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.

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

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).

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*.

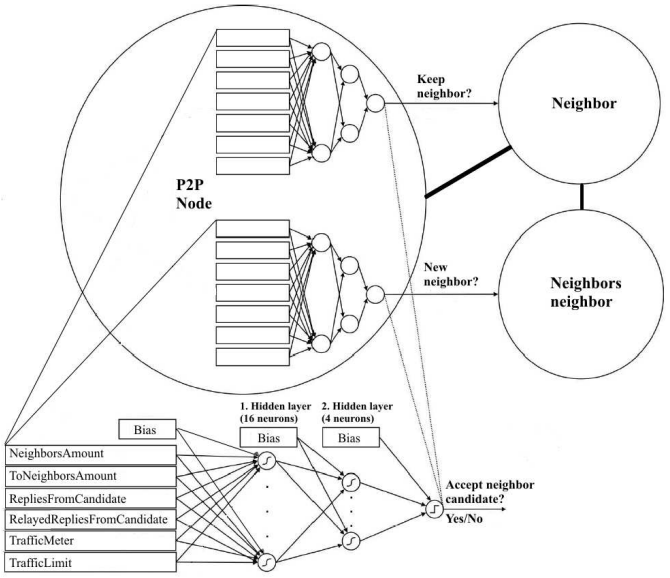

Fig. 1: The neighbors of the peer are determined with the neural network that receives information about neighbor candidates in its inputs.

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:

$$F_j = \begin{cases} Rr - (p + t + Tt_a), & if\ Rr - (p + t + Tt_a) \geq 1 \\ \frac{1}{p + t + Tt_a - Rr}, & otherwise \end{cases} \quad (4.1)$$

$$F_j = \begin{cases} 2Rr - (p + t + Tt_a), & if\ Rr - (p + t + Tt_a) \geq 1\ and\ r \geq G \\ Rr + (t + t_a - p), & if\ Rr - (p + t + Tt_a) \geq 1\ and\ r < G \\ \frac{1}{p + t + Tt_a - Rr}, & otherwise \end{cases} \quad (4.2)$$

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:

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

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.

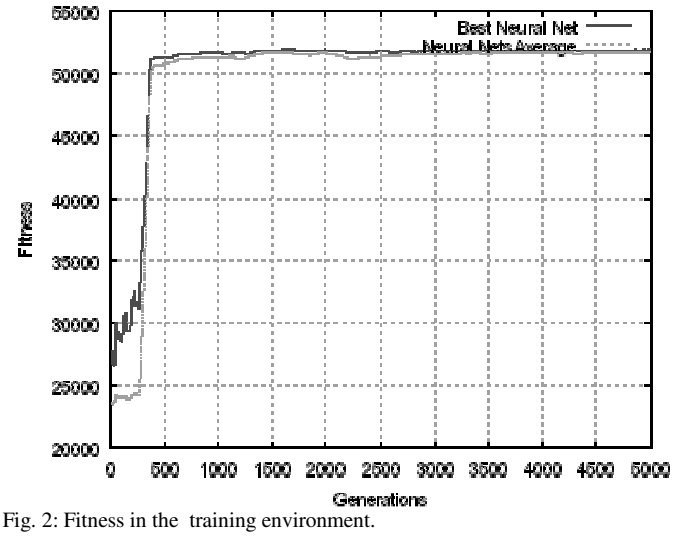

Fig. 2: Fitness in the training environment.

Fig. 3: Fitness in the generalization environment.

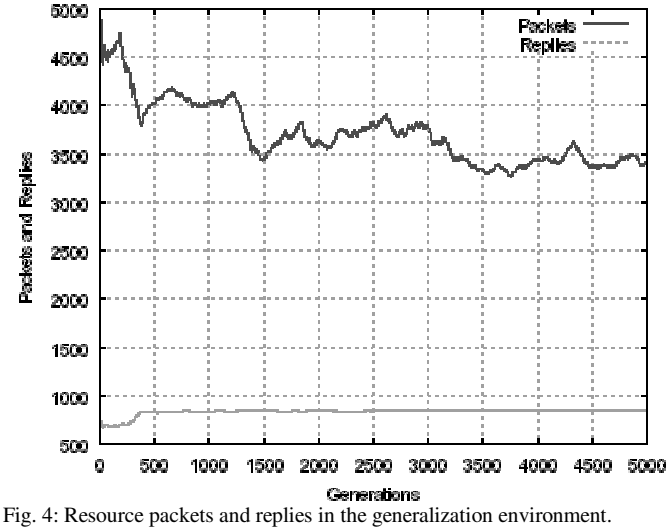

Fig. 4: Resource packets and replies in the generalization environment.

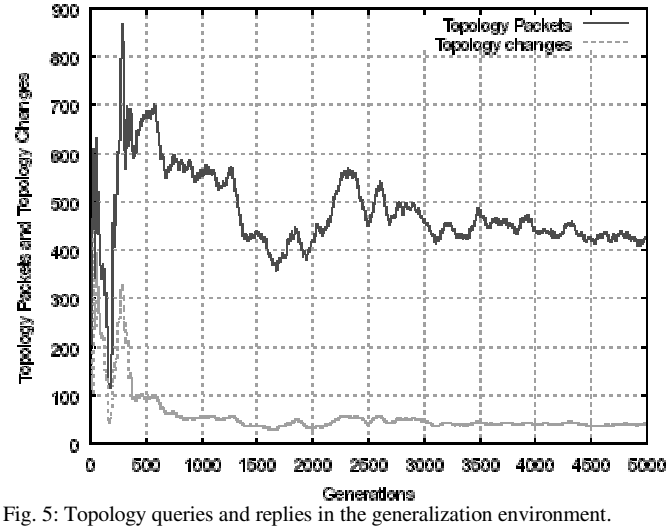

Fig. 5: Topology queries and replies in the generalization environment.

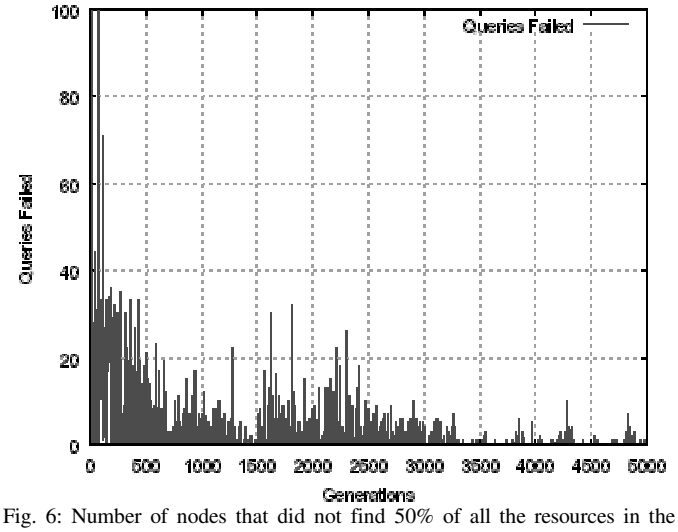

Fig. 6: Number of nodes that did not find 50% of all the resources in the generalization environment.

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

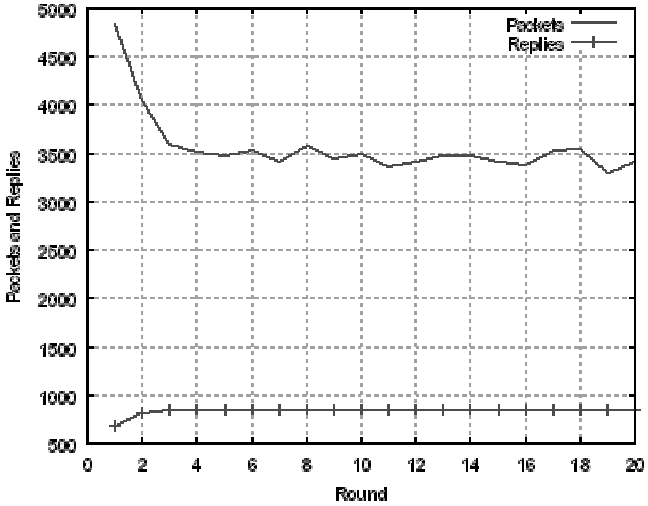

Fig. 7: NeuroTopology manages to make the grid P2P network more effective for the HDS algorithm during the first four rounds of resource querying and topology changing.

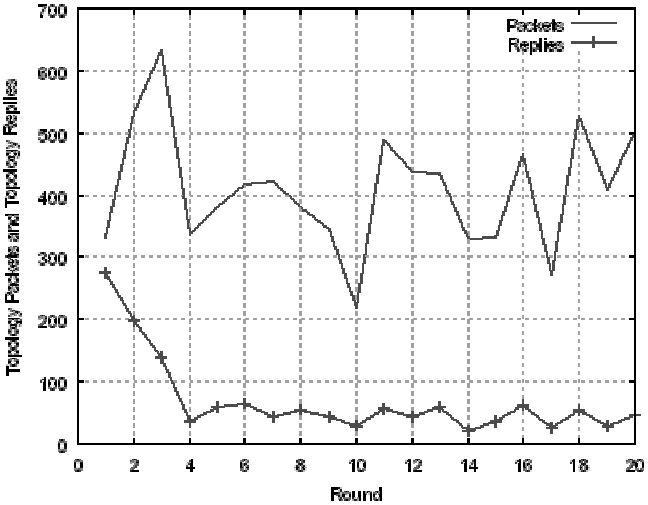

Fig. 8: The amount of topology changes convergences during the first four rounds.

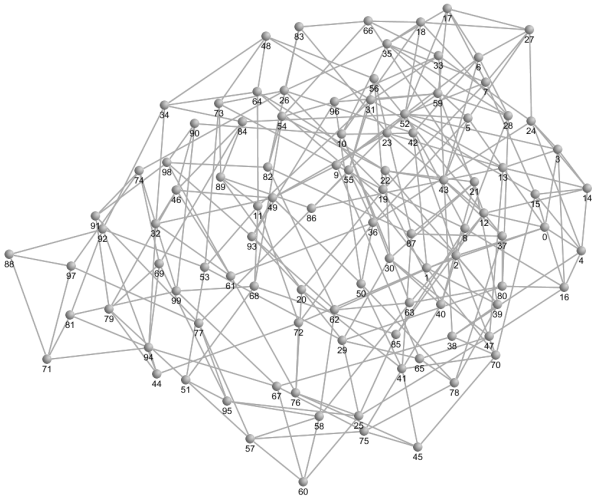

Fig. 9: End result P2P topology after 20 rounds when started from the grid topology.

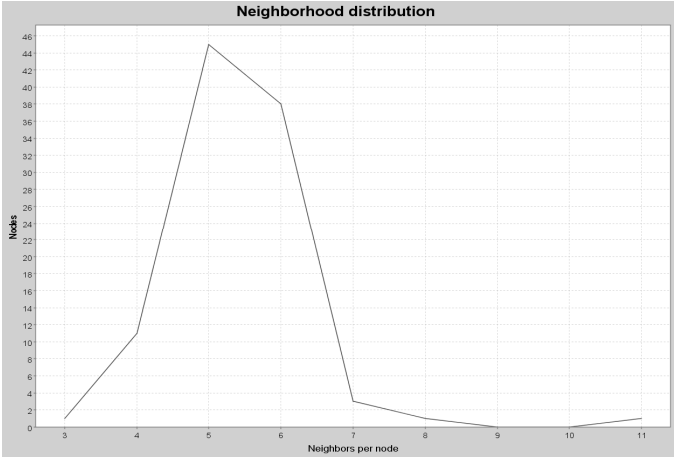

Fig. 10: Neighborhood distribution of the topology in Fig. 9.

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.