like BFS, but the neighbors select only a random subset of neighbors for forwarding the query. This reduces traffic, but adjusting the correct size of the subset for various networks may be difficult. The researchers' work uses a random graph in which all the nodes have approximately similar degrees. Thus the performance of the algorithm in power-law graphs cannot be directly determined from the results. In another algorithm they present, called Intelligent Search Mechanism, the nodes keep track of recent query results provided by their neighbors. When a new query arrives, the neighbors are sorted based on the similarity of the query to earlier replies from the neighbor. Because the nodes keep track of the earlier queries, the performance of the algorithm improves as the network evolves.

Yang and Garcia-Molina [7] experimented with many types of directed search strategies based on various heuristics. These heuristics include the number of results returned, shortest average time to satisfaction, smallest average number of hops of received results, the highest number of results returned, shortest message queue, shortest latency and highest degree. Their work suggests that, to minimize the time to satisfaction measure, the best strategy is to pass the query to the neighbor that has had the shortest average time to satisfaction for last ten queries. Also, when considering the bandwidth use, the most reliable measure is the smallest average number of hops of received results for last ten queries. The heuristics used in the study are based on history data collected locally in each node.

Similar use of history data is found from the work by Tsoumakos and Roussopoulos [8]. In their proposal, called Adaptive Probabilistic Search algorithm, neighbors keep track of the success rates of earlier queries and forward random walkers probabilistically, based on the earlier success rate. The algorithm is able to adapt to different query patterns and, therefore, performs better than random walkers.

There are certain limitations in all the approaches described above. First, each of these algorithms uses some control parameters (for example time-to-live, the number of walkers or the proportion of neighbors to forward the query) that can be used to tune the algorithm. For a search algorithm, the number

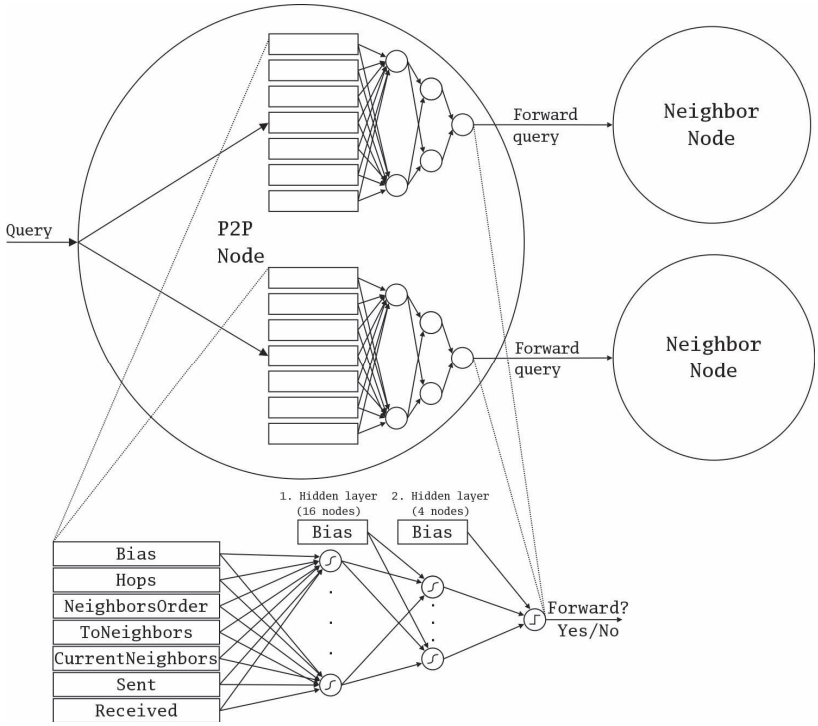

Fig. 1: Processing of NeuroSearch resource query and the NeuroSearch neural network

of control parameters should be kept to a minimal to allow zero configurability when applied to a real environment. Second, while some of these approaches have mechanisms to adapt to the environment, they do not utilize the entire potential of the environment because they rely only on one strategy (for example the similarity of the query and earlier replies, shortest average time to satisfaction for last 10 queries or the success rate of earlier queries). In general, only one strategy cannot be efficient in all scenarios and therefore an efficient algorithm should be able to utilize many strategies at the same time.

To overcome these limitations a neural network based resource discovery algorithm called NeuroSearch was designed. NeuroSearch learns by itself the correct behavior in given network conditions and uses many combinations of strategies to locate resources. To authors' knowledge this is the first time when neural networks are being applied to resource discovery problem.

III. NeuroSearch Resource Discovery Algorithm

The proposed algorithm, called as *NeuroSearch*, makes decision to whom of the node's neighbors the resource request message is forwarded based on the output neuron of three-layer perceptron neural network. The algorithm is located inside a peer node as shown in Fig. 1 and is the same for all peers in the network. NeuroSearch can be represented as a function $O: I \rightarrow \{0,1\}$, where $I \in [0,1]^7$ is a 7-dimensional input vector representing the state of a resource discovery query. The output of O defines whether in a given state query should be dropped $O = 0$ or forwarded to a peer $O = 1$ and is evaluated for each neighbor peer separately.

When a resource request arrives to the algorithm it goes through all the node's neighbors (denoted as receivers) one by one with the neural network. The input parameters for the neural network are:

- *Bias* is the bias term and has value 1.
- *Hops* is the number of hops the message has travelled.
- *NeighborsOrder* indicates in which rank this receiver is in terms of number of neighbors compared to other neighbors. The connection with highest rank has the value of 0, second rank has the value of 1 and so on.
- *ToNeighbors* is the number of the receiver's neighbors.
- *CurrentNeighbors* is the number of node's neighbors.
- *Sent* has value 1 if the message has already been forwarded to the receiver. Otherwise it has value of 0.
- *Received* has value 1 if the message has been received earlier, else it has value of 0.

Hops and *NeighborsOrder* are scaled with the function $f(x) = \frac{1}{x+1}$ and *Neighbors* and *CurrentNeighbors* with $f(x) = \frac{1}{x}$ before giving them to the neural network. Scaling is performed to ensure that all the inputs are between 0 and 1.

There are two hidden layers in the network. In the first hidden layer there are 15 nodes + bias and in the second hidden layer 3 nodes + bias. Tanh is used as an activation function in the hidden layers: $t(a) = \frac{2}{1+e^{-2a}} - 1$, where a is the weighted sum of inputs to a neuron. Activation function in the output node is the threshold function $s(a) = \begin{cases} 0, a < 0 \\ 1, a \geq 0 \end{cases}$.

Combining all together, the output O of the neural network can be calculated with the following formula:

$$O = s(1 + \sum_{k=1}^{4} w_{3k} t(1 + \sum_{j=1}^{16} w_{2j} t(\sum_{i=1}^{7} w_{1i} f(I_i)))),$$

where I_i is the value of input parameter i and w_{xy} the neural network weights on layer x in position y.

Whenever the query locates a queried resource a reply message is sent back to the neighbor, which forwarded the request to the node. When all the nodes in the query path have forwarded the reply message backward, it is finally received by the query initiator.

IV. Neural Network Optimization

The weights w_{xy} are unknown and therefore they need to be adjusted to appropriate values. For doing this we use methods of evolutionary computing [9]. The decision, which neural networks are better than the others is done by counting the query packets traversed in the test network and found resources. The fitness for the neural network is defined in two parts. Each query j is scored for the neural network h and the fitness is calculated by summing up all the scores after n queries: $fitness_h = \sum_{j=1}^{n} score_j$. The *score* is defined with the following conditions:

1. If *packets* > 300 then *score* = 0
2. If *foundResources* = 0 then *score* = $1 - \frac{1}{packets + 1}$
3. If *foundResources* < *availableResources* / 2 and *foundResources* > *0* then *score* = 50 × *foundResources* – *packets*
4. If *foundResources* ≥ *availableResources* / 2 then *score* = 50 × *availableResources* / 2 – *packets*

In the equations *availableResources* is the maximum number of resource intances that can be located in the query, *foundResources* is the number of resource instances that the neural network was able to locate for the query, and *packets* is the number of query packets the neural network used for the query. The constant value 300 was set as criterion for determining when the neural network is considered to forward the query indefinitely and the query can be stopped. Another constant value, 50, was selected to be large enough to guide the training process towards neural networks that locate more resources than other neural networks. Now a neural network could spend 49 query packets more in a query to locate one additional resource compared to other neural network, which located one resource less.

The first rule ascertains that an algorithm that eventually

stops is always better than algorithm that does not. The goal of finding half of the available resource instances was set to demonstrate the algorithm's ability to balance on a predetermined quality of service level and not just on locating all resource instances or one resource instance. The second rule makes sure that if none of the resources are found then the neural network should increase the number of query packets sent to the network. The third rule states that if the number of found resources is not enough then the neural network develops only by locating more resources. Finally the last rule ensures that when half of the available resource instances are found from the network the fitness grows if neural network uses fewer query packets.

The optimization process had an initial population of 30 neural networks whose weights were randomly defined from interval [-0.2, 0.2]. Next, every neural network was tested in the peer-to-peer simulation environment and fitness value calculated. When all neural networks had been tested 15 best were chosen for mutation and used to breed the new generation of neural networks. As a result, 30 neural networks were available for testing the new generation.

Mutation was based on the Gaussian random variation and used weighted mutation parameter to improve the adaptability of the evolutionary search. The random variation function was similar to the one used by Fogel and Chellapilla in their research [10] and is given as:

$$\sigma_i'(j) = \sigma_i(j)\exp(\tau N_j(0,1)),\ j = 1,\ldots,N_w,$$
$$w_i'(j) = w_i(j) + \sigma_i'(j)N_j(0,1),\ j = 1,\ldots,N_w,$$

where $N_w =$ is the total number of weights and bias terms in the neural network, $\tau = \frac{1}{\sqrt{2\sqrt{N_w}}}$, $N_j(0,1)$ is a standard Gaussian random variable resampled for every j, σ is the self-adaptive parameter vector for defining the step size for finding the new weight, $w_i'(j)$ is the new weight value and index $1 \le i \le 185$ denotes the number of neuron enumerated over all layers.

V. Simulation Environment

As a peer-to-peer simulation environment, we used Peer-to-Peer Realm (P2PRealm) network simulator [11] that we have developed. The simulator can be used to simulate the behavior of a static peer-to-peer network and to train neural networks using Gaussian random variation. P2PRealm has been implemented using Java.

In the test case we used power-law graphs generated using the Barabási-Albert model [12]. A power-law network's neighbor distribution follows the power-curve $P(k) = \frac{1}{k^{\gamma}}$, where $\gamma = 3$ for Barabási-Albert graph. Therefore in power-law networks there exist few hubs in the network that have many neighbors as well as many nodes that have only few neighbors. A power-law graph was selected because existing P2P networks have shown to express power-law dependencies [13]. The graphs tested contained 100 nodes with the highest degree node having 25 neighbors. Small network size was selected to allow visualisation of query paths in the network. Dynamic changes e.g., node failures were not taken into account to simplify the analysis. However, the approach can be applied in dynamic scenarios also as shown in [14].

The test case data was divided into three distinct data sets as described in [15]: a training set, a generalization set and a validation set. Training set is used for training the neural network. Generalization set is used to measure how well the trained neural network performs with a new data set indicating neural network's ability to generalize. When performance starts to decrease in generalization set the training can be stopped, because the neural network adapts only to the training set if training process is continued. Validation set is used as an objective measure to verify how well the algorithm performs with arbitrarily chosen new data set and ensures that the true generalization ability of the neural network is being measured.

The training set contained two power-law topologies with both being queried n = 50 times per generation for each neural network. Two topologies were used to have neural networks adapt to a wider range of situations than one topology would have provided. The generalization set consisted of two power-law topologies with 50 queries. When the performance started to decrease in the generalization set the neural network having highest fitness was selected and, as a validation set, one topology with 100 queries was used to produce the final simulation results.

For each topology, resource instances were allocated based on the number of neighbors each node has. There were 25 different resources in the test case and the number of different resources in a node was the same as the number of neighbors the node had. This means that the largest hub had one instance of all resources and the lower degree nodes only some of these, randomly chosen from uniform distribution. The querying nodes and queried resources were selected also randomly from a uniform distribution for each query.

As stopping criteria for the optimization process, 100,000 generations were set. This seemed to take approximately two

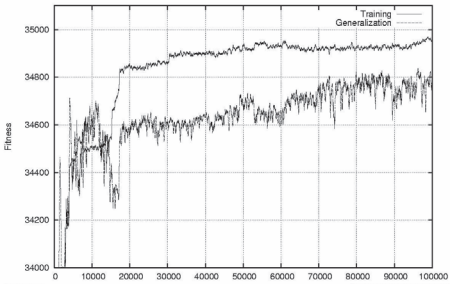

Fig. 2: Evolution of the best neural networks in each generation for training and generalization sets

weeks on our desktop PC equipped with an AMD Athlon XP 1800 processor. The evolution of the best neural network in each generation is shown in Fig. 2.

VI. Simulation Results

To evaluate the difference between BFS and NeuroSearch, we selected the best algorithm at the 85,736th generation and calculated the number of packets used and found resources for 100 different queries using validation set. The 85,736th generation was selected because between the 80,000 and 90,000 generations the neural networks had achieved steadily good results and, in particular, in the 85,736th generation, neural network had the best fitness. The results are presented in Fig. 3 and Fig. 4.

The results of Fig. 3 show that the performance of NeuroSearch regarding the number of packets is nearer to BFS with a time-to-live value 2 (BFS-2), rather than BFS with a time-to-live value 3 (BFS-3). In average NeuroSearch consumes 47.2 packets per query whereas BFS-2 consumes 30.0 and BFS-3 122.0 packets. The reason why there is some variation in the number of packets for successive BFS queries is that the number of delivered packets depends on which node is querying. If the query starts from a central node (nodes 0-10), it will produce more packets than the same query started from an edge node (nodes 90-99) because the edge query has fewer connections where BFS can spread. In case of NeuroSearch, the performance is stable and does not depend on which node is querying.

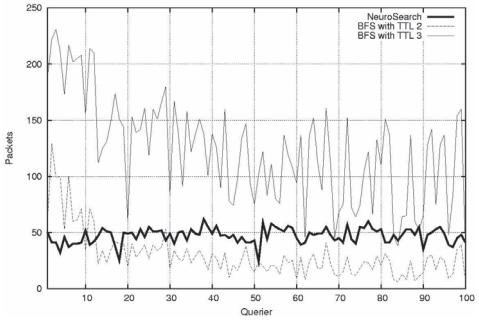

Fig. 3: Number of packets used by the algorithms

Fig. 4 shows how many resources the algorithms were able to locate. NeuroSearch's performance in terms of located resources is quite similar to BFS-2 at central nodes, but better in the edge nodes. Compared to BFS-3 NeuroSearch's performance is constantly lower, reaching the same performance level only at some edge nodes. The reason why NeuroSearch is satisfied with this level of performance is that it has already reached the goal of finding half of the available resources as defined in the fitness function and locating more resources is not needed.

By calculating the ratio between the located resources and used query packets we can determine the efficiency of the algorithms. These values are shown in Table I. The results show that NeuroSearch's efficiency is at the same level as BFS-2's locating a new resource every fifth packet. BFS-3 locates a new resource approximately every ninth packet. Efficiency is easier to keep high when locating only few resources because usually those can be found from the central nodes alone. When the number of needed resources increases, query has to spread more to the edges to locate the additional resources. Therefore the efficiency of BFS-3 decreases significantly. BFS-2 and NeuroSearch achieve near similar efficiency indicating that NeuroSearch is able to sustain a good efficiency even though it needs to locate more resources than BFS-2.

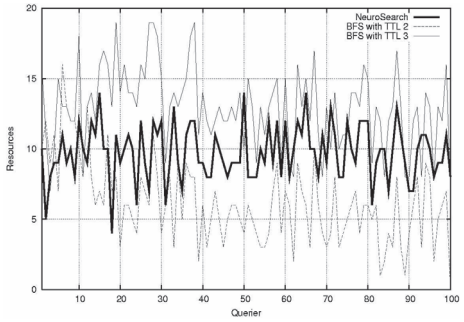

Fig. 4: Number of resources found by the algorithms

TABLE I
Efficiency of the Algorithms

Algorithm	Packets	Resources	Efficiency
BFS-2	3000	619	0.2063
BFS-3	12202	1295	0.1061
NeuroSearch	4719	975	0.2066

For each query, NeuroSearch locates approximately half of the resources or more, which can be seen in Fig. 5. There are six queries in which NeuroSearch misses the target to locate half of the resources. This variation results from the difference

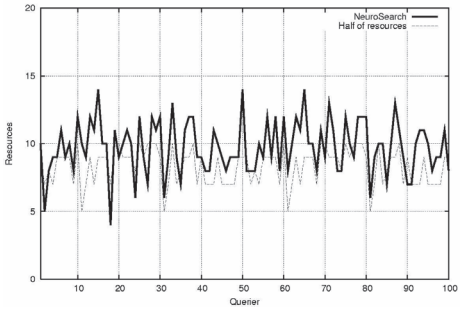

Fig. 5: Difference of located resources to half of resources

between the training set and the validation set. Nonetheless, the results indicate that the optimization process has found an algorithm that is able to locate nearly half of the resources from the network with high probability.

We analyzed the behavior of the best-evolved neural network by tracking the path used by the queries. NeuroSearch seems to prefer central nodes early in the query and uses multiple paths for doing this. After reaching central nodes or one hop later the spreading is stopped. The maximum number of hops is 5. As verification for this the behavior of a typical NeuroSearch query started from an edge node is illustrated in Fig. 5. In the figure the query travels through the connections denoted with a black line starting from node 99 with question mark (?). Nodes marked with an exclamation mark (!) contain the queried resource. In total the query uses 49 packets and locates 11 resources. Six connections are traversed from both directions, which is not shown in the figure.

VII. Conclusion

In this paper, a new resource discovery algorithm has been proposed. NeuroSearch algorithm takes into account the special characteristics of its environment and can be adjusted to different kind of P2P networks. The algorithm's performance is also stable and competitive compared to the BFS algorithm.

While NeuroSearch performs well compared to BFS it is by no means yet designed to be optimal. For example, NeuroSearch does not yet include history-based inputs even though they would significantly improve the performance. Therefore, the results obtained in [3]-[8] will be considered in forthcoming research on NeuroSearch. There are also other directions that were left out of this research. First, we are studying what improvements to the performance would be gained by varying the neural network's internal structure. Second, we are aiming to find out what are the scalability factors of NeuroSearch when the network size grows, and third we are developing an optimal resource discovery algorithm using global knowledge to be able to measure the best efficiency a resource discovery algorithm can achieve. Also, we are working on a solution to speed up the optimization process by parallelizing the evolutionary algorithm using distributed computing. This helps us to more accurately determine the performance maximum of NeuroSearch.

Acknowledgment

The authors would like to thank the co-designers of NeuroSearch Joni Töyrylä, Yevgeniy Ivanchenko, Matthieu Weber and Hermanni Hyytiälä. Also we thank Tommi Kärkkäinen for giving useful hints how to develop the algorithm further and Barbara Crawford for proofreading the article.

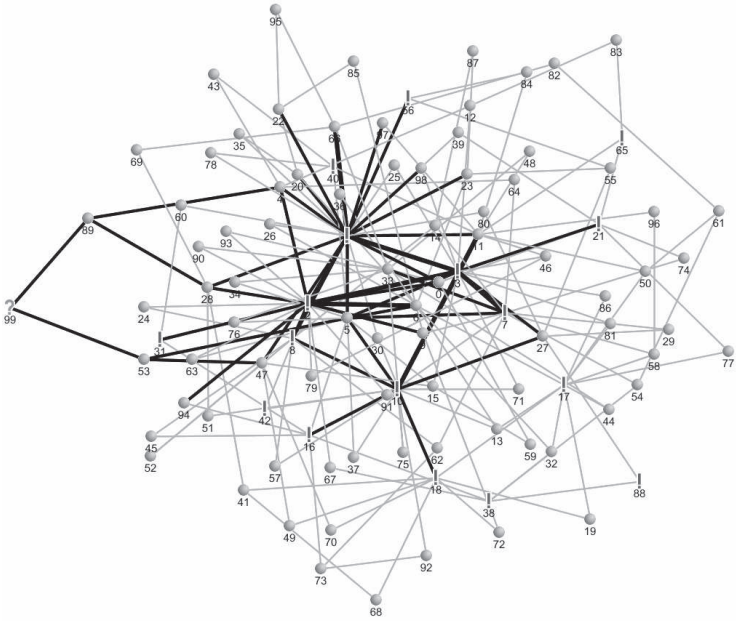

Fig. 5: Typical NeuroSearch resource query

REFERENCES

[1] N. A. Lynch, *Distributed Algorithms*, Morgan Kauffmann Publishers, 1996.

[2] A. Barabási, *Linked*, Perseus Publishing, 2002.

[3] L. A. Adamic, R. M. Lukose, and B. A. Huberman, "Local Search in Unstructured Networks", in *Handbook of Graphs and Networks: From the Genome to the Internet*, Wiley-VCH, 2003, pp. 295-317.

[4] B. J. Kim, C. N. Yoon, S. K. Han, and H. Jeong, "Path finding strategies in scale-free networks", *Physical Review E 65*, 2002.

[5] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks", in *Proceedings of the 16th International Conference on Supercomputing*, ACM Press, 2002, pp. 84-95.

[6] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yatzi, "A Local Search Mechanism for Peer-to-Peer Networks", in *Proceedings of the 11th International Conference on Information and Knowledge Management*, ACM Press, 2002, pp. 300-307.

[7] B. Yang and H. Garcia-Molina, "Improving search in peer-to-peer networks," in *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS'02)*, 2002.

[8] D. Tsoumakos and N. Roussopoulos, "Adaptive Probabilistic Search for Peer-to-Peer Networks", in *Proceedings of the Third IEEE International Conference on P2P Computing (P2P2003)*, IEEE Press, 2003, pp. 102-109.

[9] K. Miettinen, M. Mäkelä, and P. Neittaanmäki and J. Périaux (eds.), *Evolutionary algorithms in engineering and computer science*, John Wiley & Sons, 1999.

[10] K. Chellapilla and D. Fogel, "Evolving neural networks to play checkers without relying on expert knowledge", *IEEE Trans. on Neural Networks*, 10 (6), pp. 1382-1391, 1999.

[11] J. Töyrylä, *Building NeuroSearch – Intelligent Evolutionary Search Algorithm For Peer-to-Peer Environment*, Master's Thesis, University of Jyväskylä, 2004.

[12] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks", *Science 286* (1999) 509-512.

[13] M. A. Jovanovic, F. S. Annexstein, and K. A. Berman, *Scalability Issues in Large Peer-to-Peer Networks – A Case Study of Gnutella*, Technical report, University of Cincinnati, 2001.

[14] Y. Ivanchenko, *Adaptation of Neural Nets For Resource Discovery Problem in Dynamic And Distributed P2P Environment*, Master's Thesis, University of Jyväskylä, 2004.

[15] A. P. Engelbrecht, *Computational Intelligence: An Introduction*, John Wiley & Sons Ltd, 2002.

PII

CHEDAR: PEER-TO-PEER MIDDLEWARE

by

Annemari Auvinen, Mikko Vapa, Matthieu Weber, Niko Kotilainen & Jarkko Vuori
2006

IEEE 20th International Parallel and Distributed Processing Symposium

Chedar: Peer-to-Peer Middleware

Annemari Auvinen, Mikko Vapa, Matthieu Weber, Niko Kotilainen and Jarkko Vuori

Department of Mathematical Information Technology
University of Jyväskylä
P.O.Box 35 (Agora), 40014 University of Jyväskylä, Finland
{annauvi, mikvapa, mweber, npkotila, jarkko.vuori}@jyu.fi

Abstract

In this paper we present a new peer-to-peer (P2P) middleware called CHEap Distributed ARchitecture (Chedar). Chedar is totally decentralized and can be used as a basis for P2P applications. Chedar tries to continuously optimize its overlay network topology for maximum performance. Currently Chedar combines four different topology management algorithms and provides functionality to monitor how the peer-to-peer network is self-organizing. It also contains basic search algorithms for P2P resource discovery. Chedar has been used for building a data fusion prototype and a P2PDisCo distributed computing application, which provides an interface for distributing the computation of Java applications. To allow Chedar to be used in mobile devices, the Mobile Chedar middleware has also been developed.

1 Introduction

Peer-to-peer technologies have received a lot of publicity lately mainly because of Napster and other peer-to-peer systems mostly developed for distributing music and movies in the Internet. A peer-to-peer network is also well suited for sharing other resources than files, for example CPU time and storage space. Every node in a P2P network may provide resources to other nodes and consume resources the other nodes are providing, i.e. a node may serve both as a server and a client. Therefore there is no need for a central server which might become the bottleneck of the network or which failure will paralyze the whole network. Also the data traffic is more evenly distributed in the P2P network than in the centralized networks where central node's data traffic might be very large.

This work was supported in part by the Agora Center InBCT project.

Gnutella [14], published in 2000, is a decentralized pure peer-to-peer protocol [15]. Gnutella servents use TCP connections for communication and the Breadth First Search (BFS) algorithm for searching resources. When a node wants to join the Gnutella network it must first find one node in the network to which to establish a connection. That node can be found for example from a web page containing a list of nodes. In Gnutella, a node usually has some pre-defined amount of connections. To find new neighbors a Gnutella node uses ping messages. A ping message is broadcasted in the network and nodes reply to it with pong messages. The node stores information about the active connections it has so it can try to connect to those when joining the network after disconnecting.

In Gnutella the search queries are broadcasted in the network. The querier sends the query message to its neighbors, which forward the query to their neighbors except the node from where the query arrived. The amount of hops the query travels can be limited by setting a time-to-live (TTL) value. Every time a node forwards the query, it decrements the value of the TTL by one. When the value of the TTL becomes zero the node drops the message. If a node owns the queried resource it sends a reply to the querier using the same route as the query came from.

Chedar differs from Gnutella in some ways. Chedar is a middleware, i.e. it offers an API for P2P applications. It contains new kinds of topology management algorithms by which the overlay topology on top of the physical network is self-organized. Those algorithms use only the local information the nodes have on their neighbors. The purpose of the algorithms is to create a network which is scalable and fault-tolerant. Chedar also has four other search algorithms implemented, in addition to the BFS that Gnutella uses. Chedar also

1-4244-0054-6/06/$20.00

guarantees that a resource reply message can be forwarded to the querier if it still exists in the network. Chedar uses an XML-based, structured resource description and the XPath language for matching the query keywords with its resources.

This paper is organized as follows. We describe Chedar in Section 2 and its structure in Section 3. The messages Chedar uses are described in Section 4. The algorithms used for managing the topology of the Chedar network are presented in Section 5 and the paper is concluded in Section 6.

2 Chedar

We have developed the Chedar system for resource sharing and distribution. Chedar is a pure peer-to-peer middleware implemented using the Java programming language. Any application which uses the API and implements the callback functions required by the API may use Chedar and run on it. Peers, i.e. nodes in the network, communicate directly with each other using TCP connections. Chedar is developed to work in a dynamic environment where the nodes may join or leave the network whenever they want without causing significant problems to the applications running on top of Chedar. Because there are no central points in the network, Chedar is fault-tolerant and scalable. In case of link failures the topology management algorithms ensure that new peers will be contacted and the network stays connected.

Chedar can be used to distribute different kinds of resources to other nodes in the Chedar network. Distributed resources can be for example files, CPU time or storage space. Every node stores information about the resources it provides in XML format. When the node receives a query about some resource it checks by using the XPath expression whether it owns the resource.

In Chedar a neighbor's goodness is defined based on the resource replies the node receives from the neighbor to the requests the node has sent. The more the neighbor offers requested resources, the more important it is for the node. The amount of replies the neighbor has relayed to the node also affects it's goodness value. The overlay topology and the traffic in the Chedar network is managed by the Overtaking and Overload Estimation algorithms which use neighbor's goodness value as a measure for selecting which of the connections should be dropped and where to connect. Also Chedar always tries to route the resource reply to the initiator of the request. In Chedar it is possible to use multiple search algorithms, unlike in Gnutella which only uses the broadcasting search algorithm.

In our research project [4] Chedar has been used for distributed computing [9] and data fusion [13] and extended also to mobile devices [10]. Peer-to-Peer Distributed Computing (P2PDisCo) software was built on top of Chedar to speed up the training of neural networks with evolutionary computing. In the Decentralized Data Fusion System (DDFS) application each sensor node is one Chedar node. DDFS can be used to track targets based on the sensor measurement of their coordinates. Mobile Chedar is an extension to the Chedar peer-to-peer network for mobile peer-to-peer applications and has been implemented using Java 2 Micro Edition. We have also developed P2PStudio [8] monitoring application for the Chedar network to study the performance of search algorithms and the self-organizing behavior of the topology management algorithms.

3 Structure of Chedar

Chedar consists of five main components which are Connections, ConnectionManager, PropagationEngine, TopologyManager and ChedarClient. These components are illustrated in Figure 1.

3.1 Connections

The Connections include local information used by the topology management algorithms about the node's neighbors. Each neighbor is one connection object. Chedar keeps information about active connections and history data about the earlier connections in XML trees. Searches can be made to the XML tree using an XPath expression. History data also contains information about the nodes which the peer has found out from its neighbors. The nodes save the IP addresses and the TCP ports of the neighbors, the types of resources those provide and hit values per provided resource types. Hit values are described later in the next paragraph. Chedar saves also the time when the connection last replied, when the connection request has been sent to the connection and whether the request

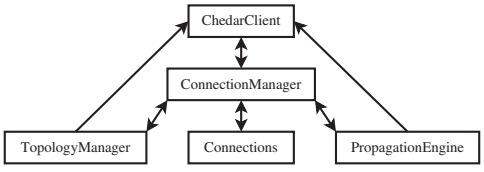

Figure 1. Main components of Chedar.

succeeded or not. Relayed hits and the number of the connection's neighbors is stored about active connections.

Every connection has three types of hit values in Chedar. First one, called *hit value*, is increased by one every time the node gets a resource reply from its neighbor. Second one is called *actual hits* and is increased when the node uses a resource the neighbor provides. *Relayed hits* values of the connection include the neighbor's neighbor nodes and the amount of the reply messages those have sent to the node through the neighbor.

3.2 ConnectionManager

The ConnectionManager manages the active connections and the history data by adding connections or removing connections according to the TopologyManager's requests. The ConnectionManager keeps a cache about the information of forwarded messages and handles all arriving messages and passes them to the classes which have informed wanting that type of messages. The ConnectionManager has also a traffic meter which measures the size of the resource messages going through the node in a given time period. The traffic meter is used by the Overload Estimation Algorithm.

3.3 TopologyManager

The TopologyManager selects the connections to establish or remove and the nodes to overtake by using the algorithms described in the Section 5. It handles *ConnectionRequest* and *ConnectionReply* messages, *NeighborListRequest* and *NeighborListReply* messages and *ServiceListRequest* and *ServiceListReply* messages which are described in the Section 4.

3.4 PropagationEngine

The PropagationEngine handles the resource messages. The application running on top of Chedar can select any search algorithm that is implemented in Chedar. Currently five different resource discovery algorithms have been implemented: BFS [12], Random Walk [11], Highest Degree Search [1, 6] and NeuroSearch [16]. Some of the algorithms are reviewed in [16]. The PropagationEngine passes the received resource request or reply message to the search algorithm specified in the message. The algorithm makes the decision where to forward the message and creates a reply message when needed. The algorithm returns to the PropagationEngine the forwarded message and a list of connections where to forward the message.

3.5 ChedarClient

The ChedarClient works as an API for Chedar. The ChedarClient provides methods for setting and getting the values of the parameters used in the algorithms. Resources can be set with the ChedarClient and it propagates events about received and sent messages and overtakings to the application. The ChedarClient also provides methods for establishing a connection to a certain node and for creating a resource request. Table 1 shows the methods that are accessible in the ChedarClient for the applications running on top of Chedar.

By implementing Chedar's monitoring interface, the iMonitor, the application may get the following events: *resourceQuerySentEvent, messageForwardedEvent, messageDiscardedEvent, resourceReplySentEvent, resourceReplyReceivedEvent, overtakingEvent, connectionRequestedEvent* and *connectionStatusEvent*.

4 Messages

There are three types of messages used for the topology management in Chedar. When a node wants to establish a new connection to another node it sends a *ConnectionRequest* message to it. The requested node sends back a *ConnectionReply* message which includes the information whether it accepts the request or not.

With a *NeighborListRequest* message the node can query a connection's neighbors, i.e neighbor's neighbors. The sender puts its own neighbors' IDs into the message. The node replies to the request by sending back a *NeighborListReply* message which includes the neighbors' IDs. The nodes save the neighbors' IDs to the history data.

A node can query the resource types its neighbor provides, e.g. file or computing time, by sending to a connection a *ServiceListRequest* message which is replied by a *ServiceListReply* message. The request message includes the resource types the sender provides and the reply message includes the replier's resource types.

Resources are searched with a *ResourceRequest* message. An application needing a resource starts a query using a resource discovery algorithm it wants. When the message arrives to a node which has the requested resource, it handles the message according to the algorithm and sends the reply message back to the initiator of the request message using the following method.

Chedar tries to guarantee that the *ResourceReply* message is forwarded to the initiator of the request message by using a simple method. Every node keeps information about *ResourceRequest* messages it has forwarded. The nodes save the ID of the message and the

startChedar()	Starts a Chedar node.
startConnecting()	The node starts establishing connections.
setMonitor(iMonitor monitor)	Sets a monitoring application for events.
pingMessage()	Checks if a Chedar node is still alive.
connect(String password)	Connects to the node. Returns true if the password is correct.
getMyID()	Returns node's ID.
getNeighbors()	Returns neighbors' IDs.
setResources(Node resource)	Sets a resource node to the XML tree.
unsetResource(String xpath)	Removes corresponding resource from the XML tree using an XPath expression.
listResources()	Returns a list of resources the node has.
createResourceQuery(String query, String algorithm, int ttl)	Creates a resource request message where the query is the searched resource as XPath, the algorithm is the used search algorithm and the ttl is a time-to-live value. Returns the id of the created message.
stopMessage(String id)	Stops forwarding the message with a given id.
setTrafficLimit(int limit)	Sets a value for the traffic limit.
getTrafficLimit()	Returns the value of the traffic limit.
getTrafficMeter()	Returns the value of the traffic meter.
resetTrafficMeter()	Sets the value of the traffic meter to zero.
forceConnection(String id)	Establishes a connection to the node specified with the id.
forceDisconnection(String id)	Disconnects the neighbor specified with id.
closeAllConnections()	Disconnects all node's connections.

Table 1. Methods accessible for applications running on top of Chedar.

IDs of the two previous nodes of the path the message arrived from. The reply message is forwarded to the initiator of the request message using the same path as the request came from, i.e. the reply message is sent to the connection where the request arrived from. This is a common way to route the replies in P2P networks because it needs only information about the previous node. In Chedar there is also other ways if the forwarding fails. If the connection to the neighbor is not available anymore, for example the neighbor has left the network, the node tries to establish a new connection to the second next node on the return path, i.e. the second previous node on the query path and sends the message there. If this does not work either, finally the node tries to establish a new connection to the initiator of the query and sends the reply message directly to it. Establishing always a direct connection to the initiator of the query would require establishing a new TCP connection, which is not always possible, e.g. in the presence of firewalls. Also keeping statistics of which nodes have relayed replies would not be possible.

5 Topology Management Algorithms

Chedar contains four algorithms for managing the topology: Node Selection for adding neighbors, Node Removal for removing neighbors, Overload Estimation for limiting the node's traffic and Overtaking for moving in the network. The algorithms have been further developed and tested in the P2PRealm simulator [7] and the behavior of the algorithms is analysed in [3].

5.1 Node Selection Algorithm

The initial list of neighbors can be obtained manually by out-of-band methods or automatically by using advertisement systems [17] or centralized entry point directories [5]. This has not been implemented in Chedar, but instead it has been left to the application running on top of Chedar. The Node Selection Algorithm handles only the case when a Chedar node already knows some nodes in the network.

When the node joins the network again it tries to establish the connections it had before leaving the network, i.e. connections saved in the active connections. In the best case it manages to establish all connections it had earlier. If the node does not manage to establish any of those connections or it needs a new connection for other reasons, it searches the next one from the history as shown in Algorithm 5.1.

First it searches connections which have hit values and tries to create a connection to one of those.

Because the node does not want to create a connection to the same node it has just dropped it searches only the connections which have not been requested in a given time. If the node did not succeed in establishing a new connection, it next searches connections based only on the time of the last request, i.e. the node has not tried to create a connection to those in a given time or at all (lacking requested information). If the node still did not successfully create a connection, it searches connections without information for hit values or request time. If the node does not have neighbors then the last way to search for a new connection is to try the connections in the history which have hit values. Then the node may select again a neighbor which it has just dropped.

Algorithm 5.1 (NodeSelectionAlgorithm)
Input: Connections h_is in node's history $H = \{h_1, ..., h_m\}$, *time* sets a limit for the time which older the previous connection request must be and *neighbors* is the number of node's neighbors.
Output: Establishes a new connection
hitsNeeded = true
timeNeeded = true
C = SearchConnections(*hitsNeeded, timeNeeded, time, H*)
for i=1 **to** $\mid C \mid$ **do**
  if EstablishConnection(c_i) **then do**
    return c_i
  end if
end for
hitsNeeded = false
timeNeeded = true
C = SearchConnections(*hitsNeeded, timeNeeded, time, H*)
for i=1 **to** $\mid C \mid$ **do**
  if EstablishConnection(c_i) **then do**
    return c_i
  end if
end for
hitsNeeded = false
timeNeeded = false
C = SearchConnections(*hitsNeeded, timeNeeded, time, H*)
for i=1 **to** $\mid C \mid$ **do**
  if EstablishConnection(c_i) **then do**
    return c_i
  end if
end for
if *neighbors* == 0 **then do**
  hitsNeeded = true
  timeNeeded = false
  C = SearchConnections(*hitsNeeded, timeNeeded, time, H*)
  for i=1 **to** $\mid C \mid$ **do**
    if EstablishConnection(c_i) **then do**
      return c_i
    end if
  end for
end if

The function SearchConnections(*hitsNeeded, timeNeeded, time, H*) returns those connections $C = \{c_1, ..., c_n\} \subseteq H$ which meet the criteria defined in the parameters. If the value of the parameter *hitsNeeded* is true, then the function only returns the connections which have hit values. If *hitsNeeded* is false the function returns the connections which do not have hit values. If the value of the parameter *timeNeeded* is true, then the function only returns the connections which have not been requested in the time defined in the parameter *time*. The function EstablishConnection returns true, if the connection was established successfully.

5.2 Node Removal Algorithm

When a node wants to remove a connection it selects the worst neighbor among the neighbors it currently has. The worst neighbor has the smallest goodness value. The goodness is the sum of the neighbor connection's hit values and relayed hits.

$$Goodness = hits + relayedhits \tag{1}$$

The Node Removal Algorithm (Algorithm 5.2) is described as follows.

Algorithm 5.2 (NodeRemovalAlgorithm)
Input: Connections $C = \{c_1, ..., c_n\}$, where c_is are node's neighbor connections.
Output: Removes the worst connection.
c = null
$lowestGoodnessValue = \infty$
for i=1 **to** $\mid C \mid$ **do**
  g = Hits(c_i)+ RelayedHitsSum(c_i)
  if $g < lowestGoodnessValue$ **then do**
    $c = c_i$
    $lowestGoodnessValue = g$
  end if
end for
if $c \neq$ null **then do**
  DisconnectConnection(c)
end if

The function Hits(*connection*) returns the connection's hit values and the function RelayedHitsSum(*connection*) returns the sum of relayed hits

of the connection's neighbors. The method DisconnectConnection(*connection*) removes the connection to the neighbor.

5.3 Overload Estimation Algorithm

There is no predefined number for the connections the node should maintain. Thus the connections are added and dropped based on the amount of traffic going through the node. The Overload Estimation Algorithm compares the traffic meter value calculated in the ConnectionManager to the predefined traffic limit values. There are upper and lower traffic limits which set the range where the traffic amount should be. If the traffic is more than the predefined upper traffic limit, one connection is dropped by using Algorithm 5.2. If the traffic is less than the lower traffic limit it tries to add a new connection using the Algorithm 5.1. At the end, the algorithm resets the traffic meter by setting its value to zero.

Algorithm 5.3 (OverloadEstimationAlgorithm)
Input: Connections $C = \{c_1, ..., c_n\}$, where c_is are node's neighbor connections, value of the traffic meter *meter* in kilobytes, value of the upper traffic limit *upperLimit* in kilobytes and value of the lower traffic limit *lowerLimit* in kilobytes.
Output: Establishes a new connection or removes one connection.

```
overloadFlag = false
if meter > upperLimit ∧ | C |> 1 then do
    overloadFlag = true
    NodeRemovalAlgorithm(C)
end if
if meter < lowerLimit then do
    NodeSelectionAlgorithm()
end if
meter = 0
```

The variable *overloadFlag* is true if the traffic amount is greater than the traffic limit. In that situation the node does not accept any new connections.

5.4 Overtaking Algorithm

The Overtaking Algorithm is used to optimize the topology. The purpose of the algorithm is that the node moves in the network closer to the nodes which provide it a lot of replies by overtaking the current connection. The node does not directly connect to a neighbor of the resource providing node but only closer step by step and that way makes sure that it does not lose good nodes on the path.

The idea is that when a reply message arrives to the querier it updates the hit value of the replier node and updates the local information concerning the relayed hits of the neighbor of the connection from which the node got the reply message. Then if the connection's hit value is bigger than 1, i.e. the node has got more than one message from the neighbor, the node checks whether the connection has a neighbor node whose proportion of the sum of all neighbors' relayed hits and connection's hits is more than the defined overtaking percent. For example if the overtaking percent is 60% it means that if there is the neighbor of the connection which has forwarded over 60% of all reply messages the node has received from the connection then the node establishes a new connection to that node and drops the current connection.

The advantages of the algorithm are that the distances of the nodes which use others' resources are shorter than in randomly connected networks. The algorithm creates clusters gathering close to its center the nodes which provide a lot of resources used by other nodes.[2, 3]

Algorithm 5.4 (OvertakingAlgorithm)
Input: Overtaking percent *overtakingPercent*, node's neighbor connection c, c's neighbors $N = \{n_1, ..., n_n\}$ and c's hit value $hitValue$.
Output: Node has overtaken a neighbor if some neighbor's neighbor is better for the node.

```
if hitValue > 1 then do
    sum = 0.0
    biggest = overtakingPercent/100.0
    bestNeighbor = null
    sum += Hits(c) + RelayedHitsSum(c)
    for i=1 to | N | do
      hitValue = RelayedHits(n_i)
      proportion = hitValue/sum
      if proportion ≥ biggest then do
        biggest = proportion
        bestNeighbor = n_i
      end if
    end for
    if bestNeighbor ≠ null then do
      if EstablishConnection(bestNeighbor) then do
        DisconnectConnection(c)
      end if
    end if
end if
```

The function Hits(*connection*) returns the connection's hit values, the function RelayedHitsSum(*connection*) returns the sum of the relayed hits of the connection's neighbors and the function RelayedHits(*neighbor*) returns the relayed hits of the neighbor. The function EstablishConnection returns

true, if establishing a connection succeeded. The method DisconnectConnection(*connection*) removes the connection to the neighbor.

6 Conclusion

The Chedar peer-to-peer middleware provides a decentralized architecture for P2P applications. The topology of the Chedar network is self-organized by the topology management algorithms and different search algorithms can be used for discovering the resources. Future work of Chedar includes further development of the topology management algorithms and NeuroSearch resource discovery algorithm to optimize the search process as well as the mobile peer-to-peer application development on top of Mobile Chedar.

References

[1] L. A. Adamic, R. M. Lukose, and B. A. Huberman. Local search in unstructured networks. In *Handbook of Graphs and Networks: From the Genome to the Internet*, pages 295–317. Wiley-VCH, 2003.

[2] A. Auvinen. Topology management algorithms in chedar peer-to-peer platform. Master's thesis, University of Jyväskylä, February 2004.

[3] A. Auvinen, M. Vapa, M. Weber, N. Kotilainen, and J. Vuori. New topology management algorithms for P2P networks. Unpublished.

[4] Cheese factory. http://tisu.it.jyu.fi/cheesefactory.

[5] Gnutellahosts. http://www.gnutellahosts.com/.

[6] B. J. Kim, C. N. Yoon, S. K. Han, and H. Jeong. Path finding strategies in scale-free networks. *Physical Review E*, 65, 2002.

[7] N. Kotilainen, M. Vapa, A. Auvinen, T. Keltanen, and J. Vuori. P2PRealm - peer-to-peer network simulator. Unpublished.

[8] N. Kotilainen, M. Vapa, A. Auvinen, M. Weber, and J. Vuori. Peer-to-peer studio - monitoring, controlling and visualisation tool for peer-to-peer networks research. Unpublished.

[9] N. Kotilainen, M. Vapa, M. Weber, J. Töyrylä, and J. Vuori. P2PDisCo - java distributed computing for workstations using chedar peer-to-peer middleware. In *Proceedings of the 19th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2005)*, 2005.

[10] N. Kotilainen, M. Weber, M. Vapa, and J. Vuori. Mobile chedar - a peer-to-peer middleware for mobile devices. In *Proceedings of the Second International Workshop on Mobile Peer-to-Peer Computing (MP2P05)*, pages 86–90. IEEE Press, 2005.

[11] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th International Conference on Supercomputing*, pages 84–95. ACM Press, 2002.

[12] N. A. Lynch. *Distributed Algorithms*. Morgan Kauffmann Publishers, 1996.

[13] S. Nazarko. Evaluation of the data fusion methods using kalman filtering and transferable belief model. Master's thesis, University of Jyväskylä, November 2002.

[14] A. Oram, editor. *Harnessing the Power of Disruptive Technologies*. O'Reilly, Sebastopol, CA, 2001.

[15] R. Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proceedings of First International Conference on Peer-to-Peer Computing (P2P'01)*, pages 101–102, 2001.

[16] M. Vapa, N. Kotilainen, A. Auvinen, H. Kainulainen, and J. Vuori. Resource discovery in P2P networks using evolutionary neural networks. In *International Conference on Advances in Intelligent Systems Theory and Applications (AISTA 2004)*, November 2004.

[17] M. Weber, J. Vuori, and M. Vapa. Advertising peer-to-peer networks over the internet. *Radiotekhnika*, 133:162–170, 2003.

PIII

PEER-TO-PEER STUDIO - MONITORING, CONTROLLING AND VISUALISATION TOOL FOR PEER-TO-PEER NETWORKS RESEARCH

by

Niko Kotilainen, Mikko Vapa, Annemari Auvinen, Matthieu Weber & Jarkko Vuori

2006

ACM International Workshop on Performance Monitoring, Measurement and Evaluation of Heterogeneous Wireless and Wired Networks

P2PStudio – Monitoring, Controlling and Visualization Tool for Peer-to-Peer Networks Research

Niko Kotilainen, Mikko Vapa, Annemari Auvinen, Matthieu Weber, Jarkko Vuori
Department of Mathematical Information Technology
University of Jyväskylä, Finland
firstname.lastname@jyu.fi

ABSTRACT
Peer-to-Peer Studio has been developed as a monitoring, controlling and visualization tool for peer-to-peer networks. It uses a centralized architecture to gather events from a peer-to-peer network and can be used to visualize network topology and to send different commands to individual peer-to-peer nodes. The tool has been used with Chedar Peer-to-Peer network to study the behavior of different peer-to-peer resource discovery and topology management algorithms and for visualizing the results of NeuroSearch resource discovery algorithm produced by the Peer-to-Peer Realm network simulator. This paper presents the features, the architecture and the protocols of Peer-to-Peer Studio and the experience gained from using the tool for peer-to-peer networks research.

Categories and Subject Descriptors
C.4 [**Performance of Systems**]: Measurement techniques

General Terms: Measurement, Performance.

Keywords
peer-to-peer; P2PStudio; monitoring tool; research infrastructure.

1. INTRODUCTION
Peer-to-Peer (P2P) networks consist of a set of peer nodes. Each peer node makes decisions on where to connect and where to forward resource queries resulting in a complex self-organizing network. Studying how different algorithms are performing requires collecting data from the entire P2P network to obtain a global view. In P2P networks research people have used crawlers [5,9] to collect data locally available for some peer nodes. This approach however is only able to gather a portion of the P2P network's behavior, because some of the peers might not accept any new connections requested by the crawlers. Also, the crawlers can only gather information, which is accessible by the P2P protocol and thus they do not have direct means to control the peer's actions.

In our approach, we use a centralized server to contact peers in the P2P network and to set filters to the peers for what events the peers need to report back to the server. This allows measuring different properties from the P2P network extensively and globally. The graphical user interface presents the collected data visually thus making the interpretation easier compared to reading plain text log files. In contrast to crawlers, we note that our work is the first attempt to create a P2P research environment, which provides strict control mechanisms and accurate measurements for studying the behavior of different P2P algorithms.

To monitor the events of a P2P network a specific monitoring interface needs to be implemented in the peer nodes. This interface is used for setting different event logging options and for accepting incoming connections for data delivery from the centralized server. However, in presence of a large P2P network the centralized server can have lots of connections to manage and presents a potential performance bottleneck in our approach compared to local gathering of data done by crawlers. This architecture can however be scaled up by using multiple servers as is common in studies with crawlers [9].

The rest of the paper is organized as follows. Section 2 presents P2PStudio, its features, architecture and protocols. Section 3 describes how P2PStudio has been used in peer-to-peer networks research for studying the performance of peer-to-peer resource discovery and topology management algorithms. Conclusions and future work are discussed in Section 4.

PM²HW²N'2006, October 2, 2006, Torremolinos, Malaga, Spain.

2. PEER-TO-PEER STUDIO
The Cheese Factory –project [3] has implemented a Java-based peer-to-peer computing platform called Chedar [1]. Chedar can be used to build a network of workstations where each node provides and consumes resources such as computing power, files and devices. Currently, Chedar is used as a middleware for P2P Distributed Computing applications [7]. Chedar has also been extended to support mobile devices [8]. In order to test and monitor the Chedar network there was a need for a tool that enables to remotely control and monitor each peer and workstation in a centralized way. By executing the Guardian student project [4], the first version of Peer-to-Peer Studio was developed in 2002.

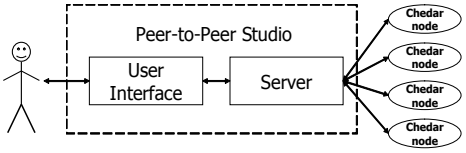

Figure 1. Components of Peer-to-Peer Studio.

P2PStudio is Java-based and it is divided into two separate programs as shown in Fig. 1: the user interface (UI) and the server. The graphical UI connects to the server program and uses

it to carry out the commands entered by the user. The server program takes care of all of the communication between the UI and Chedar nodes. It also manages the data sent from Chedar nodes. Dividing the application into two programs allows mobility of the UI from the dedicated hardware of the server. For example the server might have privileges to connect to Chedar nodes through firewalls and an UI residing on a laptop only needs to be able to connect to the server.

UI communicates with the server, sends requests to Chedar nodes, displays data from the server to the user e.g., by visualizing the network topology and showing diagrams. The UI also allows the management of Chedar nodes. Server forwards the commands sent by the UI, gathers information from the Chedar network and passes on requested data to the UI.

2.1 User Interface

The user interface draws a logical topology of the monitored network as shown in Fig. 2. From the zoomable topology view the user can select nodes and for example check their values, command queries to be sent and modify the resources owned by the nodes. Nodes can also be grouped together to ease the execution of a certain action to multiple nodes. Information on the last executed query is also shown in the topology view. The topology is generated using the WTS Veivi component from WTS Networks [12]. The component creates a visualization of network topology from a set of nodes and links optimized to minimum number of overlapping links. The topology is refreshed whenever the user desires or after a set interval.

Another feature of the UI is to show graphs of the monitoring data as shown in Fig. 3. Currently, the only graph implemented is the neighborhood distribution, but other graphs are relatively easy to be plugged in. Graphs are formed by combining multiple events into a single value, like in the neighborhood distribution, where individual neighbor amount notifications are counted and the frequency of certain value creates one data point in the graph. Graphs can be zoomed and shown also in a logarithmic scale.

The log feature of the UI allows the user to keep track of the Chedar network's actions almost in real time. Log presents the event messages coming from the Chedar nodes. The events are notifications of certain network events, for example forwarded queries, new neighbor connections or dropped messages because of congestion in a Chedar node.

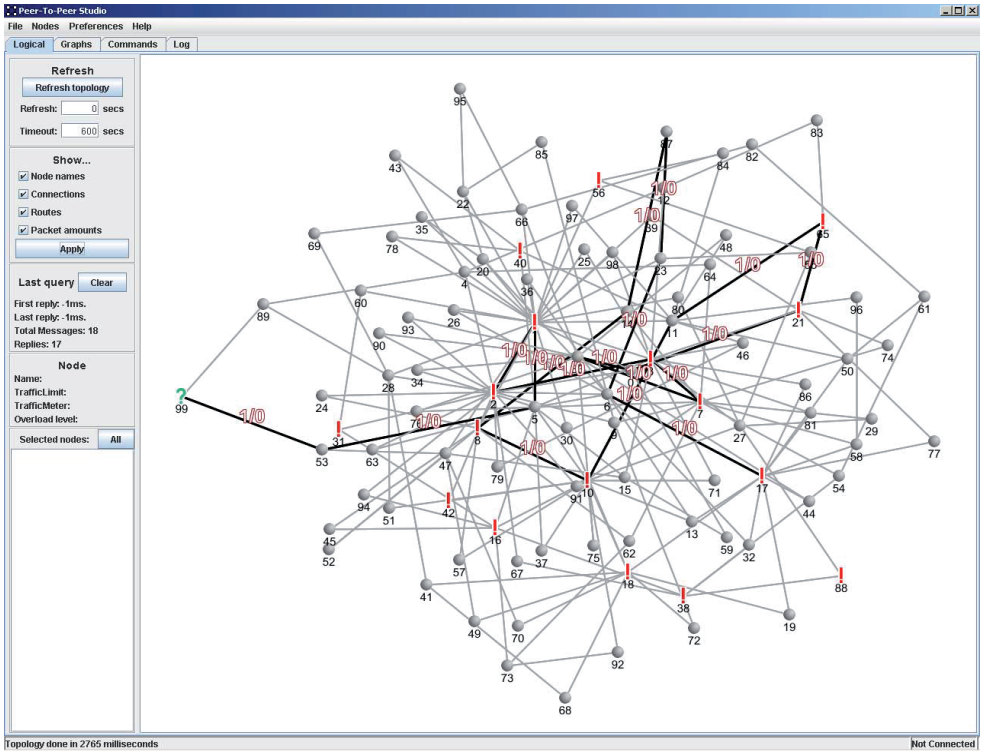

Figure 2. Topology view.

The user can also send commands to the server or to Chedar nodes via the server by typing commands in the User Interface-to-Server Message Protocol (UMP) format (for more details see the Section 2.3). The Commands view allows the user to see the sent data and the received messages from the nodes. Also batch files can be executed via the commands view. Batch files are useful when a certain peer-to-peer query pattern and measurement scenario needs to be executed multiple times.

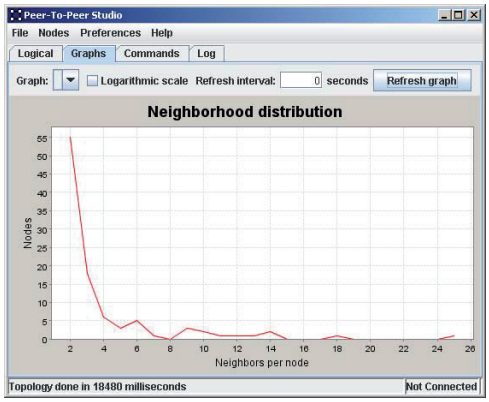

Figure 3. Graph view.

The UI can be run online as well as offline especially for demonstrations. For offline use there is a recording feature allowing the user to record actual monitoring data coming from the server to a file and later retrieve the recorded data in offline state. The UI also allows the user to create Chedar node groups and manage connections.

The functioning of the UI is quite simple. When data is received from the server it is checked and forwarded to the addressed component of the UI. The data will be presented to the user in a form of topology, graph or text depending on the view. Sending data is also rather straightforward. The user assigns a command and it is sent to the server for further handling.

2.2 Server

The server program is divided into two main components: stateless connection manager and stateful data manager. The connection manager is the part of the server which takes care of all connections. It forwards the contents of the packets without interpreting them, only adding metadata about the time the packet was received and Chedar node's IP address and port. A packet can arrive to the server either from the UI or from a Chedar node. It arrives first to the connection manager which forwards it to the data manager if necessary, otherwise directly to UI or to Chedar node(s).

The data manager is responsible for temporarily saving data coming from Chedar nodes and for combining multiple individual replies to a single reply for UI. For example to construct a neighbor distribution graph, data manager needs to collect individual neighbor amounts from Chedar nodes and build the graph data for UI. This lightweight architecture of the server allows scaling to hundreds of Chedar nodes.

2.3 Protocols

User Interface, Server and Chedar nodes use three different protocols for communication. One binary protocol was developed as a container for two message protocols, one XML protocol for communication between the server and the Chedar nodes as well as one XML protocol for communication between the UI and the server. Both XML protocols are on the top of the binary protocol as illustrated in Table 1. The binary protocol is always on the top of TCP.

Table 1. LAYERS OF THE PROTOCOLS.

Message Protocol (GMP or UMP)	*XML*
Packet Transmission Protocol (GPTP)	*Binary*
TCP	

1) Guardian Packet Transmission Protocol (GPTP)

The Guardian Packet Transmission Protocol (GPTP) is a binary protocol used between the UI and the server as well as between the server and the Chedar nodes. The GPTP packets are composed of a fixed-size 64-bit header and a data part, which varies in size. The header identifies the packet as a part of the Guardian-to-Chedar protocol and specifies the size of the data part in bytes. Without a specified data size, parsing an incoming XML message from a stream would be harder. An example of a GPTP message is shown in Table 2.

Table 2. GUARDIAN PACKET TRANSMISSION PROTOCOL.

32 bit synchronization header, 0x47324350 (G2CP)
32 bit size field, network byte order, (1234)
Byte data

2) Guardian Message Protocol (GMP)

The Guardian Message Protocol (GMP) is used between the server and the Chedar nodes on the top of the Guardian Packet Transmission Protocol. Each GMP message is a complete XML document. The header is a standard XML declaration, and the body is composed of a root element which specifies the type of message, and a variable content.

Here is the structure of GMP message:

Header: XML declaration

<?xml version="1.0" encoding="UTF-8"?>

Body

Root element: <request/> OR <reply/> OR <event/>

Content: various requests, replies or events as

XML elements and/or attributes

There are three types of messages in the Guardian Message Protocol:

Request message is sent by the server to a Chedar or a Workstation node.

Reply message is sent by a Chedar or a Workstation node to the server.

Event message is sent by a Chedar node to the server.

The request/reply pair forms a synchronous message exchange initiated by the server. The reply is not mandatory. Event messages can arrive from the Chedar nodes at any time.

3) User Interface-to-Server Message Protocol (UMP)

The User Interface-to-Server Message Protocol (UMP) is used between the UI and the server on top of the Guardian Packet Transmission Protocol. UMP uses similar message structure as GMP. The difference between UMP and GMP is in the XML elements and attributes. For example the UMP contains elements for sending a certain GMP message to all Chedar nodes.

3. P2PSTUDIO IN PEER-TO-PEER NETWORKS RESEARCH

At first, P2PStudio was developed to collect data from a Chedar network [1] consisting of tens of workstations. Experimenting with self-organization of topology and different resource discovery algorithms however usually requires a controlled environment to obtain results that are repeatable. Creating exactly same starting conditions for each test in a network of workstations is problematic, because of differencies in hardware and network traffic. Also, having each Chedar node pack and send data over the network is significantly slower than executing algorithms in a simulator, where only local data structures are being used.

Therefore, the use of P2PStudio was extended by creating the Peer-to-Peer Realm (P2PRealm) network simulator [10,6]. P2PRealm is Java-based and contains functionalities for creating peer-to-peer network scenarios with different topologies, resource distributions and query patterns, executing different resource discovery and topology management algorithms, and collecting various statistics of the execution to log files. In addition to textual viewing of log files, P2PStudio can be used for graphical viewing e.g., to plot how queries spread in the network and what kind of topologies emerge from the execution of algorithms.

A special use case for P2PStudio and P2PRealm is the development of the NeuroSearch resource discovery algorithm [11], which is based on neural networks. Optimizing neural networks requires not only simulation of a certain scenario once, but usually thousands of times to reach a near-optimum state in learning. Therefore network simulators, such as Ns-2 [2], which are based on scripting languages and mainly developed for detailed protocol studies are not fast enough. For studying the behavior of neural networks, P2PStudio provides a view containing the inputs of neural network and the corresponding output decisions.

4. CONCLUSIONS AND FUTURE WORK

P2PStudio is a well-established research tool for peer-to-peer networks research providing functionalities for peer-to-peer network monitoring, controlling and visualization. P2PStudio has been used with two different peer-to-peer software, Chedar and P2PRealm, for algorithm development. The centralized architecture of P2PStudio is a potential bottleneck for scalability in the future when the size of the P2P networks being studied grows. As a future work we envision changes in the architecture to support multiple servers as well as adding new functionalities to UI to determine certain network characteristics such as diameter, shortest paths and multiple distinct paths between nodes.

5. ACKNOWLEDGMENTS

The authors would like to thank the other members of the Guardian student project: Joni Töyrylä, Jussi Rastas and Ville Pentti. Niko Kotilainen was supported by the InBCT-project and Mikko Vapa and Annemari Auvinen were supported by the GETA graduate school.

6. REFERENCES

[1] A. Auvinen, M. Vapa, M. Weber, N. Kotilainen, and J. Vuori, “Chedar: Peer-to-Peer Middleware”, *Proceedings of the 20th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2006)*, Rhodes Island, Greece, Arpil 2006.

[2] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan K. , X. Ya, and Y. Haobo, “Advances in network simulation”, IEEE Computer, Vol. 33, Issue 5, pp. 59-67, 2000.

[3] Cheese Factory – Peer-to-Peer Computing Project, tisu.it.jyu.fi/cheesefactory.

[4] Guardian project, www.mit.jyu.fi/opiskelu/sovellusprojektit/guardian/.

[5] M. A. Jovanovic, F. S. Annexstein, and K. A. Berman, “Scalability Issues in Large Peer-to-Peer Networks – A Case Study of Gnutella”, Technical report, University of Cincinnati, 2001.

[6] N. Kotilainen, M. Vapa, A. Auvinen, T. Keltanen, and J. Vuori, ”P2PRealm – Peer-to-Peer Network Simulator”, *Proceedings of the 11th International Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks (CAMAD 2006)*, Italy, June 2006 .

[7] N. Kotilainen, M. Vapa, M. Weber, J. Töyrylä, and J. Vuori, "P2PDisCo – Java Distributed Computing for Workstations Using Chedar Peer-to-Peer Middleware*", Proceedings of the 19th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2005)*, Denver, Colorado, USA, 2005.

[8] N. Kotilainen, M. Weber, M. Vapa, and J. Vuori, "Mobile Chedar - A Peer-to-Peer Middleware for Mobile Devices", *Workshops Proceedings of the Third IEEE Conference on Pervasive Computing and Communications (Percom 2005)*, pp. 86-90, Kauai Island, Hawaii, USA, 2005.

[9] D. Stutzbach, R. Rejaie, “Capturing Accurate Snapshots of the Gnutella Network”, *Proceedings of the 8th IEEE Global Internet Symposium*, Miami, Florida, 2005.

[10] J. Töyrylä, "Building NeuroSearch - Intelligent Evolutionary Search Algorithm For Peer-to-Peer Environment", Master's Thesis, University of Jyväskylä, 3.9.2004.

[11] M. Vapa, N. Kotilainen, A. Auvinen, H. Kainulainen, and J. Vuori, "Resource Discovery in P2P Networks Using Evolutionary Neural Networks", *International Conference on Advances in Intelligent Systems - Theory and Applications (AISTA 2004)*, Luxembourg, 2004.

[12] WTS Networks, www.wts.fi

PIV

P2PREALM - PEER-TO-PEER NETWORK SIMULATOR

by

Niko Kotilainen, Mikko Vapa, Annemari Auvinen, Teemu Keltanen & Jarkko Vuori
2006

IEEE 11th International Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks

P2PRealm – Peer-to-Peer Network Simulator

Niko Kotilainen, Mikko Vapa[+], Teemu Keltanen, Annemari Auvinen[+] and Jarkko Vuori
Department of Mathematical Information Technology, University of Jyväskylä
P.O.Box 35 (Agora), 40014 University of Jyväskylä, Finland
firstname.lastname@jyu.fi

***Abstract*—Peer-to-Peer Realm (P2PRealm) is an efficient peer-to-peer network simulator for studying algorithms based on neural networks. In contrast to many simulators, which emphasize on detailed network simulation, the speed of simulation in P2PRealm is essential, because neural networks require a time consuming training phase. Efficiency has been obtained by optimizing training loops inside the simulator, using Java Native Interface (JNI) as well as distributing the simulator to hundreds of workstations using the P2PDisCo platform. In this paper we describe the architecture of P2PRealm and its input/output interfaces. Also, we present the mechanisms used for internally optimizing the implementation and the configuration used for distribution. Finally, we present the use of P2PRealm with the P2PStudio network visualization tool.**

Keywords - peer-to-peer; P2PRealm; network simulation; research infrastructure; neural networks, optimization methods;

I. INTRODUCTION

Peer-to-Peer (P2P) algorithms have been studied at least using three different approaches. These are crawlers, emulators and simulators. Crawler is an implementation of a peer node specially designed for P2P networks research. A crawler can collect data passing through it to get a local view of the P2P network. By deploying multiple crawlers, a bigger part of the peer-to-peer network can be monitored. However, this approach is not able to gather the global view of the network, because the behavior of nodes which are not connected to crawlers is unknown. This problem is solved by emulators, which contain the implementation of a peer node and are used to build a complete P2P network. Multiple emulators can be deployed inside one workstation usually providing quite large P2P networks with only a few workstations.

Even though emulators can be used to get the global view, they are restricted to slow execution, because messages need to be passed between emulator processes through network protocols such as TCP. The third option, simulator, contains an abstracted implementation of peer nodes equivalent to emulators, but uses local data structures for message passing. The use of local data structures significantly increases the speed of execution and therefore is well-suited approach for computationally intensive algorithm studies. The downside of the approach is the inaccuracy of results compared to real-world P2P networks and the difficulty of modeling user behavior. This knowledge can only be obtained using crawlers or by monitoring network traffic inside routers.

Developing peer-to-peer resource discovery and topology management algorithms based on neural networks are computationally intensive tasks. For example, it takes a week for one low-cost workstation to train a good neural network based resource discovery algorithm for a rather small P2P network [23]. Therefore, in computationally intensive algorithmic research the most important factor to consider for a simulator is speed.

This paper describes an end product of a process where emulators were first used for studying P2P algorithms and later re-implemented as an efficient simulator to decrease the time used for execution. Latest improvement of the simulator is the distributed execution on a Peer-to-Peer Distributed Computing platform (P2PDisCo) [11] allowing us to parameter sweep different features of neural network based P2P algorithms.

The paper is structured as follows. Section 2 compares existing P2P simulators with our work and states their differences. Section 3 introduces P2PRealm network simulator and section 4 describes its input and output interfaces. Section 5 describes the main use case of P2PRealm: the training of evolutionary neural networks for P2P resource discovery. Section 6 describes the internal modifications used for optimizing the code and the combination of P2PRealm with P2PDisCo platform. Section 7 illustrates the use of P2PStudio for visualizing the output of P2PRealm and section 8 concludes the paper.

II. RELATED WORK

There are various network simulators available for studying P2P networks. However, many of these simulators are not primarily designed for speed and none of them contains functionalities for neural networks. Because the speed is the most important factor in our simulation environment, it is obvious that abstractions on the level of details are necessary. Packet-level simulators model the P2P protocols with precise protocol headers and field structures, whereas message-level simulators only take into account the number and sizes of the

[+] The works of M. Vapa and A. Auvinen are supported by Graduate School in Electronics, Telecommunications and Automation (GETA).

packets. While packet-level simulation is a desirable feature, it is still often too expensive in terms of computing resources. In addition to speed, other desirable features in our simulations are compilation of statistics on simulation results and visualization of P2P networks. From this viewpoint, we next overview some of the existing P2P simulators.

A. NS-2

NS-2 [15] is one of the most widely used network simulators. The NS-2 is object-oriented discrete event simulator, which closely follows the architecture of the OSI model. It suits well for simulating packet switched networks and small scale networks. The Parallel and Distributed Simulation (PADS) research group has developed an extension that allows network simulation to be run in parallel on multiple machines [17]. Being very detailed simulator, it still does not scale well enough and is slow from a computational point of view. In addition, adding new modules is not straightforward, because of it's complex module structure [14].

B. PLP2P

Packet-level Peer-to-Peer Simulator (PLP2P) [6] provides a framework for other packet-level simulators, e.g. NS-2, in order to provide detailed model of the underlying network. This is done with wrappers, which translate P2P events into underlying packet-level simulator. The authors assert that abstracting low-level details can impact the simulation results to a large extent. The scalability problem of packet-level simulations is solved by running simulations on parallel machines. Nevertheless, as training neural networks requires substantial part of available computing power in order to get result within reasonable time, we need to abstract the level of details of the P2P network.

C. QueryCycle

The QueryCycle simulator [19] is specialized to file-sharing simulations. It has realistic models for content distribution, query activity, download behavior etc. The content distribution is based on a model, where each file belongs to one category and that category is defined by the popularity of the file. Simulations proceed in query cycles representing the time period between issuing a query and receiving a response. Generated queries are passed into a queue and handled on a First-In-First-Out basis.

D. 3LS

3LS [21] is an open-source simulator for overlay networks designed to overcome the problems of extensibility and usability. The system is separated to three architectural levels: a network model, a protocol model and a user model. The network model uses a two-dimensional matrix as a storage of distances between the nodes. The protocol model defines the current protocol being simulated. The user model is the input interface for the user. The 3LS uses most of the memory resources to a graphical interface as the simulator uses main memory to store each event executed for visualization and this limits the system to less than a thousand nodes on a low-cost workstation [14].

E. PeerSim

PeerSim [18] has been developed especially with scalability and support for dynamicity in mind. PeerSim is Java-based and has two simulation engines, one is cycle-based and the other is event driven. Cycle-based engine allows scalable simulation but is not very accurate. Handling large-scale overlay networks requires simplifying assumptions about the simulation details. For example, the details of the transport communication protocol stack are not taken into account. Event driven engine supports dynamicity and is more realistic, but decreases the scalability of the simulation. The abstractions of cycle-based simulations are similar to ours. The difference is that when PeerSim uses the benefits of abstraction for high scalability, we use it to increase the computational efficiency. Parallel execution is a necessity in order to PeerSim to be useful for our research.

F. NeuroGrid

The NeuroGrid simulator [8] was initially designed to support comparative resource search simulations between FreeNet [5], Gnutella [16] and NeuroGrid [8] systems. The simulator is single-threaded, Java-based and uses discrete events. Several protocols are now available for NeuroGrid e.g., Domain Name Service (DNS) and a distributed e-mail protocol. NeuroGrid supports property files that specify the parameters of the simulation to run. This includes the protocol to simulate, the parameters of the network and the amount of searches.

NeuroGrid would be a promising simulator for our research if it wasn't single-threaded and non-parallel.

G. GPS

The General Peer-to-Peer Simulator (GPS) [24] is aiming to respond to a call for extensible framework for simulating P2P networks efficiently and accurately. Efficiency is accomplished with message-level simulation instead of packet-level simulation. Improvement to the level of detail is achieved by tracking the network infrastructure and using a macroscopic mathematical model to obtain accurate estimate of the message behavior, e.g. TCP. The GPS also models downloads of the files, which is often left out from the simulators. GPS is extensible for modeling any P2P protocol, integration with a GUI and network visualization and provides support for topology generation tools.

The GPS is still in it's early stages and details about scalability, usability and performance are scarce. The GPS has also only been used for simulating BitTorrent, where resource discovery is not an essential problem. It is single-threaded, but

according to the authors their aim is to include multi-threading into the simulator in the future.

H. Summary

A comparison of the different characteristics for reviewed P2P simulators is shown in Table I. Overlay with Routers column tells if the simulator contains both the logical overlay network topology and the underlying router structure of the physical network. After surveying the existing literature about P2P simulators it is obvious that there is a need for standardization in the area of P2P simulation [7]. The field is highly fragmented and most of the current projects use their own simulators tailor-made for their purposes. One of the problems of the widely used simulators is their complexity and therefore poor scalability for P2P simulation purposes.

Although our project strongly supports the call for general open-source P2P simulator that is easily extensible and even some attempts for such a simulator have recently appeared, our research area is still too specified to be implemented satisfactorily in any other way than building a specifically optimized simulator. Training neural nets is computationally very demanding and requires parallel computing and simplified network simulation. To the best of our knowledge P2PRealm is the only message-level simulator that allows simulations on parallel machines. In addition, there is no other neural networks based P2P algorithms that we know of and neither simulators supporting neural network algorithms.

The problem of specifically built simulators is that the results are not exactly similar with the ones made by other simulators. This is a compromise that had to be made. In P2PRealm the most common P2P resource discovery algorithms are implemented to allow comparison with the ones neural networks create. This provides the baseline for results obtained in other simulators.

III. PEER-TO-PEER REALM

Peer-to-Peer Realm (P2PRealm) is a Java based peer-to-peer network simulator designed for optimizing neural networks used in P2P networks. The simulator has been developed in Cheese Factory peer-to-peer research project [4]. With the simulator, it is possible to determine a certain P2P network scenario and requirements for a resource discovery or topology management algorithm and get as an output a neural network optimized for that scenario. For example, a P2P network scenario could contain Gnutella's [16] topology, resource distribution and query pattern and the requirements could state that we want an algorithm, which needs to locate certain amount of resources (say 150) using as few query packets as possible. The end result would be an adapted resource discovery algorithm for that particular P2P network scenario. The first results of this kind of an algorithm development was reported in [23]. Also, the simulator contains implementations of various P2P resource discovery algorithms such as Breadth-First Search [13], Random Walker [12], Highest Degree Search [1,22] and optimal path K-Steiner Tree approximation [22]. These algorithms can be used as performance measures for neural network based algorithms or for studying their performance in different P2P network scenarios. The simulator has also been used for studying topology management algorithms for P2P networks [3].

The simulator is divided into four parts: P2P network, P2P algorithms, neural network optimization and input/output interface. P2P network contains the characteristics of a P2P network including the network topology, distribution of resources and query patterns of P2P network users. P2P algorithms contains the implementations of various resource discovery and topology management algorithms. Neural network optimization takes care of neural network structure and different optimization algorithms used for training the neural network structure. Input/output interface is used for reading configuration files and for outputting the statistics of training and final results. The final results consist of the optimized

TABLE I. CHARACTERISTICS OF THE CURRENT UNSTRUCTURED P2P NETWORK SIMULATORS

	Level of Detail	Parallel	Scalability	Overlay with Routers	Dynamic Network	Programming Language
NS-2	Packets	Yes	Very low	Yes	No	C++
PLP2P	Packets	Yes	Medium	-	-	C++
QueryCycle	Messages	No	?	Yes	Yes	Java
3LS	Messages?	No	Very low (<1000 peers)	Yes	?	Java
PeerSim	Messages	No	Very high (10^6 peers)	Yes	Yes	Java
NeuroGrid	Messages	No	High (300 000 peers)	No	Yes	Java
GPS	Messages	No	?	No	Yes	Java
P2PRealm	Messages	Yes	Medium (100 000 peers)	No	Yes	Java

neural network and the used query paths for different queries.

IV. Input and Output Interfaces

The following information is required as an input to P2PRealm (described in a configuration file):

- P2P network topologies containing the resource distribution
- Query pattern
- P2P resource discovery algorithm
- Percentage of available resource instances to be located in each query
- Number of queries executed in each training generation
- Neural network inputs
- Number of training generations, number of neural networks and the neuron structure of neural networks
- Optimization method

As an output Peer-to-Peer Realm (P2PRealm) provides the following files:

- The used topology and neighbor distribution
- A trace of training process with separate files for training and generalization sets
- The best and all neural networks of each generation
- Query routes started from each node of the P2P network
- Configuration file, which was given as an input

V. Training NeuroSearch

Next, we briefly describe how P2PRealm can be used for P2P algorithm development. As an example we use NeuroSearch resource discovery algorithm [23], but other algorithms for example topology management algorithms based on neural networks could be used [9].

NeuroSearch resource discovery algorithm uses local information about query situation in a peer-to-peer network to decide if query should be forwarded to a neighboring peer node or not. The local information can be e.g. number of hops the query has traveled, number of replies still needed to be located etc. The forwarding process is illustrated in the following algorithm:

1. *One peer node starts a query specifying a resource it wants to locate.*
2. *For each neighbor the node has, do the following:*
2.1 *Fill all the input fields of neural network.*
2.2 *Compute the output of neural network.*
2.3 *If output is greater than zero, then forward the query to neighbor and increase the number of sent query packets by one.*
3. *A forwarded query packet arrives to peer node. If this is the first query packet arriving to this node, check whether the peer node contains a resource being queried. If peer has the queried resource then increase the number of found resources by one.*
4. *Go to step 2.*

The algorithm terminates when there are no more query packets to process. At the end the quality of neural network is determined by the number of found resources and the number of query packets used.

To get good neural networks, they need to be trained so the algorithm has to be executed many times (typically millions query executions). There are various neural network weight adjusting algorithms and depending on the used methods the training times can vary a lot. Still, all optimization methods have in common iterative behavior and therefore executing the algorithm efficiently is an important feature of the simulator.

The internal execution loops of P2PRealm used for training NeuroSearch are illustrated in Fig. 1. Each simulation run can have multiple simulation cases, where each case has its own environment parameters according to the input information described in section 4. Furthermore, each case produces NeuroSearch resource discovery algorithm optimized to this environment accordingly. With multiple cases it is possible to do parameter sweeps and to eliminate the need of starting the simulator manually each time one wants to use multiple training environments. The execution of different cases can also be distributed on Peer-to-Peer Distributed Computing platform [11] further described in section 6.

Execution of one case is divided into three different sections:

- Training of the neural networks
- Analyzing the training of best neural network in generalization environment
- Analyzing routes of the best neural network after the training

First, the case has its P2P networks, neural networks and other parameters initialized. Then the simulator proceeds to the training phase. In each generation multiple neural networks are evaluated by forwarding queries according to the resource discovery algorithm presented above. The queries are forwarded in one or more P2P networks and statistics of the query performance of each neural network is recorded at the same time. Usually between generations it is worth to do more specific analysis of the best neural network in generalization

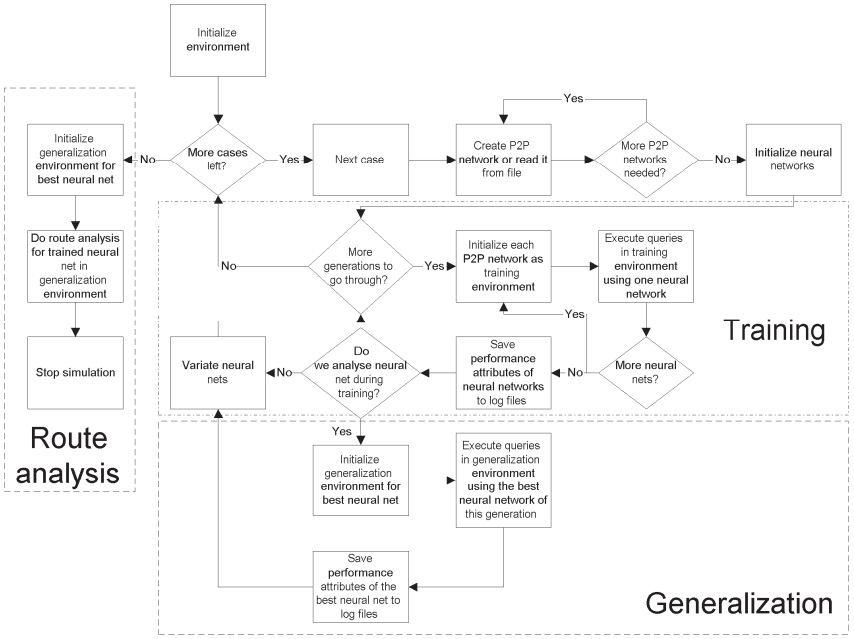

Figure 1. Execution Loops of P2PRealm

environment, where we can determine how the same neural network performs in an unknown environment. Generalization environment can be used to control when neural network is specializing too much on training environment and loses the ability to perform well in unknown but similar environments than the training environment.

After the evolution has proceeded the predetermined amount of generations, the simulator moves to the last phase of the process: route analysis. In the route analysis the same generalization environment is initialized as earlier for the best neural network, but now the queries start from each peer node at a time. The query paths produced by these queries are recorded and written to files to get accurate data about input and output values of neural network during a query. Finally, when routes have been recorded, the simulation ends.

VI. SPEEDING THE EXECUTION WITH P2PDISCO

The first implementation of P2PRealm used approximately one week for training the neural networks on a desktop computer. This was a severe limitation in research because it forced to study only small P2P networks and still getting results was very time consuming.

We started the internal code optimization process to see how much can be saved by optimizing internal loops of the simulator. After P2PRealm was profiled we found that the use of Vector object instead of Array in Java consumed lots of time (in particular getting the size of a vector through method call). Java container classes such as HashMap and HashTable can contain only objects and therefore reimplementing them to store only primitive values saved some execution time. Also we found that caching results of different method calls to avoid new method calls resulted in significantly faster execution times. The total time decreased to about 60% with these optimizations.

Java bytecode is interpreted in Java virtual machine yielding slower execution compared to compiled code. Java Native Interface [20] has been developed to allow native code for example compiled C++ to be executed from a Java program. We reimplemented the calculation of neural network output with C yielding an execution time about 70-80% compared to first version of P2PRealm. Combining both the internal code

optimization techniques and Java Native Interface implementation of neural network output calculation, we thus achieved execution time of about 50% compared to first version of P2PRealm.

This was however not enough, because reducing execution time of one simulation case from a week to 3-4 days was still quite slow. As a solution, we started developing Peer-to-Peer Distributed Computing platform (P2PDisCo) [11] allowing the distribution of simulation cases to multiple machines.

Earlier in our project [4] we had developed Chedar P2P middleware [2], which provided the basis for building P2PDisCo on top of it. P2PDisCo allows the workstations joined in a Chedar P2P network to publish certain distributed computing application as a resource in Chedar P2P network. When other Chedar nodes find this resource, it can be used to deliver needed input files to computing nodes and the produced output files to the node, which started the computations. For further information on the behavior of P2PDisCo the reader is referred to [11].

The speed up of execution with P2PDisCo is nearly linear, because each simulation case is delivered to different workstation. In university environment it is easy to locate machines, which are idle most of the time, so getting hundred of machines (and thus 100 times faster execution) was relatively easy scaling the research process to much faster rates. The resulting architecture is shown in the Fig. 2. Master denotes the peers, which create simulation cases and P2PRealm denotes the peers, which compute these cases.

VII. Visualization of Data using P2PStudio

Peer-to-Peer Studio (P2PStudio) [10] is a monitoring, controlling and visualization tool for P2P networks research. When combined with P2PRealm only visualization features can be used, because current version of simulator does not provide monitoring data during execution of a simulation. For visualization, P2PStudio provides functionalities to draw network topology and different graphs e.g., neighbor distribution of the topology. Also, the location of resources and query paths can be illustrated on a screen to qualitatively analyze how algorithms are performing. In case, that the simulation contains neural network, the input and output values of a certain query will be shown in a separate table. A screenshot of P2PStudio is shown in Fig. 3 and the specific features of P2PStudio are described in separate article [10].

VIII. Conclusions

Peer-to-Peer Realm is a simulator for studying P2P networks. Its unique functionalities contain training methods for neural networks and optimized speed of execution. By combining P2PRealm with other tools developed in our project, the simulator can grow to a large-scale distributed P2P research environment.

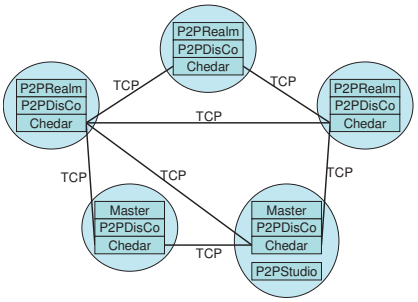

Figure 2. Architecture of P2PRealm combined with P2PDisCo, Chedar and P2PStudio

The future work of P2PRealm includes the parallelization of simulation such that multiple computers can process the same simulation task. Now only one simulation task can be allocated to a certain computer and speed ups are gained only when multiple cases are being simulated. Also, with the advent of multi-core processors for desktop machines, we are going to implement threaded version of simulator to support multiple processors within a single computer. For P2P network visualization, P2PStudio's user interface can be replaced in the future to support large P2P networks to be visualized and better usability of the program. Also the list of improvements for P2PRealm contain different query distributions and new input types for neural networks. As a longer term goal, we are aiming to combine neural network based topology management algorithms with neural network based resource discovery algorithms to study optimal construction of P2P networks.

References

[1] Adamic L., Lukose R. and Huberman B., Local Search in Unstructured Networks, *Handbook of Graphs and Networks: From the Genome to the Internet*, Wiley-VCH, 2003, 295-317.

[2] Auvinen A., Vapa M., Weber M., Kotilainen N. and Vuori J., "Chedar: Peer-to-Peer Middleware", Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2006), Rhodes Island, Greece, 2006.

[3] Auvinen A., Vapa M., Weber M., Kotilainen N. and Vuori J., "New Topology Management Algorithms for Unstructured Peer-to-Peer Networks", unpublished.

[4] Cheese Factory –project, http://tisu.it.jyu.fi/cheesefactory

[5] Clarke I., Sandberg O., Wiley B. and Hong T., "Freenet: A distributed anonymous information storage and retrieval service", *Proceedings of Workshop on Design Issues in Anonymity and Unobservability (ICSI),* Berkeley, CA, USA, 2000.

[6] He Q., Ammar M., Riley G., Raj H. and Fujimoto R., "Mapping Peer Behavior to Packet-level Details: A Framework for Packet-level Simulation of Peer-to-Peer Systems", *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS 2003),* Orlando, USA, 2003.

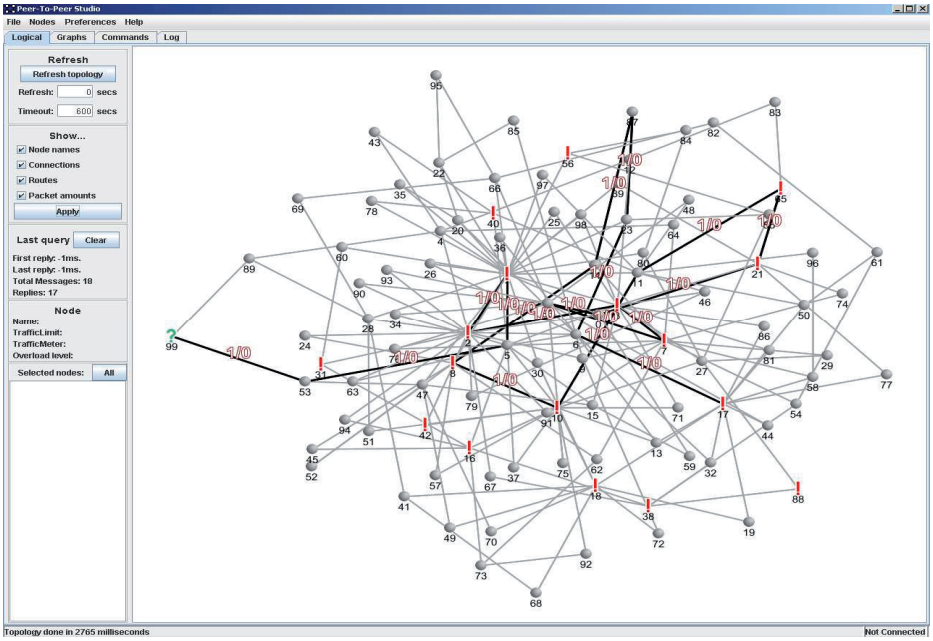

Figure 3. P2PStudio User Interface for P2PRealm

[7] Joseph S., "An Extendible Open Source P2P Simulator", *P2P Journal*, November 2003, 1-15.

[8] Joseph S. and Hoshiai T., "Decentralized Meta-Data Strategies: Effective Peer-to-Peer Search", *IEICE Transactions on Communications*, Vol.E86-B, No.6, 1740-1753.

[9] Keltanen T., "NeuroTopology: Topology Management Algorithm for P2P Networks", unpublished.

[10] Kotilainen N., Vapa M., Auvinen A., Weber M. and Vuori J., "P2PStudio - Monitoring, Controlling and Visualization Tool for Peer-to-Peer Networks Research", unpublished.

[11] Kotilainen N., Vapa M., Weber M., Töyrylä J. and Vuori J., "P2PDisCo - Java Distributed Computing for Workstations Using Chedar Peer-to-Peer Middleware", *Proceedings of the 19th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2005)*, Denver, Colorado, USA, 2005.

[12] Lv Q., Cao P., Cohen E., Li K. and Shenker S., Search and Replication in Unstructured Peer-to-Peer Networks, *Proceedings of the 16th International Conference on Supercomputing*, ACM Press, 2002, 84-95.

[13] Lynch N. *Distributed Algorithms*, Morgan Kauffmann Publishers, 1996.

[14] Montresor A., Di Caro G. and Heegaard P., "Architecture of the Simulation Environment", Technical Report: D11, BISON project, University of Bologna, 2003.

[15] NS-2, http://www.isi.edu/nsnam/ns/

[16] Oram A., *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, O'Reilly Media, 2001.

[17] PDNS - Parallel/Distributed NS, http://www.cc.gatech.edu/computing/compass/pdns/

[18] PeerSim, http://peersim.sourceforge.net/

[19] Schlosser M., Condie T. and Kamvar S., "Simulating a P2P File-Sharing Network", *1st Workshop on Semantics in Grid and P2P Networks*, 2002.

[20] Sun Microsystems, Java Native Interface Specification, http://java.sun.com/j2se/1.5.0/docs/guide/jni/spec/jniTOC.html

[21] Ting N. and Deters R., "3LS - A Peer-to-Peer Network Simulator", *Proceedings of the 3rd International Conference on Peer-to-Peer Computing (P2P 2003)*, IEEE Press, 2003, 212-213.

[22] Vapa M., Auvinen A., Ivanchenko Y. Kotilainen N. and Vuori J., "Optimal Resource Discovery Paths of Gnutella2", unpublished.

[23] Vapa M., Kotilainen N., Auvinen A., Kainulainen H. and Vuori J., "Resource Discovery in P2P Networks Using Evolutionary Neural Networks", *International Conference on Advances in Intelligent Systems - Theory and Applications (AISTA 2004)*, Luxembourg, 2004.

[24] Yang W. and Abu-Ghazaleh N., "GPS: A General Peer-to-Peer Simulator and its Use for Modeling BitTorrent", *Proceedings of the 13th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '05)*, Atlanta, USA, 2005.

PV

NEW TOPOLOGY MANAGEMENT ALGORITHMS FOR UNSTRUCTURED PEER-TO-PEER NETWORKS

by

Annemari Auvinen, Mikko Vapa, Matthieu Weber, Niko Kotilainen & Jarkko Vuori
2007

IEEE Second International Conference on Internet and Web Applications and Services, Best Paper Award.

New Topology Management Algorithms for Unstructured P2P Networks *

Annemari Auvinen, Mikko Vapa, Matthieu Weber, Niko Kotilainen and Jarkko Vuori
Department of Mathematical Information Technology
University of Jyväskylä
P.O.Box 35 (Agora), 40014 University of Jyväskylä, Finland
(annauvi, mikvapa, mweber, npkotila)@jyu.fi, jarkko.vuori@gmail.com

Abstract

In this paper we present new topology management algorithms used to self-organize the overlay of a peer-to-peer network. The algorithms are Node Selection, Node Removal, Overload Estimation and Overtaking algorithms. The algorithms have been evaluated using a simple P2P scenario using the P2PRealm network simulator. Based on the simulation results, the algorithms produce an overlay which is stable and has a short average distance between nodes.

1 Introduction

Peer-to-peer (P2P) technologies have received a lot of publicity lately mainly because of Kazaa and other P2P file sharing systems. Other resources, for example CPU time and storage space, can also be shared in a P2P network. Every peer i.e. a node in the P2P network may provide resources to other nodes and consume the resources other nodes are providing. This means that a node may serve both as a server and a client. Therefore there is no need for a central server which might become the bottleneck of the network or which failure will paralyze the whole network.

The P2P network can be structured or unstructured. In an unstructured network, like Gnutella and our network, a node's place in the network is not pre-defined. A node may join the network by establishing a connection to another node on the P2P network. A resource search is not very efficient in that kind of a network but maintaining the topology does not produce extra work.

The topology management algorithms affect the network's overlay topology by making the network more scalable and effective for resource discovery. Nodes want to stay connected to the network and find resources efficiently without using too much of their own capacity for being in the network. It is suitable for example that the nodes connecting with a modem are in the edge of the network and the nodes capable of handling a lot of traffic are in the center. With topology management algorithms the network can also be kept from partitioning.

In our proposition a central point for the topology management algorithms is the goodness of a peer. A good neighbor provides resources to the node. The node tries to select neighbors so that those are the best nodes the node knows. A decision is made based on the local information the peer has about its neighbors and neighbors' neighbors.

This paper is organized as follows. We present the related work in Section 2. The topology management algorithms are described in Section 3. Test cases and the analysis of the test results are presented in Section 4 and the paper is concluded in Section 5.

2 Related Work

Ramanathan *et al* [13] have developed an algorithm where a peer moves closer to the peers which have provided search results. In the proposed method a peer keeps track of the replies it receives for the sent queries. When a peer finds a good peer, i.e. a peer which provides results and has same high degree of similar interest, it creates a new connection to that peer. This forms clusters with similar interests. Advantages of the method are that it reduces the number of messages, allocates resources efficiently and scales well with respect to the number of peers.

Condie *et al* [5] have proposed a protocol for forming adaptive P2P networks. It is based on the idea that a peer should connect to the peers from which it is likely to download satisfactory content in the future. The peers save local trust values and connection trust values for each peer they are interacted with and use those for estimating the likelihood of a future successful download.

The two preceding methods take into account only the sender of the reply message and a new connection is established directly to that peer. In that case the peer might lose

*This work was funded by the Agora Center InBCT project.

Second International Conference on
Internet and Web Applications and Services (ICIW'07)
0-7695-2844-9/07 $20.00 © 2007 IEEE

good peers which appear on the route between the quering peer and the peer providing a resource.

Pandurangan *et al* [12] have proposed a protocol for forming P2P networks, without any global knowledge about the network, guaranteeing that the distances of the nodes are short. When using the protocol new nodes decide where to connect and nodes make a decision when and how to replace the lost connections. They proved that by using the protocol, the network has a constant degree and a logarithmic diameter. A central point of the protocol is a host server, which makes the system more vulnerable to attacks and failures. The host server may also become a bottleneck of the system because nodes search a new connection from it also other times than just when joining the network.

Chawathe *et al* [3] have developed the distributed and unstructured Gia P2P file sharing system. The goal of the research was to develop the Gnutella-like system which could handle a high query rate and which would work well while the system grows. The purpose was to find the best possible neighbors to a node i.e., the nodes which have a lot of processing power or a large bandwidth, and they proposed a topology adaptation algorithm achieving that.

Lv *et al* [10] have presented the algorithm that restricts the flow of queries into each node so that they do not become overloaded and dynamically evolves the overlay topology so that queries flow towards the nodes that have sufficient capacity to handle them.

There are two types of connections in the model of Cooper and Garcia-Molina [6]: index and search links. Nodes may select any node they want where to establish a connection. The link types can be given probability values. If a node becomes overloaded, it has to drop one connection.

These three methods are mainly focusing on traffic distribution. Chawathe's *et al* purpose is to get the nodes which have capacity to handle a great amount of traffic to the center of the network. In Lv's *et al* research, traffic is distributed evenly and in the study of Cooper and Garcia-Molina only the load the neighbor creates and the defined amount of traffic affects to the dropping. The methods do not take into account the amount of the resources the nodes are providing or quering. The node may have a neighbor which provides lots of resources to the node and with these methods that kind of neighbor may easily drift many hops away from the node. According to our definition a good neighbor would be lost.

Iles and Deugo [8] have developed a flooding broadcast meta-protocol which is capable of describing a wide range of possible flooding broadcast network protocols. Each instance of the meta-protocol is represented with two expressions: CONN specifies the number of connections for the peer to maintain and RANK is evaluated for each existing or potential connection. By using genetic programming the automatic generation of new protocols from the meta-protocol is provided.

Wouhaybi and Campbell [16] have developed a peer-to-peer algorithm called Phenix which can construct low-diameter resilient topologies. It creates a topology where the degree of the distribution follows the power-law distribution. In this case the goodness of the network is based on the degree of the distribution. If the algorithm would take into account also the resources provided by nodes, the searching would be more effective.

3 Topology Management Algorithms

We have developed four algorithms for managing the topology: Node Selection, Node Removal, Overload Estimation and Overtaking. The algorithms use only local information the nodes have about their neighbors. The nodes save information about active neighbors and in the history information about other known nodes. The saved information are the IP address and the port number of the neighbor, the time when the neighbor has been requested and information whether the request succeeded or not. The node saves also hit information about the neighbors. A hit value tells how many resource replies the node has got from the neighbor. In that case the neighbor has had the resource. A relayed hit value is the amount of resource replies that the neighbor's neighbors have relayed to the node through this neighbor.

3.1 Node Selection Algorithm

We suppose that the initial list of the neighbors can be obtained manually by out-of-band methods or automatically using advertisement systems [15] or centralized entry point directories [7]. Node Selection Algorithm's responsibility is then to select among known nodes where to connect.

When the node joins the network again, it tries to establish connections to the neighbors it had before leaving the network. In the best case it manages to establish the connections to all the neighbors it had earlier. If the node does not manage to establish any of those connections or it otherwise needs a new neighbor, it searches the next one from the history as shown in Algorithm 3.1. First it searches the nodes which have hit values and tries to create a connection to one of those. Because the node does not want to create a connection to the same node it has just dropped, it searches only the nodes which have not been requested in a given time. If the node did not succeed in establishing a new connection, it next searches nodes based only on the time of the last request i.e the node has not tried to create a connection in a given time or at all (lacking requested information). If the node still did not successfully create a connection, it searches nodes without information for hit values or request

time. If the node does not have neighbors, then the last way to search a node is to try only those nodes in the history which have hit values. Then the node may select again a neighbor which it has just dropped.

Algorithm 3.1 (Node Selection Algorithm)
Input: Nodes $h_i s$ in the node's history $H = \{h_1, ..., h_n\}$, $time$ sets a limit for the time which older the previous connection request must be and $neighbors$ is the number of node's neighbors.
Output: Establishes a new connection
if !Connect(*true, true, time, H*) **then do**
  if !Connect(*false, true, time, H*) **then do**
    if !Connect(*false, false, time, H*) $\wedge$ $neighbors == 0$ **then do**
      Connect(*true, false, time, H*)
    end if
  end if
end if

The function Connect(*hitsNeeded, timeNeeded, time, H*) tries to create a connection to the one node in the history's nodes which meet the criteria defined in the parameters. If the value of the parameter *hitsNeeded* is true, then the function takes into account only those nodes which have hit values. If *hitsNeeded* is false, the function takes into account those nodes which do not have hit values. If the value of the parameter *timeNeeded* is true, then the function takes into account only those nodes which has not been requested for the period of the time defined in the parameter *time*. The function returns true if a connection was established succesfully.

3.2 Node Removal Algorithm

When a node wants to remove a connection to a neighbor, it selects the worst neighbor among the neighbors it currently has. The worst neighbor has the smallest goodness value. The goodness is the sum of the neighbor's hit values and relayed hits.

3.3 Overload Estimation Algorithm

There is no predefined number for the connections the node should maintain. Thus the connections are added and dropped based on the amount of traffic going through the node. The Overload Estimation Algorithm compares the calculated traffic amount to the predefined traffic limit values. There are upper and lower traffic limits which set the range where the traffic amount should be. The value of the lower traffic limit is the fraction of the upper traffic limit defined by the lower traffic limit percent. If the traffic amount is less than the lower traffic limit, the node tries to add a new connection using the Node Selection Algorithm described in Section 3.1. If the traffic amount is more than the predefined upper traffic limit, one connection is dropped by using Node Removal Algorithm described in Section 3.2. At the end, the algorithm resets the traffic amount by setting its value to zero.

3.4 Overtaking Algorithm

The Overtaking Algorithm is used to optimize the topology. The purpose of the algorithm is that the node moves closer to the nodes which provide lots of replies to it by overtaking the current neighbor. The node does not directly connect to the resource providing node but only moves closer step by step and that way makes sure that it does not lose good nodes on the path.

The algorithm works such that when a reply message arrives to the querier, it updates the hit value of the sender and then adds its local information of the relayed hits of the neighbor of the node where the node got the reply message. Then if the neighbor's hit value is bigger than 1, i.e the node has got more than one message from the neighbor, the node checks whether the neighbor has a neighbor whose proportion of the neighbor's goodness is more than the defined overtaking percent. In that case it overtakes the neighbor. For example if the overtaking percent is 60%, it means that if there is the neighbors's neighbor which has forwarded over 60% of all the reply messages the node has got from the connection, the node establishes a new connection to that node and drops the connection to the current neighbor.

Advantages of the algorithm are that the distances of the nodes, which use others' resources, are shorter than in randomly connected networks. The algorithm creates a connected network of clusters having those nodes close in the center which provide lots of resources other nodes use [1].

Algorithm 3.2 (Overtaking Algorithm)
Input: The overtaking percent $overtakingPercent$, the node's neighbor c, c's neighbors $N = \{n_1, ..., n_n\}$ and c's hit value $hitValue$.
Output: Node has overtaken a neighbor if some neighbor's neighbor is better for the node.
if $hitValue > 1$ **then do**
  biggest = *overtakingPercent*/100.0
  bestNeighbor = null
  sum = Hits(c) + RelayedHitsSum(c)
  for i = 1 **to** $\mid N \mid$ **do**
    hitValue = RelayedHits(n_i)
    proportion = *hitValue/sum*
    if $proportion \geq biggest$ **then do**
      biggest = *proportion*
      bestNeighbor = n_i
    end if

end for
if $bestNeighbor \neq null$ **then do**
if EstablishConnection(*bestNeighbor*) **then do**
DisconnectConnection(c)
end if
end if
end if

The function Hits(*node*) returns the node's hit values, the function RelayedHitsSum(*node*) returns the sum of the relayed hits of the node's neighbors and the function RelayedHits(*node*) returns the relayed hits of the node. The function EstablishConnection(*node*) returns true, if establishing a connection succeeded. The method DisconnectConnection(*node*) removes the connection to the node.

The behavior of the algorithm with example values is illustrated in the Figure 1. The node's neighbor has the hit value two and the relayed hits of neighbor's neighbors are 19 and 7. The sum of all is 28 and the percentual proportion of the neighbor's hits is 7% and neighbor's neighbors' 68% and 25%. If the overtaking percent is defined to be 80, nothing is done. If it is 60, then the node tries to establish a new connection to the neighbor's neighbor node 3 and if it succeeds the current connection to the neighbor node 2 is dropped.

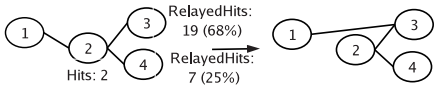

Figure 1. A situation before and after executing the overtaking algorithm.

4 Test Cases and Analysis

The algorithms were tested in the Peer-to-Peer Realm (P2PRealm) [9] simulator, which we have developed in our research project [4]. A testing environment consisted of 500 nodes and in the beginning the network was normally distributed with 997 links. The initial network was connected and in general when the network is connected the initial network does not have a large impact on the results. The initial network was randomly generated and we studied if the network could be organized with our algorithms in a more efficient, i.e. power law distributed, network.

Resources were set such that half of the nodes provided 15 resources per node and the other half provided 5 resources. So there were two groups of the nodes: those which provided lots of resources and those which provided small amount of resources. In addition to that in the both groups half of the nodes had 10 times bigger probability to query a resource than other half. The resources were identified by numbers. The resources were not transferred after the query and therefore the resource distribution was fixed throughout the simulations.

65500 randomly generated queries were sent to the network. The algorithms were tested with three different set of queries. The starting topology was the same in every test and the queries were sent in same order so the cases were repeatable. The queries used a Breadth-First Search (BFS) algorithm [11] without any TTL value so the resources were always found if the network was connected.

In the test environment every node had the same traffic limit values. The traffic amount was checked each time the simulator sent 50 resource queries to the network. The unit for the traffic limit values was also the number of resource queries. The algorithms were tested with upper traffic limit values from 100 to 600 at intervals of 25. The tests were run with the lower traffic limit percents 20, 40 and 60 and with the overtaking percent 80 and without the overtaking. 80% was selected as the overtaking percent because in the earlier studies we found that with 80% the network attained the balance [1]. The balance means that the network attains the state where no more topology changes happens.

In the analysis we studied the neighbor distribution the algorithms created, the hop numbers and the balance of the network. The simulator saved three kind of information during the tests. It saved information about the resource queries the nodes sent. At the end of each test the simulator saved the neighbor, traffic, resource reply message and resource amounts from each node. The third statistics the simulator saved was information about the amounts of topology changes in the network.

4.1 Neighbor Distribution

Table 1 presents the neighbor distributions with the different lower traffic limit percents and with and without the overtaking. With the lower traffic limit percent 20% and without the overtaking the networks were normally distributed. When the overtaking was used with the small upper traffic limit values (100-200), the topologies were stars: one node had over 440 neighbors in all the test cases until the upper traffic limit reached 225. The neighbor distributions with the overtaking were power law distributed.

With the lower traffic limit percent at 40% and without the overtaking the networks were again normally distributed. With the overtaking and the upper traffic limit 100 one node had over 400 neighbors in all the test cases so the topologies were stars. The neighbor distributions were power law distributed.

With the lower traffic limit percent at 60% without and with the overtaking the networks were normally distributed. In this case the interval, where the traffic should be, was

Table 1. Neighbor distributions with different lower traffic limit percents with and without overtaking.

Lower traffic limit %	20%		40%		60%	
Overtaking	*no*	*80%*	*no*	*80%*	*no*	*80%*
Distribution	normally	power law	normally	power law	normally	normally

smaller and overtaking did not have the same effect as with the smaller lower traffic limit percents because the amount of connection establishments and droppings was increased.

4.2 Average Hops

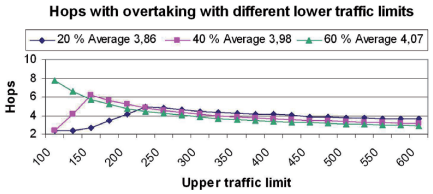

Figure 2. Average hops with different lower traffic limit percents with overtaking.

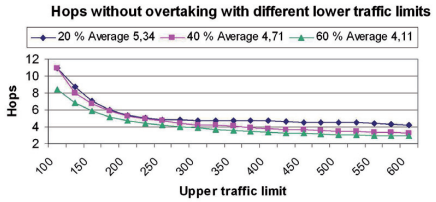

Figure 3. Average hops with different lower traffic limit percents without overtaking.

Figures 2 and 3 contain the average hops of the resource messages with the different upper traffic limit values and lower traffic limit percents without and with the overtaking. When using the overtaking and the lower traffic limit percent at 20% the resources were found closer and the number of hops was more than one less than without the overtaking algorithm. When the lower traffic limit percent increased, the difference between the hop values decreased. Because every node had the same traffic limit, it had too big influence on the topology. When the lower traffic limit percent increased, the interval, in which the traffic should be, became smaller. So the amount of connection establishments

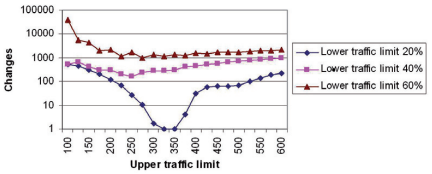

Figure 4. The amount of changes with lower traffic limit percents and with different upper traffic limit values and without overtaking.

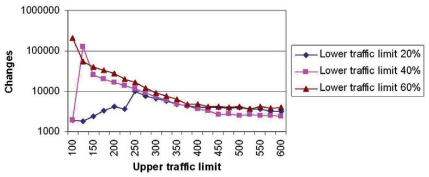

Figure 5. The amount of changes with lower traffic limit percents and with different upper traffic limit values and with overtaking.

and droppings were also increased when the nodes were trying to keep the traffic inside the limits and the overtaking did not have a chance to affect the topology.

The overtaking percent parameter describes how much better the neighbor's neighbor has to be than other neighbors so that the overtaking occurs. By defining a small overtaking percent the nodes overtake easily and the topology does not attain the balance. A high value forms a topology with a balanced state. When the overtaking algorithm was tested without any traffic limits, the generated network was power law distributed which means that it is fault-tolerant and the distances of the nodes in the network are short.

4.3 Network Balance

When using the lower traffic limit percents 20% and 40% without the overtaking the networks attained the balance. By 60% the networks gained the balance when the upper

Second International Conference on
Internet and Web Applications and Services (ICIW'07)
0-7695-2844-9/07 $20.00 © 2007 IEEE

traffic limit value was more than 200.

When the overtaking was used and the lower traffic limit percent was 20%, the networks were totally in balance until the upper traffic limit value reached 225. After that there were a few changes. With the lower traffic limit percent at 40% the networks were balanced when the upper traffic limit was 100 and again since the upper traffic limit value was 400 or above. With the lower traffic limit percent at 60% after the upper traffic limit value 325 there were only a few changes and the networks gained balance.

Figures 4 and 5 include the topology changes occured in the tests. With the lower traffic limit percent at 20% and without the overtaking the amount of the changes in the networks was smallest. 40% gave also the small amount of changes compared to 60% where the amount of changes was much bigger. So the amount of changes increased with the increasing lower traffic limit percent.

With the overtaking, the lower traffic limit percent at 60% created lots of changes with the upper traffic limit value 100, i.e. the nodes in turn established and dropped connections. The same effect was with the lower traffic limit percent at 40% when the upper traffic limit was 125. When the upper traffic limit was less than 275, there were more overtakings with the lower traffic limit percents 40% and 60% than with 20%, since that with 20% more overtakings happened. Also with the upper traffic limit value 100-350 there were more removings with the lower traffic limit percent 40% and after that with 20%. The amount of the additions increased when the lower traffic limit percent increased.

5 Conclusion

We presented four topology management algorithms which use the goodness of the nodes when deciding on where to connect or which neighbor should be dropped. We studied the topology algorithms only with some parameters and some values. The lower traffic limit percent 40% with the overtaking and with the upper traffic limit values 350 or above were the best combinations. With those values the amount of the changes in the network was small, the topology got balanced, the neighbor distribution was power law distributed and the number of the hops was the second smallest.

Now we have started to study the topology construction using neural networks like we have done with the NeuroSearch algorithm [14] for the resource discovery. In the future it is thus easier to study the different combinations of the input parameters about the topology and see if the parameter has some impact when deciding where to connect or which neighbor should be dropped. In the future also traffic limits for the nodes are set so that those represent the distribution of the network bandwidths in the current P2P networks. Later the algorithms can be deployed in the Chedar P2P network [2].

References

[1] A. Auvinen. Topology management algorithms in chedar peer-to-peer platform. Master's thesis, University of Jyväskylä, February 2004.

[2] A. Auvinen, M. Vapa, M. Weber, N. Kotilainen, and J. Vuori. Chedar: Peer-to-peer middleware. In *20th International Parallel and Distributed Processing Symposium*, 2006.

[3] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *ACM SIGCOMM*, 2003.

[4] CheeseFactory. http://tisu.it.jyu.fi/cheesefactory.

[5] T. Condie, S. D. Kamvar, and H. Garcia-Molina. Adaptive peer-to-peer topologies. In *Proceedings of the Fourth International Conference on Peer-to-Peer Computing (P2P'04)*, 2004.

[6] B. F. Cooper and H. Garcia-Molina. Ad hoc, self-supervising peer-to-peer search networks. *ACM Transactions on Information Systems*, 23(2):169–200, 2005.

[7] Gnutellahosts. http://www.gnutellahosts.com/.

[8] M. Iles and D. Deugo. A search for routing strategies in a peer-to-peer network using genetic programming. In *Proceedings of 21st IEEE Symposium on Reliable Distributed Systems*, pages 341–346, 2002.

[9] N. Kotilainen, M. Vapa, A. Auvinen, T. Keltanen, and J. Vuori. P2prealm - peer-to-peer network simulator. In *11th International Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks*, pages 93–99, 2006.

[10] Q. Lv, S. Ratnasamy, and S. Shenker. Can heterogeneity make gnutella scalable? In *Electronic Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, 2002.

[11] N. A. Lynch. *Distributed Algorithms*. Morgan Kauffmann Publishers, 1996.

[12] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter p2p networks. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS'01)*, 2002.

[13] M. K. Ramanathan, V. Kalogeraki, and J. Pruyne. Finding good peers in peer-to-peer networks. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)*, 2002.

[14] M. Vapa, N. Kotilainen, A. Auvinen, H. Kainulainen, and J. Vuori. Resource discovery in p2p networks using evolutionary neural networks. In *International Conference on Advances in Intelligent Systems Theory and Applications (AISTA 2004)*, November 2004.

[15] M. Weber, J. Vuori, and M. Vapa. Advertising peer-to-peer networks over the internet. *Radiotekhnika*, 133:162–170, 2003.

[16] R. H. Wouhaybi and A. T. Campbell. Phenix: Supporting resilient low-diameter peer-to-peer topologies. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004)*, 2004.

PVI

TOPOLOGY MANAGEMENT IN UNSTRUCTURED P2P NETWORKS USING NEURAL NETWORKS

by

Annemari Auvinen, Teemu Keltanen & Mikko Vapa 2007

IEEE Congress on Evolutionary Computation

Topology Management in Unstructured P2P Networks Using Neural Networks

Annemari Auvinen, Teemu Keltanen and Mikko Vapa

Abstract— **Resource discovery is an essential problem in peer-to-peer networks since there is no centralized index in which to look for information about resources. In a pure P2P network peers act as servers and clients at the same time and in the Gnutella network for example, peers know only their neighbors. In addition to developing different kinds of resource discovery algorithms, one approach is to study the different topologies or structures of the P2P network. In many cases topology management is based on either technical characteristics of the peers or their interests based on the previous resource queries. In this paper, we propose a topology management algorithm which does not predetermine favorable values of the characteristics of the peers. The decision whether to connect to a certain peer is done by a neural network, which is trained with an evolutionary algorithm. Characteristics, which are to be taken into account, can be determined by the inputs of the neural network.**

I. INTRODUCTION

Peer-to-peer technologies have received a lot of publicity lately mainly because of Kazaa and other peer-to-peer file sharing systems. Other resources, for example CPU time and storage space, can also be shared in a peer-to-peer network. In the P2P network every peer, i.e. a node may provide resources to other nodes and consume the resources other nodes are providing. This means that a node may serve both as a server and as a client. The P2P network can be structured or unstructured. In an unstructured network, like Gnutella and in our study, a node's place in the network is not pre-defined like it is in a structured network. A node may join the network by establishing a connection to another node in the P2P network. Resource discovery is not very efficient in that kind of network but maintaining the topology does not produce extra work.

Topology management algorithms affect the network's overlay topology by making the network more scalable and effective for resource discovery. Nodes want to stay connected to the network and find resources efficiently without using too much of their own capacity for being in the network. It is suitable for example that the nodes connecting with a modem are in the edge of the network and the nodes capable of handling a lot of traffic are in the center. With topology management algorithms the network can be kept connected, i.e. there are no clusters, which are not connected to each other.

Manuscript received March 15, 2007. This work was supported in part by the Graduate School in Electronics, Telecommunications and Automation (GETA).

A. Auvinen, T. Keltanen, and M. Vapa are with Department of Mathematical Information Technology, University of Jyväskylä, Finland (e-mail: firstname.lastname@jyu.fi).

In our research project we have developed four algorithms for topology management [2]. In this paper we propose a different kind of solution. We used neural networks to create the algorithm by machine instead of constructing algorithm by humane hand. The neural network gets the characteristics of the P2P network as inputs and as an output the decision whether to create a connection to a certain node.

The paper is organized as follows. We present related work in Section II. Section III describes the developed NeuroTopology algorithm for managing the topology of unstructured P2P networks. The optimization process is described in Section IV. Section V presents the test case used in the study and Section VI the analysis of the simulation results. The paper is concluded in Section VII.

II. RELATED WORK

Much research has been done regarding the efficiency of a pure unstructured P2P network by changing the structure or the topology of the network. One way to approach the problem is to organize the nodes so that they form up clusters according to their interests of the previous resource queries (interest-based locality). Ramanathan et al. [9] searched for good neighbors by connecting to those peers that repeatedly give good results and disconnecting those peers that give poor results. As a result, peers have few neighbors and they form clusters where resources of the same interest are close to each other. Because there are only few connections between clusters, this is not efficient if peers are interested in resources from multiple subjects or suddenly change their interest. Sripanidkulchai et al. [11] formed shortcuts in the Gnutella network to peers that, based on the previous queries, are interested in resources of the same topic. The shortcuts form their own logical network on top of the Gnutella topology, which is therefore not changed. When resources are searched, the shortcut topology is used first and if resources are not found, the Gnutella topology is used traditionally. Condie et al. [3] developed a protocol that connects to peers that will probably give good results in the future. This is based on a score assigned to peers according to their previous answers. In addition, the reliability of the peer is also scored in order to reduce the effect of freeriders and malicious peers distributing corrupted files. In the study, poor peers moved to the edge of the network and other peers formed clusters according to their interest of resources. Crespo and Garcia-Molina [6] have suggested Semantic Overlay Network or SON, where joining

1-4244-1340-0/07/$25.00 ©2007 IEEE

peers connect into one or more logical clusters based on the content of peers' resources and the available SON-networks in the P2P network.

Another way to approach the topology problem is to take into account only the technical characteristics of the peers e.g. bandwidth or traffic amounts. These techniques do not consider the query history or the quality of the resources. Cooper and Garcia-Molina [4] made the overloaded peer to disconnect neighbors that are burdening the link most or neighbors that overrun some predetermined traffic limit. In the study, it was also possible to concentrate on favoring efficient peers instead of helping overloaded peers.

The studies presented above are concentrating on a single problem or characteristic of the P2P network. As a result, techniques that use the interest-based clustering are forming topologies where popular nodes are under lot of strain. Similar to this, in the techniques where only technical characteristics are taken into account, one might discriminate the nodes that have good resources and weaken query times. Sakarayan and Unger [10] measured the evolution of a P2P topology when it was affected by traffic overloading and interest-based clustering. During resource searching, information about peers is gathered into messages and therefore peers know more peers than just their neighbors. This information consists of data about resources owned by the peers, addresses and how long the message was in a peer. The algorithm that reacts to traffic overloading wakes up when the queue of incoming messages exceeds some limit and sends warnings to nearby peers. Peers re-route messages to avoid traffic. The algorithm that affect interest-based locality wakes up when access times or the distance that messages pass exceed some limit. According to this study, locally operating algorithms can affect the efficiency of the network also in the scale of the Internet.

Iles and Deugo [7] developed a meta-protocol that works with the BFS algorithm. The evolution of the meta-protocol is guided by genetic programming and it produces P2P protocols as implementations. These protocols define the topology of the P2P network. Two expressions affect the evolution of the meta-protocol: CONN, which is the amount of neighbors that is desirable for each peer to have and RANK, which is a comparison indicator for each possible neighbor. The CONN-expression uses information like current neighbor amount and statistical information about the traffic amount of the peer. The RANK-expression uses mostly information that is related to the previous queries and their success. Measuring the success of the evolutionary process is done with the fitness function:

$fitness = searches_{successful} + 0.5 * searches_{timed-out}$. In the study it was noticed, that the protocol used by Gnutella is in many cases optimal and that P2P networks, where bandwidths are small, form clusters still remaining connected. It was also noticed, that genetic programming is an effective search technique for the Gnutella networks and it produces peers that are adjustable in a varying environment. The problem of the study was small network with only 30 peers.

In this study, we do not predetermine any values of the characteristics that are desirable for the P2P network. Instead we try to find out whether the evolutionary neural networks are able to form efficient P2P topologies for resource queries when we determine the characteristics that the neural network should take into account. These characteristics are given to the neural network as inputs and can be e.g. bandwidth or information about the previous resource queries. As a result, we hope to gain a dynamic P2P network, where the topology takes shape in interaction with the resource discovery algorithm.

III. NeuroTopology Algorithm

NeuroTopology algorithm affects the overlay of the peer-to-peer network by using a neural network for the topology construction. NeuroTopology was implemented as a plugin for the P2PRealm simulator [8].

The idea is that every peer has a neural network to make decisions about establishing new connections in the P2P network (Fig. 1). The information that the neural network needs, is gathered during resource queries. NeuroTopology algorithm is executed in every peer after a predefined amount of resource queries. The NeuroTopology is described in Algorithm 3.1:

Algorithm 3.1.

Input: The node's u neighbor candidates are $N = \{s_1, s_2, ..., s_k, w_1, ...w_h, L\}$, where $k \le n-1, h \le n-2$, s_i is the node's neighbor, w_i is the node's neighbor's neighbor and L are the nodes, from which the node has received resource replies. T is the number of topology packets i.e., the amount of traffic topology requests produced. T_a is the number of topology replies i.e., the amount of traffic produced when establishing a new connection.

Output: Connections to neighbors.

For all $b_i \in N$

1. The input parameters for the neural network are set according to the information about neighbor candidate b_i.
2. Calculate the output for the candidate b_i.
3. If the output is 1 then
   - 3.1 Node u requests candidate b_i to be its neighbor. The number of topology packets is incremented by one: $t = t + 1$.
   - 3.2 The input parameters for the neural network are set according to the information about node u.
   - 3.3 The output for node u is calculated.
   - 3.4 If the output is 1, the connection between nodes u and b_i is confirmed and the number of topology replies is incremented by one $t_a = t_a + 1$.
   - 3.5 If the output is 0, node b_i does not accept the request. The number of topology packets is incremented by one $t = t + 1$. If b_i is a neighbor of node u, the connection to node u is dropped.
4. If the output is 0 and b_i is a neighbor, the connection to node b_i is dropped.

The algorithm goes through all neighbor candidates. A connection to a candidate is not established one-sided but also the candidate evaluates with the same neural network whether it wants to establish a connection to the requesting node.

The input parameters for the neural network are:

- *Bias* is the bias term and has value 1.
- *CurrentNeighborsAmount* is the number of the node's neighbors.
- *ToNeighborsAmounts* is the number of the node's candidate neighbor's neighbors.
- *RepliesFromCandidates* is the number of resource replies received from a candidate neighbor.
- *RelayedRepliesFromCandidates* is the number of resource replies which a candidate neighbor has relayed to the node.
- *TrafficMeter* is a counter, which calculates the amount of resource reply messages going through a candidate neighbor.
- *TrafficLimit* simulates the bandwidth of a candidate neighbor. If *TrafficMeter* value is bigger than *Trafficlimit*, the candidate neighbor will not reply to resource requests.

All input parameters should be scaled in [0,1] so that any parameter will not be dominant, thus slowing down the optimization. *RepliesFromCancidate*, *TrafficMeter* and *RelayedRepliesFromCandidate* can have value of zero so those are scaled with the function

$$f(x) = \frac{1}{x+1}.$$

ToNeighborsAmount, CurrentNeighborsAmount and *TrafficLimit* are scaled with the function

$$f(x) = \frac{1}{x}.$$

NeuroTopology uses a neural network with two hidden layers. There are 15 nodes (neurons) and a bias in the first hidden layer and 3 nodes and a bias in the second. The activation function in the hidden layers is the hyperbolic tangent (tanh)

$$t(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

The activation function in the output node is the threshold function

$$s(x) = \begin{cases} 0, x < 0 \\ 1, x \geq 0 \end{cases}.$$

The output of the neural network is attained by combining the functions presented above and output values of the neurons' with the formula

$$O = s(1 + \sum_{k=1}^{3} w_{3k} t(1 + \sum_{j=1}^{15} w_{2j} t(\sum_{i=1}^{7} w_{1i} f(I_i)))),$$

where I_i is the value of input parameter *i* and w_{xy} is the neural network weights on layer *x* in position *y*.

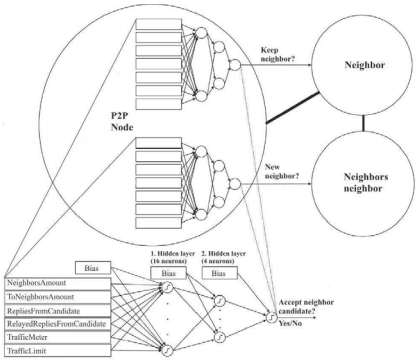

Fig. 1: The neighbors of the peer are determined with the neural network that receives information about neighbor candidates in its inputs.

IV. Neural Network Optimization

Before NeuroTopology can be used for managing the topology, the weights of the neural network have to be optimized. We used evolutionary computing to optimize the weights.

The fitness of the used neural network is defined based on the amount of traffic in the P2P network. Each query *j* (both resource and topology queries) is scored for the neural network *h* and the fitness is sum of scores F_j.

$$fitness_h = \sum_{j=1}^{n} F_j.$$

The scores are defined as follows:

$$F_j = \begin{cases} Rr - (p + t + Tt_a), & if\ Rr - (p + t + Tt_a) \geq 1 \\ \dfrac{1}{p + t + Tt_a - Rr}, & otherwise \end{cases} \quad (4.1)$$

$$F_j = \begin{cases} 2Rr - (p + t + Tt_a), & if\ Rr - (p + t + Tt_a) \geq 1\ and\ r \geq G \\ Rr + (t + t_a - p), & if\ Rr - (p + t + Tt_a) \geq 1\ and\ r < G \\ \dfrac{1}{p + t + Tt_a - Rr}, & otherwise \end{cases} \quad (4.2)$$

Where *p* is the number of resource queries, *r* is the number of resource replies, *R* is a constant which affects the impact of the replies and the sent packets on the scoring, *t* is the number of packets the topology query used, t_a is the number of new connections and *T* is a constant which affects the impact of new connections on the scoring. *G* is the goal for the number of the resources the resource query should locate.

When measuring the performance of the P2P network in the generalization environment (see Section V), the formula 4.1 is used. If there are enough replies for the queries, the neural network will receive better fitness values by decreasing the amount of packets: $F_j = Rr - (p + t + Tt_a)$. If there are not

enough resources in relation to the sent packets, the neural network will attain better fitness values by increasing the amount of packets: $F_j = \frac{1}{p + t + Tt_a - Rr}$.

When training the neural network, formula 4.2 is used. If the network locates the predefined amount of resources, the score from replies is doubled. Then the neural network can attain better fitness values by decreasing the number of packets and topology packets. Especially, the number of new connections is encouraged to be decreased with $F_j = 2Rr - (p + t + Tt_a)$, because the current topology already has some desired properties. If there are enough resource replies when the sent packets are taken into account (also the topology packets) but the goal is not achieved, the neural network will attain better fitness values by increasing the amount of topology packets: $F_j = Rr + (t + t_a - p)$. If there are not enough located resources in proportion to sent packets, the neural network will attain better fitness values by increasing the amount of query and topology packets: $F_j = \frac{1}{p + t + Tt_a - Rr}$.

The optimization process had an initial population of 24 neural networks whose weights were randomly defined from the [-0.2, 0.2] interval. Next, every neural network was tested in the peer-to-peer simulation environment and the fitness value was calculated. When all neural networks had been tested, the 12 best were chosen for mutation and used to breed the new generation of neural networks. As a result, 24 neural networks were available to be tested at the next generation.

The mutation was based on the Gaussian random variation and used the weighted mutation parameter to improve the adaptability of the evolutionary search. The random variation function was similar to the one used by Fogel and Chellapilla in their research [5] and is given as:

$$\sigma_i'(j) = \sigma_i(j)\exp(\tau N_j(0,1)),\ j = 1,\ldots,N_w,$$

$$w_i'(j) = w_i(j) + \sigma_i'(j)N_j(0,1),\ j = 1,\ldots,N_w,$$

where $N_w =$ is the total number of weights and bias terms in the neural network, $\tau = \frac{1}{\sqrt{2\sqrt{N_w}}}$, $N_j(0,1)$ is a standard Gaussian random variable resampled for every j, σ is the self-adaptive parameter vector for defining the step size for finding the new weight and $w_i'(j)$ is the new weight value. This method can be seen as a memetic algorithm because when the self-adaptive parameter σ_i is small, the optimization is local.

V. Simulation Environment

As a peer-to-peer simulation environment, we used the Peer-to-Peer Realm (P2PRealm) network simulator [8] which was originally developed for studying a resource discovery algorithm based on neural networks [12]. In this research, we added a neural network guided topology management algorithm to P2PRealm.

In the test case we used a P2P network that had 100 peers. Resources were power-law distributed so that peers with small peer numbers had more resources than others. The amount of resources was 491 and there were 25 different kinds of resources. Each peer had a traffic limit which determined the maximum amount of resource packets during 10 resource queries. The traffic limits were:

- Nodes 0-24, traffic limit=30
- Nodes 25-49, traffic limit=15
- Nodes 50-74, traffic limit=10
- Nodes 75-99, traffic limit=6

The resource discovery algorithm had a target of finding 50% of the desired resources. The goal of finding half of the available resource instances was set to demonstrate the algorithm's ability to balance on a predetermined quality of service level and not just on locating all resource instances or one resource instance. We used breadth-first search (BFS), highest degree search (HDS) and random walker (RW) as resource discovery algorithms.

The test case is divided into the training environment, where the neural networks are trained and the generalization environment, where the performance of the best neural network is measured in a new but similar environment indicating the neural network's ability to generalize. When the performance starts to decrease in the generalization environment, the training can be stopped. At that point the neural network is adapting only to the training set if the training process is continued. In the training set each generation is started with a grid topology P2P network and follows the algorithm:

1. Do rounds 20 times
    a. 10 random peers execute resource queries
    b. Execute NeuroTopology algorithm in every peer using information from resource queries
2. Execute 10 resource queries in the P2P network
3. Calculate the fitness for the neural network using information from step 2

The generalization set is the same as the training set, except that resource queries were executed by every peer in the P2P network. In order to keep traffic limits functioning properly, the traffic meters were reset after every 10 queries. Another difference is in the use of the fitness function (Section IV). In the training set the parameter R was 300 and in the generalization set the value is 50. The parameter R can be considered as a reward for founding resources and the value 300 produces consistently well-trained neural networks. R was selected to be large enough to guide the training process towards neural networks that locate enough resources, but also small enough to prevent nodes from connecting to all the neighbors that have wanted resource during some random query. In the generalization set the value 50 was chosen as a standard value for comparing the neural networks that were trained with different kind of attribute values. The value of parameter T was 5 in both environments. This penalty term simulates the amount of TCP-packets when establishing new connections.

The training of the neural networks was done using the HDS algorithm and the amount of generations was 5000. The results of the test case are represented in Fig. 2-6. In the training set there is no significant improvement in the fitness value after generation 400 but some optimization still took place because in the generalization set the fitness is not converging until generation 3500. In the generalization set the HDS algorithm finds desired resources (845 of them) most of the time after generation 400, but the amount of packets is decreasing until generation 3500. Also the topology packets, the topology changes and the amount of failed queries remain relatively stable after generation 3500. "Failed queries" represents the amount of nodes that do not reach their target of 50% found resources. Thus, it was possible to train the neural networks in a computationally easier environment and to use the trained networks in a more demanding environment.

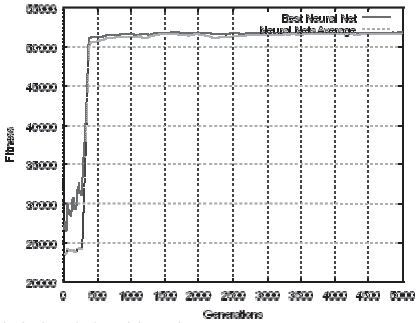

Fig. 2: Fitness in the training environment.

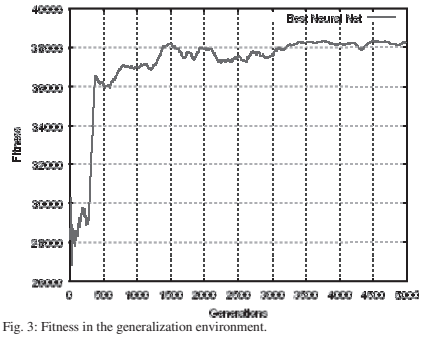

Fig. 3: Fitness in the generalization environment.

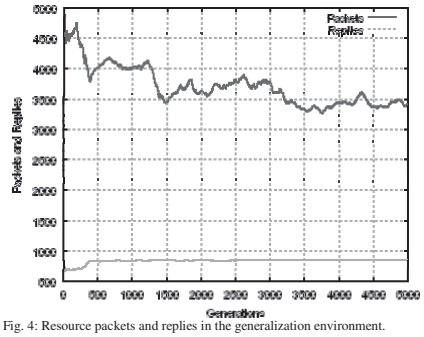

Fig. 4: Resource packets and replies in the generalization environment.

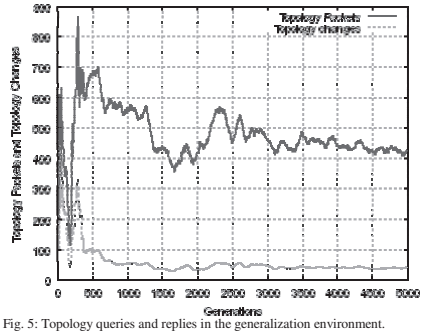

Fig. 5: Topology queries and replies in the generalization environment.

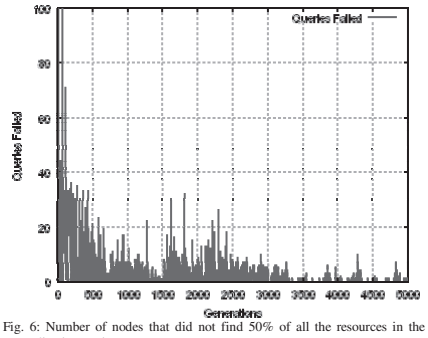

Fig. 6: Number of nodes that did not find 50% of all the resources in the generalization environment.

VI. Simulation Results

To evaluate the efficiency of the topology that is produced with the trained NeuroTopology algorithm, in addition to the grid topology, we generated a power-law topology and a random graph topology for comparison. The power-law

topology was generated using the Barabási-Albert model and the random graph topology using the Erdös model [1]. Parameters for the traffic limits, resource distribution and fitness function are the same as in the generalization set of the training neural network. The results of the networks where peers are searching resources with highest degree search (HDS), random walker (RW) and breadth first search with TTL value 3 (BFS) in the above mentioned topologies, are documented in Table 1. By calculating the ratio between the located resources and the used query packets, we can determine the efficiency of the algorithms. Walker algorithms (HDS and RW) perform best in the power-law topology finding nearly all resources with an efficiency of 0.23 and 0.18 respectively. The best topology for the BFS algorithm is the random graph, where 470 of 845 desired resources were found with an efficiency of 0.14. Only 13 peers reached the target of finding 50% of all the resources.

Next, we analyze the effect of NeuroTopology with nine different scenarios: three different starting topologies using three different resource algorithms. Every peer executes resource queries and then executes the NeuroTopology algorithm that was trained using the HDS algorithm. This procedure is done 20 times and the results are in Table 2. In the efficiency columns the first one is counted with topology packets and the second one without them. When comparing the fitness values (rewarding every resource with 50 points) between Tables 1 and 2, we can see significant improvement in most of the cases. The power-law topology is the hardest one to improve. For example the HDS algorithm finds roughly the same amount of resources in the power-law P2P network and in the NeuroTopology generated P2P network but uses 470 less packets in the latter one. Nevertheless, when considering the traffic used by the topology management, the fitness value remains roughly the same. A general observation from the results is that the NeuroTopology trained using the HDS algorithm is able to improve the efficiency of the walker algorithms regardless of the starting topology. The training was done using the HDS algorithm, which prefers nodes with high neighbor amount. The BFS algorithm prefers P2P networks where the neighbor distribution is more uniform. Due to the different nature of these algorithms, the neural network has not learnt to generate a topology, which improves the efficiency of both BFS and HDS at the same time.

Values in Table 2 are average values of 20 rounds so they do not give us information about the convergence of the P2P network. A good topology algorithm would change the inefficient grid topology on the early rounds and limit the changes when the efficient topology has been reached. An example is in Fig 7 and Fig. 8. NeuroTopology started from the grid topology where the HDS algorithm was used. The P2P network has converged after 4 rounds of resource queries and topology changes. The topology after 20 rounds is presented in Fig 9.

VII. Conclusion

NeuroTopology has proved to be an adaptable algorithm for the P2P network topology management. P2P topologies generated by NeuroTopology are significantly more efficient than grid, random or power-law topologies. Nevertheless, managing topology produces traffic. One has to case-specifically consider, how worthy it is to find resources with less query packets. For example, using the random walker in the power-law topology without NeuroTopology uses 12% more resource packets to find roughly the same amount of resources compared to using NeuroTopology. Adding the topology management traffic to the equation, the efficiency is roughly the same. Nevertheless, the results are encouraging and further research includes testing the algorithm in larger P2P networks.

TABLE I
Efficiencies of Resource Algorithms in Static P2P Topologies

Algorithm	Topology	Fitness	Packets	Resources	Failed Queries	Hops	Efficiency
HDS	Grid	30059	4791	697	24	47.93	0.145
RW	Grid	30449	4501	699	24	45.08	0.155
BFS	Grid	10302	2598	258	99	2.92	0.099
HDS	Power	38216	3634	837	6	36.34	0.230
RW	Power	36193	4507	814	7	45.07	0.180
BFS	Power	18293	4707	460	71	2.86	0.097
HDS	Random	28209	3891	642	45	38.97	0.164
RW	Random	26047	3603	593	50	36.07	0.164
BFS	Random	19340	3350	470	87	2.96	0.140

TABLE 2
EFFICIENCIES OF RESOURCE ALGORITHMS WHEN USING NEUROTOPOLOGY

Algorithm	Topology	Fitness	Improvement in Fitness	Packets	Resources	Failed Queries	Topology Packets	Topology Changes	Hops	Efficiency	Efficiency (only resource packets)
HDS	Grid	37502	24.76 %	3549	836	1	414	67	35.50	0.207	0.236
RW	Grid	34522	13.38 %	4272	795	10	486	94	42.72	0.164	0.186
BFS	Grid	22497	118.38 %	5833	589	57	510	122	2.93	0.091	0.101
HDS	Power	38130	-0.23 %	3164	838	1	366	48	31.64	0.234	0.265
RW	Power	37216	2.83 %	4010	837	1	384	48	40.10	0.188	0.209
BFS	Power	20768	13.53 %	5923	553	62	464	99	2.90	0.085	0.093
HDS	Random	37505	32.95 %	3409	834	2	456	66	34.10	0.212	0.245
RW	Random	35496	36.28 %	4382	815	6	497	75	43.83	0.165	0.186
BFS	Random	23382	20.90 %	5728	605	56	545	119	2.94	0.095	0.106

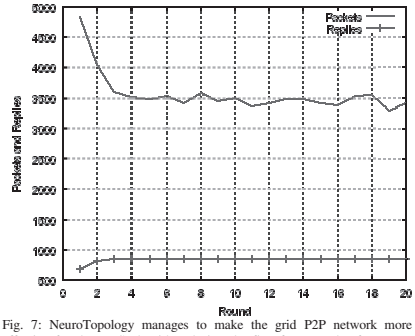

Fig. 7: NeuroTopology manages to make the grid P2P network more effective for the HDS algorithm during the first four rounds of resource querying and topology changing.

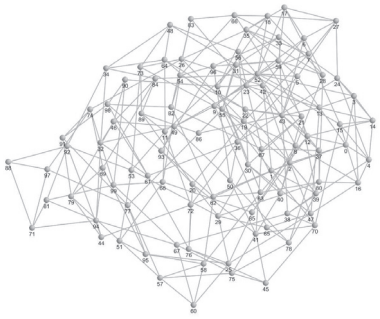

Fig. 9: End result P2P topology after 20 rounds when started from the grid topology.

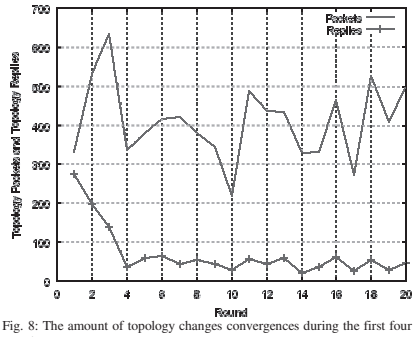

Fig. 8: The amount of topology changes convergences during the first four rounds.

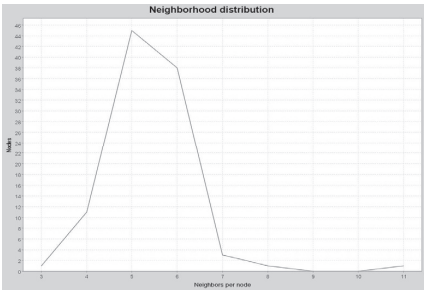

Fig. 10: Neighborhood distribution of the topology in Fig. 9.

REFERENCES

[1] E.M. Airoldi and K.M. Carley. Sampling Algorithms for Pure Network Topologies: a Study on the Stability and the Separability of Metric Embeddings. SIGKDD Explor. Newsl., 7(2):13–22, 2005.

[2] A. Auvinen, M. Vapa, M. Weber, N. Kotilainen, J. Vuori. New Topology Management Algorithms for Unstuctured P2P Networks. Second International Conference on Internet and Web Applications and Services, May 2007.

[3] T. Condie, S. Kamvar and H. Garcia-Molina. Adaptive Peer-to-Peer Topologies. In Proceedings of the Fourth IEEE International Conference on Peer-To-Peer Computing, 2004.

[4] B.F. Cooper and H. Garcia-Molina. Ad hoc, Self-Supervising Peer-to-Peer Search Networks. Technical report, 2003.

[5] K. Chellapilla and D. Fogel. Evolving neural networks to play checkers without relying on expert knowledge. IEEE Trans. on Neural Networks, 10 (6), pp. 1382-1391, 1999.

[6] A. Crespo and H. Garcia-Molina. Semantic Overlay Networks for P2P Systems. Technical report, 2002.

[7] M. Iles and D. Deugo. Adaptive Resource Location in a Peer-to-Peer Network. In The 16th International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems, July 2003.

[8] N. Kotilainen, M. Vapa, T. Keltanen, A. Auvinen and J. Vuori. P2Prealm – Peer-to-Peer Network Simulator. In Proceedings of the 11th IEEE International Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks, 2006.

[9] M.K. Ramanathan, V. Kalogeraki and J. Pruyne. Finding Good Peers in Peer-to-Peer Networks. In Proceedings of IEEE International Parallel and Distributed Computing Symposium, April 2002.

[10] G. Sakaryan and H. Unger. Influence of the Decentralized Algorithms on Topology Evolution in P2P Distributed Networks. In Proceedings of Design, Analysis, and Simulation of Distributed Systems (DASD 2003), 2003.

[11] K. Sripanidkulchai, B. Maggs and H. Zhang. Efficient Content Location Using Interest-based Locality in Peer-to-Peer Systems. In Proceedings of Infocom, 2003.

[12] M. Vapa, N. Kotilainen, A. Auvinen, H. Kainulainen, and J. Vuori. Resource Discovery in P2P Networks Using Evolutionary Neural Networks. In International Conference on Advances in Intelligent Systems - Theory and Applications (AISTA 2004), November 2004.

PVII

OPTIMAL RESOURCE DISCOVERY PATHS OF GNUTELLA2

by

Mikko Vapa, Annemari Auvinen, Yevgeniy Ivanchenko, Niko Kotilainen & Jarkko Vuori 2008

IEEE 22nd International Conference on Advanced Information Networking and Applications

22nd International Conference on Advanced Information Networking and Applications

Optimal Resource Discovery Paths of Gnutella2

Mikko Vapa, Annemari Auvinen, Yevgeniy Ivanchenko, Niko Kotilainen and Jarkko Vuori
Department of Mathematical Information Technology
P.O.Box 35 (Agora), 40014 University of Jyväskylä, Finland
firstname.lastname@jyu.fi

Abstract

This paper shows that the performance of peer-to-peer resource discovery algorithms is upper bounded by a k-Steiner minimum tree and proposes an algorithm locating near-optimal query paths for the peer-to-peer resource discovery problem. Global knowledge of the topology and the resources from the peer-to-peer network are required as an input to the algorithm. The algorithm provides an objective measure for defining how good local search algorithms are. The performance is evaluated in simulated peer-to-peer scenarios and in the measured Gnutella2 P2P network topology with four local search algorithms: breadth-first search, self-avoiding random walker, highest degree search and Dynamic Query Protocol.

Keywords - peer-to-peer; P2P; resource discovery; k-Steiner minimum tree; optimal paths; Gnutella2;

1. Introduction

Peer-to-Peer networks (P2P) are distributed systems, which consist of resource sharing processes. A typical use case for a P2P network is the file sharing, where users can share the files located in their computers to other users in the network. The shared files can be found by executing a query, which locates the instances of the queried file and returns the information for downloading them. Thus the processes connected to the P2P network act both as a client and a server consuming and offering resources.

Locating resources is an essential problem in peer-to-peer networks, because there is no centralized point or index from which the information about the resources could be found. Therefore developing efficient resource discovery algorithms is crucial.

In the *peer-to-peer resource discovery problem*[1], any node can possess resources and query resources from other nodes in the network. The problem consists of network with nodes, links and resources. Resources are identified by unique IDs and nodes may contain any number of resources. One node knows only the resources it is currently hosting. Any node in the network can start a query, which means that some of the links are traversed based on the local forwarding decisions in the network. Whenever the query reaches a node which has the resource, the node replies. The goal is to locate a predetermined amount of resource instances with a given ID using as few query packets as possible.

The problem can be solved using a distributed search algorithm, in which the querying node sends a query to its neighbors, who in turn forward the query further until the algorithm stops. Whenever a queried resource is located, a reply message is relayed back using the query path. Such an algorithm works optimally if the query is forwarded only to the neighbors, who either provide the queried resource, or can provide a minimal cost path to a set of nodes containing the queried resource.

With the global information about the topology and the resources the problem can be formulated as a Steiner minimum tree problem in graphs [19], giving an upper bound for the performance of resource discovery algorithms. In the Steiner tree problem, given a graph containing the vertices and the edges and a terminal set containing the vertices, the task is to compute a spanning tree containing all vertices in the terminal set. Steiner minimum tree is the tree with minimum length of all such spanning trees. The terminal set contains the node, which starts the query and the matching resource instances that can be located in the network.

The peer-to-peer resource discovery problem can be mapped to the Steiner minimum tree problem only if the number of needed resource instances is the same as the size of the terminal set minus one (because the query originator also needs to be in the set). However, it is often sufficient to locate for example half of the available resources, because the query originator may use, e.g. download, only some of the located resources. Also locating only one instance is not always a feasible solution, because there can be many different resources matching the query keyword, but only some of them represent the resource the query initiator is interested in.

[1] Note that peer-to-peer resource discovery problem differs from the resource discovery problem described in [4] because only one node needs to discover the other nodes containing resources. Peer-to-peer resource discovery problem has also other names such as the resource-location problem [12].

1550-445X/08 $25.00 © 2008 IEEE
DOI 10.1109/AINA.2008.89

Usually locating only a portion of resource instances reduces the amount of query traffic significantly. This is beneficial especially in mobile and wireless peer-to-peer networks, where the use of battery power and therefore the amount of forwarded query packets should be minimized. Also, as was seen in the first version of Gnutella [18] the scalability of the peer-to-peer network weakens in wired networks when the resource discovery algorithm is not properly designed.

In this paper we show that the peer-to-peer resource discovery problem with global knowledge is identical to the Steiner tree problem when all resources need to be found and therefore can be used to find optimal paths for the peer-to-peer resource discovery problem. Also, to enable only a part of the resources to be discovered we modify the original Steiner minimum tree problem to *Rooted k-Steiner minimum tree* problem, where *k* represents the number of resources that needs to be located and present an approximation algorithm for solving the problem.

The approximation is needed because *k*-Steiner minimum tree problem is known to be *NP*-hard and thus no efficient polynomial algorithm exists for practically solving the Steiner minimum tree problem in large graphs. To demonstrate the use of the proposed algorithm we present an analysis of different peer-to-peer scenarios including the topology recently crawled from Gnutella2 network. As a comparison algorithms we use breadth-first search, self-avoiding random walk and highest degree search and the proposed minimum spanning tree *k*-Steiner algorithm (MST *k*-Steiner) as an approximation of optimal using global knowledge of network topology and resources. The results show that there is a significant gap between the performance of local search algorithms and the optimal solution.

2. Related Work

Peer-to-Peer resource discovery problem has been investigated extensively in the research literature [1,4,6,8,9,10,16,20,22,23,25].

Adamic et al. [1] propose High-Degree Seeking algorithm for finding one node in a graph by forwarding query to the highest degree neighbor, which has not yet been visited. They evaluate the performance of their algorithm in random graphs, power-law graphs and a snapshot of Gnutella P2P network. Compared to Random Walker, where query is forwarded to a randomly selected neighbor, the traffic reduction is in the order of magnitude.

Lv et al. [12] evaluate the use of multiple Random Walkers and Expanding Ring algorithm against Breadth-First Search (BFS) in random graphs, power-law graphs and a regular two-dimensional grid graph as well as in a snapshot of Gnutella. Traffic reductions of one or two orders of magnitude are gained with multiple Random Walkers compared to the BFS.

Crespo and Garcia-Molina [4] propose routing indices, which provide shortcuts for random walkers to locate the resources. As an evaluation graphs they use trees, trees with additional cycles and power-law graphs. Compared to random walkers routing indices reduce the traffic up to 50% and compared to BFS the traffic reduction is in the order of one or two magnitudes with uniform resource distributions.

Yang and Garcia-Molina [25] propose Directed BFS, which selects the first neighbor based on heuristics and further uses BFS for forwarding the query. They also propose the use of Local Indices for replicating resources to a certain radius of hops from a node. Evaluations are conducted on a snapshot of Gnutella and the performance of these algorithms are compared to the BFS. The Directed BFS reduces traffic to 38% while locating significantly less resources than the BFS. Local Indices, however, locates similar numbers of resources as the BFS with 39% traffic generated by the BFS.

Kalogeraki et al. [8] propose Modified Random Breadth-First Search as an improvement to the BFS algorithm. In their algorithm only a subset of neighbors are selected for forwarding. Also, they propose an Intelligent Search Mechanism, which stores the performance of past queries for each neighbor and thus can direct further queries to the neighbors, which are likely to have the queried resource. For evaluation they use randomly connected P2P network and reduce traffic to 35% compared to the BFS.

Menascé [19] follows the ideas of Kalogeraki et al. and propose a modification of BFS, where only a subset of neighbors are randomly selected for forwarding. Evaluations are done in a random graph without a comparison algorithm.

Tsoumakos and Roussopoulos [22] propose Adaptive Probabilistic Search, where the feedback from previous queries is used to tune probabilities for the further forwarding of random walkers. The algorithm is evaluated in random graphs and power-law graphs against Lv et al.'s multiple Random Walkers and Gnutella's UDP extension for scalable searches [5]. While keeping approximately the same level of traffic, APS doubles the success rate of queries compared to multiple Random Walkers.

Sarshar et al. [20] propose Percolation Search algorithm for power-law networks. The idea is to replicate copies of resources to sufficient number of nodes and thus ensure that the algorithm locates at least one replica of the resource. The algorithm's performance is evaluated in power-law graphs and a snapshot of Gnutella P2P network without a comparison algorithm.

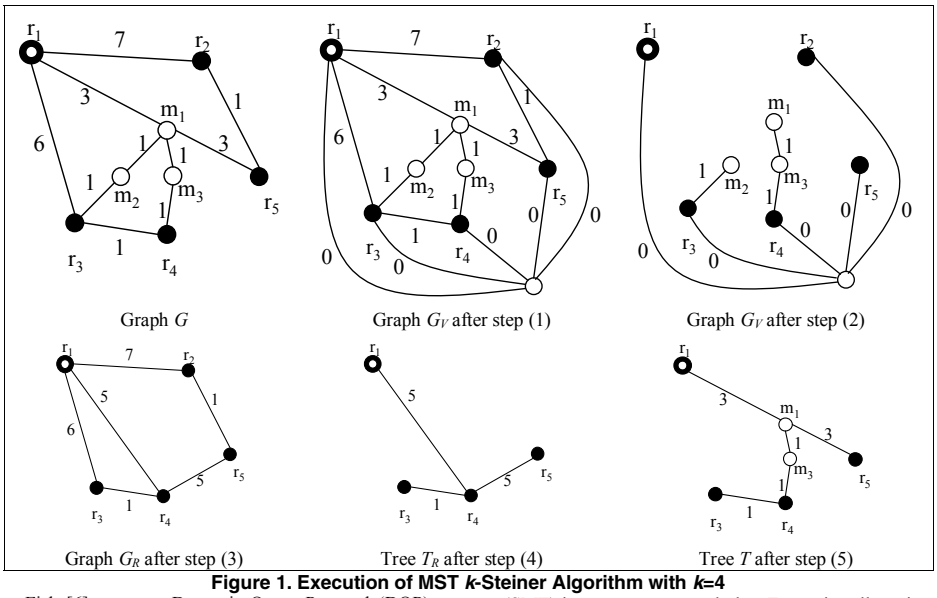

Figure 1. Execution of MST *k*-Steiner Algorithm with *k*=4

Fisk [6] proposes Dynamic Query Protocol (DQP), which has now been implemented in Gnutella2 peers. DQP executes first a probe query to estimate how rare the resource is and based on the obtained results calculates proper TTL and number of neighbors, which the query will be forwarded. The query is terminated when 150 resource instances has been located, there are no connections left to query or when the theoretical horizon of the query has hit the limit of 200,000 peers.

Vapa et al. [23] propose NeuroSearch, which is a neural network based resource discovery algorithm. In NeuroSearch a neural network is given a set of heuristics and by calculating the output of the neural network the algorithm can decide which of the neighbor nodes will receive the query. The evaluations are done in small power-law graphs and the traffic is reduced approximately to 80% from the BFS.

The main theme of all the papers reviewed in this section has been to introduce new algorithm(s) and to compare their performance to other algorithms of a similar type. However, the level of performance is not properly identified if the optimal performance is not measured in the simulations. The algorithm proposed later in this paper aims to overcome this problem.

3. Steiner Minimum Tree Problem

Let $G = (V,E)$ be an undirected graph, where V is a set of vertices and E is a set of edges having edge costs. Given a terminal set $R \subseteq V$, a Steiner minimum tree (SMT) is a tree $T \subseteq G$ such that T contains all vertices of R and the length $w(T)$ is minimum among all Steiner trees. $w(T)$ is defined as a sum of all edge costs $e \in E$ contained in T.

Compared to a minimum spanning tree, which contains all vertices of a graph, SMT spans only a subset of vertices and thus if the cardinality of the terminal set $|R| = |V|$ these problems are equivalent. Also, if $|R| = 2$, SMT reduces to solving a shortest-path problem.

In SMT the vertices are divided into two sets: terminal vertices and non-terminal vertices. Terminal vertices belong to a set, which has to be included in the solution, whereas non-terminal vertices can shorten the length of the solution.

SMT is known to be *NP*-complete problem [9]. Being in complexity class *NP* means that there exists a polynomial time algorithm to check whether the given solution is a correct Steiner tree and whether the length of a given solution is less than a given bound B, but there is no polynomial algorithm (unless $P=NP$) that would find such a Steiner tree. Therefore exact solving of the problem is not practical with large graphs. Also, when a problem is classified as *NP*-complete it means that the problem is the hardest among all problems contained in *NP*. More information about the *NP*-completeness of the Steiner tree problem can be found in [19].

Because SMT is *NP*-complete, approximation needs to be used. An approximated solution is not guaranteed

to locate the Steiner minimum tree, but it can give guarantees that the length of a located solution is within certain range from the optimal solution.

4. Peer-to-Peer Resource Discovery As Steiner Tree Problem

As was described earlier the peer-to-peer resource discovery problem does not map to the Steiner tree problem if only part of the resources needs to be found. Therefore we introduce *k*-Steiner Minimum Tree problem as described in [3] with an addition of a root vertex to the solution set. In Rooted *k*-Steiner Minimum Tree problem (Rooted *k*-SMT) it suffices to select only *k* terminal vertices from *R* to be included in the Steiner minimum tree starting from the root vertex *r*. Also we propose an approximation algorithm for solving the Rooted *k*-SMT problem.

4.1. Rooted *k*-Steiner Minimum Tree

Problem: `Rooted k-Steiner Minimum Tree`

Given: A connected graph $G = (V,E)$, a terminal set $R \subseteq V$, a root vertex $r \in R$ and $2 \le k \le |R|$

Find: A Steiner tree T for R in G rooted to vertex r and containing k terminal vertices, such that $w(T) = \min \{|w(T')| \mid T'$ is a Steiner tree for k vertices in $R\}$

The Rooted *k*-SMT becomes equivalent to the SMT by selecting $k=|R|$ and as a root any vertex in *R*. The SMT thus reduces to a special case of the Rooted *k*-SMT and therefore Rooted *k*-SMT for all *k* is at least as hard as SMT. When applied to the resource discovery problem the terminal set *R* is formed of query originator as root vertex and $|R|$-1 resource instances.

4.2. Approximation of Rooted *k*-Steiner Minimum Tree Problem With Minimum Spanning Tree

A well-known method for approximating the SMT is the use of a minimum spanning tree (MST) [19,24]. The MST *k*-Steiner Minimum Tree algorithm (MST *k*-Steiner) proposed here uses the same principles as MST-approximation algorithm to locate a solution for Rooted *k*-SMT.

MST *k*-Steiner starts by computing Voronoi regions of each terminal node. Voronoi region of a terminal node contains all the nodes which are closer to that terminal node than to other terminal node. Voronoi regions can be computed by adding one node in the graph *G* and connecting this node to all terminal nodes of *R* with edge length 0. Let G_V denote this graph. Then by executing a minimum spanning tree on G_V the Voronoi regions are obtained. This also gives the distance of each non-terminal node to its closest terminal node. The technique used here was introduced by Mehlhorn in [15].

Next, the Voronoi regions are used to compute the shortest distance graph G_R of vertices in *R*. Let $l(u,v)$ denote the edge cost of the edge between nodes *u* and *v*. Let $l(u)$ denote the distance of node *u* from the closest terminal node. Let $t(u)$ denote the closest terminal node of node *u*. Shortest distance graph G_R is obtained by going through each edge (u,v), $u,v \in E$, $u \ne v$ and computing the two triplets $(t(u), t(v), l(u)+l(u,v)+l(v))$ and $(t(v), t(u), l(u)+l(u,v)+l(v))$. These triplets are collected in a list and only those where $t(u) \ne t(v)$ and $l(u)+l(u,v)+l(v)$ is the shortest are kept in the list. This list is used to create the graph G_R by associating two terminal nodes *u* and *v* if they have a corresponding triplet in the list and setting the edge cost to be the third value of the triplet.

Then a *k*-minimum spanning tree approximation T_R containing *k* vertices is located greedily from G_R by selecting the closest node to the spanning tree starting from the vertex *r* and decomposed back to the original graph by replacing the edges with their shortest paths.

Algorithm: `MST k-Steiner Minimum Tree`

Input: A connected graph $G = (V,E)$, a terminal set $R \subseteq V$, a root vertex $r \in R$ and $2 \le k \le |R|$

Output: A Steiner tree T for R in G rooted to the vertex *r* containing *k* terminal vertices.

(1) Add one node to the graph *G* and connect it to all terminal nodes contained in *R* with an edge having cost 0. The result is denoted as graph G_V.
(2) Replace G_V with the minimum spanning tree of G_V.
(3) Compute the shortest path between two terminal nodes by iterating all edges of *E* in *G* and constructing the corresponding triplets. Transform the resulting triplets to graph G_R.
(4) Compute a *k*-minimum spanning tree approximation T_R from G_R rooted to the vertex *r* and containing *k* vertices of *R*.
(5) Transform T_R into subtree *T* of *G* by replacing each edge of T_R by the corresponding shortest path.

An example execution of the MST *k*-Steiner algorithm when $k=4$ and $|R|=5$ is shown in the Figure 1. In the figure a graph *G* is given with the terminal set $R = \{r_i \mid 1 \le i \le 5\}$ (denoted as ● including root vertex r_1, which is denoted as ◘) and the non-terminal nodes $m_i, 1 \le i \le 3$ (denoted as ○). Integers associated to the edges represent the edge costs.

5. Time Complexity

MST *k*-Steiner executes MST algorithm once in step (2) and once in step (4) stopping when *k* nodes have been reached. The transformation of the graph in step (3) using bucket sort [19] requires at maximum $|V|\log|V|+|E|$

steps, where $|V|$ is the number of vertices in the input graph G and $|E|$ is the number of edges in input graph G. Therefore the time complexity required for the algorithm is:

$$MST + MST_k + |V|\log|V| + |E|, \quad (5.1)$$

where MST denotes the time required for executing the Minimum Spanning Tree and MST_k denotes the time required for executing the Minimum Spanning Tree for k nodes. Certainly $MST_k \leq MST$ and $|V| \leq |E|\text{-}1 \leq |E|$, bounding the time complexity to:

$$2*MST+|E|\log|E|+|E|. \quad (5.2)$$

Minimum Spanning Tree can be implemented for example using the Kruskal's algorithm [24] having $O(|E|\log|E|)$ time complexity. Therefore MST k-Steiner algorithm's time complexity is $O(|E|\log|E|)$, which allows the algorithm to be used also in large graphs.

6. Approximation Ratio

Approximation ratio of an algorithm is computed as a ratio between the worst case performance and the optimal performance. For $k = 2$ the approximation ratio is 1, because the shortest path to the nearest resource is always selected. Also when $k = |R|$, MST k-Steiner reduces to a well-known MST-approximation algorithm [19,24] for Steiner Minimum Tree problem having approximation ratio 2. So, it still remains to determine what the approximation ratio is when $2 < k < |R|$.

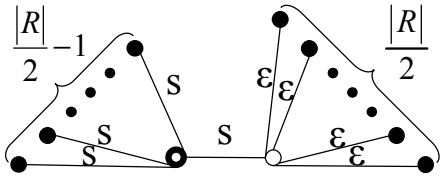

Figure 2. A graph where MST *k*-Steiner makes a large approximation error

A difficult case for MST k-Steiner is a graph shown in Figure 2. In the scenario, the root node is located within S distance from $\frac{|R|}{2}-1$ terminal nodes and within $S + \varepsilon$ distance from the other half of terminal nodes. The difference between these distances is that on the left hand side discovering each terminal node requires travelling S distance and on the right hand side discovering the first terminal node requires travelling $S + \varepsilon$ distance, but then the other terminal nodes can be discovered with ε distance.

Without a loss of generality the analysis can be restricted to cases where $|R|$ is even. Now the approximation ratio α between the discovered path and the optimal path can be calculated for $k = \frac{|R|}{2}$ as:

$$\alpha = \frac{\left(\frac{|R|}{2}-1\right)S + S + \varepsilon}{S + \frac{|R|}{2}\varepsilon} = \frac{\frac{|R|}{2}S + \varepsilon}{S + \frac{|R|}{2}\varepsilon} \quad (6.1)$$

Considering $\varepsilon \approx 0$ the approximation ratio becomes:

$$\alpha = \frac{|R|}{2} \quad (6.2)$$

This implies that when the size of the terminal set grows and the number of discovered terminals k is close to $\frac{|R|}{2}$ the approximation ratio can become large. Still, the approximation ratio seems to be bounded to $\frac{|R|}{2}$, because adding terminal node on the left hand side and removing one terminal node from the right hand side makes the optimal path longer while keeping the discovered path almost the same (decreased by ε). In contrast by adding a terminal node on the right hand side and removing one terminal node from the left hand side makes the discovered path shorter while keeping the optimal path the same. Also decreasing k from $\frac{|R|}{2}$ will decrease the length of the discovered path faster than the optimal path thus giving a lower approximation ratio. Increasing k will lengthen the optimal path faster than the discovered path resulting in a lower approximation ratio than $\frac{|R|}{2}$.

As a summary, the approximation ratio of the algorithm depends on the number of available resources and can be no less than $\frac{|R|}{2}$. It is still left for future work to show that the ratio could not be even worse. There are however approximation algorithms for k-Steiner Minimum Tree, which achieve constant factor approximation ratios [3]. They rely on integer programming and by relaxing the constraints to a linear program sustain approximation ratio guarantees.

7. Simulations

In this section we present an analysis of five algorithms: Breadth-First Search (BFS) [13], Self-avoiding Random Walk (RWSA), Highest Degree Search (HDS) [1,10], Dynamic Query Protocol (DQP) [6] and MST k-Steiner Minimum Tree. The simulations were conducted in P2PRealm network simulator.

7.1. Peer-to-Peer Network Scenarios

As simulation scenarios we used power-law graphs, normal distributed random graphs and a recently measured topology of Gnutella2 P2P network [21] with an edge cost 1 for all edges. Power-law graphs were generated using Barabási-Albert model [2]. In power-law network few hub nodes have many neighbors

and many nodes have only few neighbors. Gnutella2 topology was obtained by extracting the largest connected component from the topology data of 02/02/05 presented in [21] and removing those nodes whose edges were not referenced by other nodes. Finally those edges whose one end point was missing were removed.

Resource instances were allocated for power-law and random graphs based on the number of neighbors each node had such that the number of different resource instances in a node was the same as the number of neighbors the node had. This means that in the power-law graphs the hubs were more likely to contain the queried resource. Resources were allocated to nodes by randomly sampling from a uniform distribution. The queried resources and the querying nodes were selected also randomly from a uniform distribution for each query.

In Gnutella2 topology the resources were allocated based on the measured resource distributions of Gnutella network in September 2003 [14]. The number of different resources was selected to be 10, so the topology files could be kept small enough, but the number of resource instances for each resource was sampled from the resource distribution of [14] which produced 43216 different resources instances. These resource instances were allocated randomly to nodes following the measured distribution of shared files in nodes [14] such that one node could not have multiple instances of the same resource. Now when 100 queries were executed each resource was queried multiple times, but from a different location, which was randomly selected. The queried resource was selected according to the peer keyword distribution of [14].

Table 1 illustrates the characteristics of each scenario used in the simulations.

Table 1. Simulation Scenarios

Scenario	PL10000	N10000	Gnutella2
Distribution	Power-Law	Normal	-
Nodes	10000	10000	74297
Edges	19997	19997	609036
Largest hub	161	11	360
Resources	1000	1000	10
Res. instances	39994	39994	43216
Queries	100	100	100
Diameter	8	10	12

7.2. Results

The tests were conducted by varying the target amount of resource instances that was needed to be found by the algorithms. The target percentage of the discovered resource instances determines the amount how many resource instances of a certain resource needs to be discovered out of all resource instances of that resource and represents the k parameter of the Rooted k-SMT problem. The measured variables were the average number of query packets used in a query as shown in figures 3, 4 and 5 and the average number of maximum hops as shown in figures 6, 7 and 8.

As can be seen from Figure 3 in power-law graphs MST k-Steiner algorithm produces query paths between one and two orders of magnitude shorter than local search algorithms. Also, the approximation error of MST k-Steiner in the scenario is at most $\alpha = 2$, because the theoretical optimum is k-1 query packets when each node can have only one instance of the queried resource. k-1 represents a situation that each forwarded query packet would locate one new resource instance and the query originator does not have the queried resource.

The performance of HDS is close to the paths of MST k-Steiner algorithm when only one or two instances of resources needs to be located (resource percentage < 3%). This is a bit surprising even though the scenario is designed directly for HDS type of algorithms. The resources are discovered more probably in the center of the network and as noted in [1] HDS travels those nodes early in the search process. However, when more resources needs to be discovered HDS travels multiple times to the central nodes and sometimes randomly forward the query packet decreasing the performance. Compared to RWSA and BFS, HDS performs significantly better when half of the available resource instances needs to be located and after that RWSA becomes a better algorithm. BFS in turn is at the same level with RWSA when less than 40% of resources needs to be located having TTL values between 1 and 4. With TTL values 5-7 BFS cannot keep up with RWSA. DQP is significantly less performing than BFS when small amount of resource instances needs to be located, because DQP requires always executing a two hop query first. DQP however reaches the same level with BFS when 40% or more resources needs to be located. Because of maximum TTL restrictions DQP cannot locate more than 60% of available resource instances.

In normal distributed graphs, as shown in Figure 4, MST k-Steiner retains its characteristics having largest approximation error at most $\alpha = 4$. Normal distributed graphs have larger diameter than power law distributed graphs and therefore estimating the optimal performance with k is too pessimistic. This argument is supported by the fact, that when 100% of resource instances needs to be discovered, the approximation ratio is at maximum $\alpha = 2$ as discussed in Section 6. It is therefore not clear, whether as short paths as k would exist in the normal distributed graph and presumably the real approximation error is at a similar range as in power-law graphs. Thus we conclude that the approximation ratio derived in section 6 highly overestimates the optimal performance in power-law and normal distributed P2P scenarios.

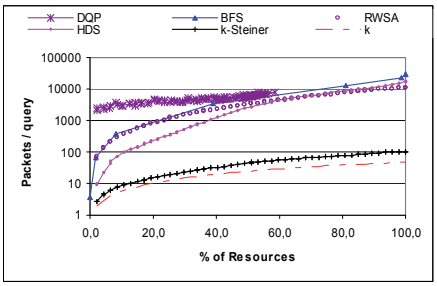

Figure 3. Query packets in PL10000

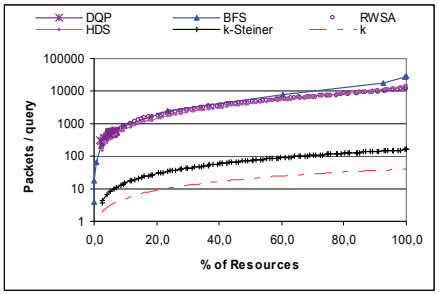

Figure 4. Query packets in N10000

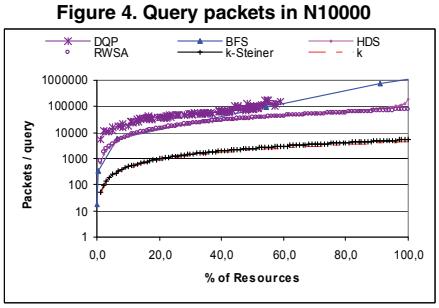

Figure 5. Query packets in Gnutella2

The difference between local search algorithms and MST *k*-Steiner paths is again in the order of one or two magnitudes. In contrast to power-law graphs, the local search algorithms in normal distributed graphs have similar performance when less than half of available resource instances needs to be located. After that RWSA and HDS outperform BFS. Random graph does not contain hub nodes and therefore HDS does not benefit from its ability to travel to high degree nodes. Basically, HDS appears as a self-avoiding random walker, because all the neighbors are almost equally connected. The large diameter of normal distributed graph restricts DQP

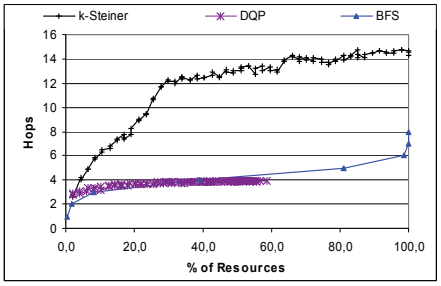

Figure 6. Maximum number of hops in PL10000

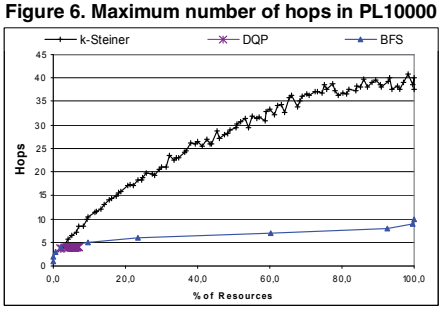

Figure 7. Maximum number of hops in N10000

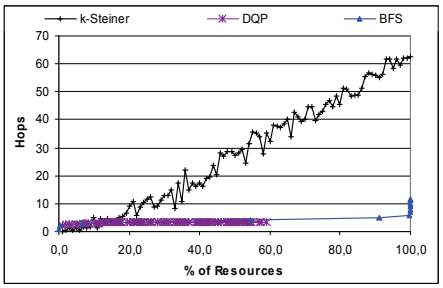

Figure 8. Maximum number of hops in Gnutella2

to locate only 7% of resource instances with time-to-live 4.

In Gnutella2 topology, as shown in Figure 5, MST *k*-Steiner does not seem to make any approximation error suggesting that Gnutella2 topology is highly connected and thus allowing each hop of a query to locate a new resource instance. The difference between MST *k*-Steiner paths and local search algorithms is in the order of a magnitude. HDS and RWSA perform equally well and BFS can keep up with them to 40% of resource instances. Then BFS departs to the level of DQP, which can locate at maximum 60% of resource instances.

The average of maximum hops for MST *k*-Steiner, BFS and DQP is plotted in Figures 6, 7 and 8. HDS and RWSA are omitted as their number of hops is shown in Figures 3, 4 and 5. Because HDS and RWSA forward to only one direction at a time their maximum hops are in different scale than what MST *k*-Steiner, BFS and DQP are using. Therefore if low latency in the network is critical, HDS and RWSA may not be suitable as local search algorithms. From the Figures 6 and 7, it can be seen that BFS and DQP require in N10000 two or three hops more than in PL10000 to locate the same amount of resource instances. BFS locates the shortest paths to resources and therefore has a small latency. However, MST *k*-Steiner does not seem to be using these paths. Reason for this is that the shortest paths do not necessarily contain resources along the path and therefore collecting some resources using a longer route may lead to a path which is more efficient. The latency in power-law graphs also stays comparable to BFS, but in normal distributed graphs the length of query paths grows significantly. This is, however, in completely different scale than the hops used by HDS and RWSA.

8. Conclusion

The Rooted *k*-Steiner Minimum Tree problem connects the resource discovery problem to a solid foundation of graph theory providing means to calculate near-optimal query paths in a graph. The MST *k*-Steiner algorithm computes an approximation of the shortest tree between the querying node and the nodes having the queried resource instances and thus is an upper bound for the performance of local search algorithms. The algorithm can be used in cases, where nodes contain only one instance of queried resource and the problem has to be further extended if multiple resource instances in a node is to be supported. In overall, the results presented here show that local search algorithms commonly used in P2P networks are far from optimal paths.

Based on the findings in Gnutella2 topology, DQP has slightly lower performance than BFS, but because of automatic adaptation of time-to-live parameter it can be feasibly used in current P2P networks. HDS and RWSA suffer from implementation problems because to avoid already visited nodes they need to keep record of visited nodes and therefore the size of the query packet grows in large graphs limiting their use.

What makes the resource discovery problem hard in P2P networks is that only local information is available. It would be interesting to know how close to the optimum can algorithms get using local knowledge. A record of the global network topology is used in Open Shortest Path First [17] IP routing protocol and Dijkstra's algorithm for computing the shortest paths suggesting possibilities that MST *k*-Steiner tree algorithm could be adapted to distributed P2P networks. In this case, information about the resources needs to be at least partially cached in the nodes. This, however, needs further research.

References

[1] L. A. Adamic, R. M. Lukose, B. A. Huberman, "Local Search in Unstructured Networks", Handbook of Graphs and Networks: From the Genome to the Internet, Wiley-VCH, 2003, pp. 295-317.

[2] A.-L. Barabási, R. Albert, "Emergence of Scaling in Random Networks", Science 286, 1999, pp. 509-512.

[3] F. A. Chudak, T. Roughgarden, D. P. Williamson, "Approximate k-MSTs and k-Steiner Trees via the Primal-Dual Method and Lagrangean Relaxation", Proceedings of the 8th International Integer Programming and Combinatorial Optimization Conference (IPCO), Springer, 2001, pp. 60-70.

[4] A. Crespo, H. Garcia-Molina, "Routing Indices For Peer-to-Peer Systems", Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS'02), IEEE Press, 2002, pp. 23-33.

[5] S. Daswani, A. Fisk, "Gnutella UDP extension for scalable searches (GUESS) v. 0.1".

[6] A. Fisk, "Gnutella Dynamic Query Protocol v0.1", Gnutella Developer's Forum, May 2003.

[7] M. Harchol-Balter, T. Leighton, D. Lewin, "Resource Discovery in Distributed Networks", 18th Annual ACM-SIGACT/SIGOPS Symposium on Principles of Distributed Computing (PODC'99), Atlanta, 1999.

[8] V. Kalogeraki, D. Gunopulos, D. Zeinalipour-Yatzi, "A Local Search Mechanism for Peer-to-Peer Networks", Proceedings of the 11th International Conference on Information and Knowledge Management, ACM Press, 2002, pp. 300-307.

[9] R. M. Karp, "Reducibility Among Combinatorial Problems", Complexity of Computer Computations, Plenum Press, New York, 1975, pp. 85-103.

[10] B. J. Kim, C. N. Yoon, S. K. Han, H. Jeong, "Path finding strategies in scale-free networks", Physical Review E 65, 2002.

[11] N. Kotilainen, M. Vapa, A. Auvinen, T. Keltanen, J. Vuori, "P2PRealm – Peer-to-Peer Network Simulator", 11th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks, IEEE Communications Society, pp. 93-99, Trento, Italy, 2006.

[12] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks", Proceedings of the 16th International Conference on Supercomputing, ACM Press, 2002, pp. 84-95.

[13] N. Lynch, "Distributed Algorithms", Morgan Kauffmann Publishers, 1996.

[14] P. Makosiej, G. Sakaryan, H. Unger, "Measurement Study of Shared Content and User Request Structure in Peer-to-Peer Gnutella Network", Proceedings of Design, Analysis, and Simulation of Distributed Systems (DASD 2004), Arlington, USA, April 2004. pp. 115-124.

[15] K. Mehlhorn, "A faster approximation algorithm for the Steiner problem in graphs", Information Processing Letters, vol. 27 issue 3, 1988, p. 125-128.

[16] D. A. Menascé, "Scalable P2P Search", IEEE Internet Computing, Vol. 7, No. 2, March-April 2003, pp. 83-87.

[17] J. Moy, "OSPF Version 2", RFC 2328, The Internet Society, April 1998.

[18] A. Oram, "Peer-to-Peer: Harnessing the Power of Disruptive Technologies", O'Reilly & Associates, March 2001.

[19] H.-J. Prömel, A. Steger, "The Steiner Tree Problem: A Tour through Graphs, Algorithms, and Complexity", Advanced Lectures in Mathematics, Vieweg Verlag, 2002.

[20] N. Sarshar, P. O. Boykin, V. P. Roychowdhury, "Percolation Search in Power Law Networks: Making Unstructured Peer-to-Peer Networks Scalable", Proceedings of the Fourth International Conference on P2P Computing (P2P'04), IEEE Press, 2004, pp. 2-9.

[21] D. Stutzbach, R. Rejaie, S. Sen, "Characterizing Unstructured Overlay Topologies in Modern P2P File-Sharing Systems", Proceedings of the ACM SIGCOMM Internet Measurement Conference, Berkeley, October 2005.

[22] D. Tsoumakos, N. Roussopoulos, "Adaptive Probabilistic Search for Peer-to-Peer Networks", Proceedings of the Third International Conference on P2P Computing (P2P'03), IEEE Press, 2003, pp. 102-109.

[23] M. Vapa, N. Kotilainen, A. Auvinen, H. Kainulainen, J. Vuori, "Resource Discovery in P2P Networks Using Evolutionary Neural Networks", International Conference on Advances in Intelligent Systems – Theory and Applications (AISTA 2004), 2004.

[24] B. Y. Wu, K.-M. Chao, "Spanning Trees and Optimization Problems", Discrete Mathematics and Its Applications, Chapman & Hall/CRC, 2004.

[25] B. Yang, H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks", Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS'02), IEEE Press, 2002, pp. 5-14.