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.

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_is 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 $|N|$ 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.

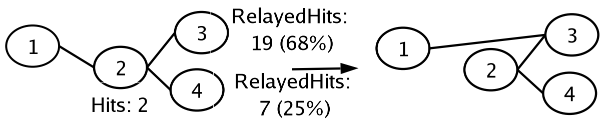

Figure 1. A situation before and after executing the overtaking algorithm.

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

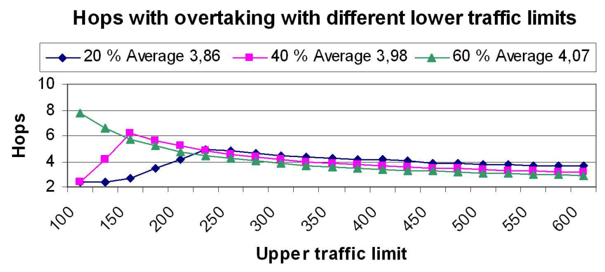

Figure 2. Average hops with different lower traffic limit percents with overtaking.

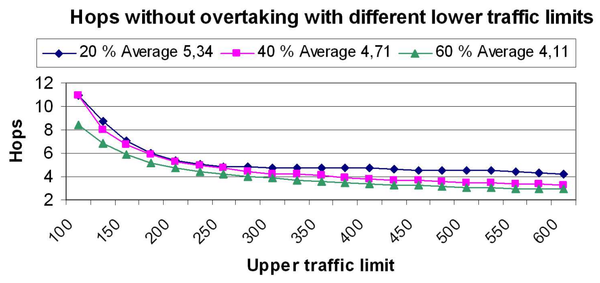

Figure 3. Average hops with different lower traffic limit percents without overtaking.

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

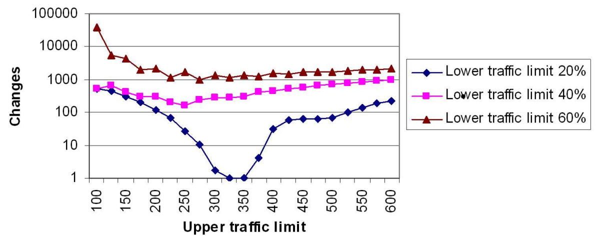

Figure 4. The amount of changes with lower traffic limit percents and with different upper traffic limit values and without overtaking.

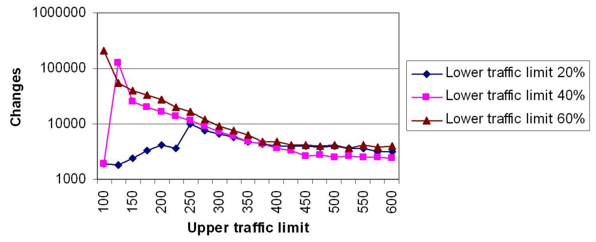

Figure 5. The amount of changes with lower traffic limit percents and with different upper traffic limit values and with overtaking.

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 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.