A Robust Protocol for Building Superpeer Overlay Topologies

Alberto Montresor

Technical Report UBLCS-2004-8

May 2004

Department of Computer Science
University of Bologna
Mura Anteo Zamboni 7
40127 Bologna (Italy)

A Robust Protocol for Building Superpeer Overlay Topologies [1]

Alberto Montresor[2]

Technical Report UBLCS-2004-8

May 2004

Abstract

The concept of superpeer *has been recently introduced to improve the performance of popular file-sharing applications. A superpeer is a node in a P2P network that operates as a server for a set of clients, and as an equal w.r.t. other superpeers. By exploiting heterogeneity, the superpeer paradigm allows P2P networks to run more efficiently, without compromising their decentralized nature. This paper proposes a novel generic mechanism for the construction and the maintenance of overlay topologies based on superpeers. This mechanism is based on the well-known gossip paradigm, with nodes exchanging information with randomly selected peers and re-arranging the topology according to the requirements of the particular P2P application. The resulting protocol is extremely efficient and robust, capable to deal with a continuous flow of nodes joining and leaving the system, as well as to repair a network where up to 100% of the existing superpeers have been removed.*

1. This work was partially supported by the Future & Emerging Technologies unit of the European Commission through Project BISON (IST-2001-38923).
2. Department of Computer Science, University of Bologna, Italy

1 Introduction

Modern P2P networks [10] present several unique aspects that distinguish them from traditional distributed systems. Networks comprising hundred of thousand or even millions of peers are not uncommon. As a consequence of such scale, they are characterized by extreme dynamism, with a continuous flow of nodes joining or leaving the network.

Such large scale and dynamism present several complex challenges to the developer. Neither a central authority nor a fixed communication topology can be employed to control the various components. Instead, a dynamically changing overlay topology is maintained and control is completely decentralized. The topology is defined by ”cooperation” links among nodes, that are created and deleted based on the requirements of the particular application.

Until recently, most of the P2P applications deployed on the Internet were characterized by the absence of a specific mechanism for enforcing a particular overlay topology. The consequence of this choice was the adoption of inefficient communication schemes, such as flooding. Today, however, the situation has changed: several academic projects have recognized the importance of selecting, constructing and maintaining appropriate topologies for the implementation of efficient and robust P2P systems [16, 11, 1].

Even popular file-sharing applications [8, 4] have started to consider more structured topologies. By introducing the concept of *superpeer*, their topologies are now organized through a two-level hierarchy: nodes that are faster and/or more reliable than “normal” nodes take on server-like responsibilities and provide services to a set of clients. For example, in the case of file sharing, a superpeer builds an index of the files shared by its clients and participates in the search protocol on their behalf, leveraging them from participating in costly protocols and reducing the overall traffic by forwarding queries only among superpeers.

The superpeer paradigm allows decentralized networks to run more efficiently by exploiting heterogeneity and distributing load to machines that can handle the burden. On the other hand, it does not inherit the flaws of the client-server model, as it allows multiple, separate points of failure, increasing the health of the P2P network.

Furthermore, the usefulness of the superpeer approach is not limited to file-sharing. For example, it is possible to envisage distributed game systems [14], where multiple locations of a simulated virtual environment are maintained by a distributed collection of superpeers, that manage the virtual environment on behalf of their clients. Grid management systems [5] and distributed storages [9] are other possible candidates for such architectural paradigm.

Building and maintaining a superpeer topology, however, is not simple. The extreme scale and dynamism call for robust and efficient protocols, capable to self-organize and self-repair a superpeer overlay in spite of both voluntary and unexpected events like joins, leaves and crashes.

Furthermore, the problem is complicated by the fact that the shape of a superpeer topology is strongly application-dependent. Several questions arise: How many superpeers are needed? How superpeers are linked together? Can nodes be clients of more than one superpeer? For example, consider a file-sharing application: nodes could be client of more than one superpeer without compromising efficiency (but reducing performance, though), and superpeers could be connected in an unstructured, Gnutella-like fashion. On the other hand, in P2P gaming, nodes must be client of a single superpeer, and the sub-topology of superpeers must reflect the characteristics of the particular virtual environment that is simulated.

The contribution of this paper is to present a novel protocol for the construction and management of superpeer-based overlay topologies. The protocol is based on the well-known gossip paradigm [2, 3]. Each node periodically initiates a *information exchange* with a peer node selected randomly. The nodes involved in the exchange send each other information about their current status: whether they are superpeers or simply clients, the number of clients they are serving, and so on. Based on this information, role changes and/or client transfers can occur: a client may decide to become a superpeer and take responsibility for some of the clients of the other node, to alleviate its load; alternatively, a superpeer may decide to move all its clients to the other node and become a client by itself, to reduce the number of superpeers and thus the traffic generated by communication between superpeers.

Our protocol generates topologies with the following characteristics: (i) every client is associated with exactly one superpeer; (ii) superpeers are connected through a random network; (iii) the number of superpeers is minimized, i.e. it impossible to find a smaller set of nodes whose total capacity is sufficient to cover all remaining nodes as clients.

Such a topology can be easily exploited to implement a file-sharing application, as minimizing the number of superpeers (compatibly with the capability of superpeers to deal with the traffic generated by clients) has the effect of decreasing the total traffic generated by the application. Yet, we believe that the proposed approach is general enough to be customized for producing other topologies as well. For example, it could be easily extended with mechanisms for introducing superpeer redundancy, with nodes being clients of multiple superpeers.

The resulting protocol has proven to be extremely robust. The continuous gossiping of topology information captures the dynamic nature of P2P systems: nodes may learn about a new node by receiving its identifier in an exchange, while crashed nodes are progressively forgotten and then removed from the network. Robustness is not limited to dynamic scenarios where a continuous flux of nodes joins and leave the network; the protocol is even capable to deal with catastrophic failures, where any percentage of the superpeers (up to 100%) are suddenly removed from the network. For space restrictions, communication failures are not dealt explicitly in this paper; yet, the mechanism adopted for dealing with node crashes can be easily adapted to deal with message omissions and partitionings.

Furthermore, our protocol is also extremely efficient — the total number of messages exchanged among all nodes scales linearly with the size of the network — and ”democratic” — no single node is ever required to deal with an excessive number of messages. Finally, and most importantly, the time needed to self-organize the network into an almost-optimal configuration is constant with respect to network size, while the time needed to self-organize the network into an optimal configuration scales logarithmically.

The paper is organized as follows. An informal model of our network environment is presented in Section 2. Section 3 illustrates our algorithm. Section 4 describes the experimental results that validate our mechanism. Finally, Section 5 discusses previous results regarding superpeer networks and draws directions for future work.

2 System Model

We consider a network consisting of a large collection of *nodes* that are assigned unique identifiers and that communicate through message exchanges. The network is highly dynamic; new nodes may join at any time, and existing nodes may leave, either voluntarily or by *crashing*. Since voluntary leaves can be trivially managed by simple “logout” protocols, in the following we limit our discussion to node crashes, that present a much harder challenge. Byzantine failures, with nodes behaving arbitrarily, are excluded from the present discussion.

We assume that nodes are connected through an existing routed network, such as the Internet, where every node can potentially communicate with every other node. To actually communicate, a node has to know the identifiers of a set of other nodes, called its *neighbors*. This neighborhood relation over the nodes defines the topology of the *overlay network*. Given the large scale and the dynamism of our envisioned system, neighborhoods are typically limited to small subsets of the entire network. The neighbors of a node (and so the overlay topology) can change dynamically.

Nodes are heterogenous: they may differ in their computational and storage capabilities, and also (and more importantly) on the bandwidth of its network connection. In order to distinguish nodes that are capable to act as superpeers from nodes that can join just as clients, we associate each node n with a parameter c_n representing its *capacity*, i.e. the number of clients that can be handled by n. In other words, we use the concept of capacity to abstract in a single quantity all the characteristics listed above. In order to simplify our simulations, we assume that each node knows its capacity parameter; in a real implementation, this value could be computed on the fly, by performing on-line measurements; the result is strongly dependent on the particular application to be implemented. The techniques used to perform this computation are outside the scope of this paper.

3 The Algorithm

Informally speaking, a *superpeer overlay topology* is characterized as follows. Nodes must be assigned one of two mutually-exclusive roles: superpeer or client. The assignment of roles is not permanent: nodes

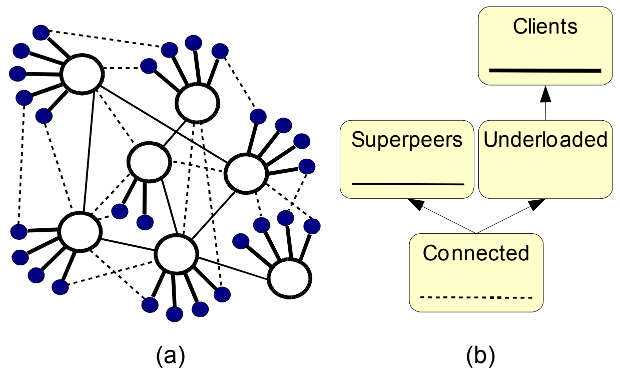

Figure 1. (a) The superpeer topology obtained from our protocol. b) The architecture of a node participating in the superpeer protocol.

may start as superpeers, and later become clients if nodes with more capacity appear. The neighborhood relationship of a node depends on its role: superpeers are linked to a random sample of other superpeers, as well as to the set of clients for which they are responsible for. Clients, on the other hand, are linked to exactly one superpeer.

Our goal is to produce a superpeer topology characterized by a minimum number of superpeers. In order to do that, we adopt a classification criteria based on the capacity measure introduced above: nodes with higher capacity are considered better candidates as superpeers. At each time, the *target topology* is the one composed by the minimum set of nodes whose total capacity is sufficient to cover all other nodes as clients. Clearly, only in a static network the target topology may be obtained; in the presence of dynamism, the real topology will just approximate it.

To build a topology with such characteristics, we propose a mechanism based on the well-known gossip paradigm [2, 3]. Topology information such as identifier, capacity, current role and neighborhood of participating nodes are disseminated through periodic gossip messages between randomly selected nodes. Based on the received information, nodes update their neighborhoods in order to obtain a better approximation of the target topology.

The continuous gossiping of node information has two effects. First, it enables the construction of overlays that capture the dynamic nature of P2P systems. Nodes may learn about new members of the system, by receiving their identifiers in an information exchange; or they may forget about crashed nodes that are not participating in the exchanges any more. Second, it provides nodes with continously fresh samples of the entire network, giving them the possibility of modifying the topology based on a larger choice of nodes.

Possible approaches to build a superpeer topology are (i) to transform an existing topology into a superpeer one, by rewiring the overlay connections between nodes; or (ii) to build such topology from scratch, by organizing the continuous flux of nodes joining and leaving the system. In either cases, the resulting topology would be the only one present in the system. Given the augmented fragility of superpeer overlays, this is potentially dangerous: if a large amount of superpeers fail, for example due to a malicious attack, the resulting topology may be disconnected. Although this may occur rarely, a mechanism to bootstrap again a new superpeer network with the remaining nodes is needed.

To solve this problem, we adopt an alternative approach: the superpeer topology is built as an additional overlay, superimposed over an existing connected topology and continously "extracted" from it through gossiping. The resulting topology is shown in Figure 1(a). Dashed lines represent the underlying topology, which does not make distinctions between clients and superpeers, but just provides a connected communication network over all participating nodes. The superpeer topology is shown using continuous lines, with thick ones representing client-superpeer relationship, and thin ones representing the connection between superpeers.

The modular architecture of each node is shown in Figure 1(b). Four different neighborhood sets are maintained, each of them representing a different kind of relationship with other nodes.

```
do forever
    wait($\delta$ time units)
    $q \leftarrow$ RANDOMPEER()
    SENDSTATE($q$)
    $s_q \leftarrow$ receive($q$)
    UPDATE($s_q, q$)
```

(a) active thread

```
do forever
    $s_q \leftarrow$ receive(*)
    RETURNSTATE($q$)
    UPDATE($s_q$, sender($s_q$))
```

(b) passive thread

Figure 2. The skeleton of a gossip-based protocol. Notation: q is the remote peer. s_q is the state received from q.

At the lower layer, the CONNECTED set contains the neighbors forming the underlying topology. This sample of the entire network is used by the gossip protocols implemented at the next layer to obtain a random node with which to perform an information exchange.

The middle layer is composed of two sets. SUPERPEERS contains a random sample of nodes currently acting as superpeers. UNDERLOADED contains only those superpeers that can accept additional clients, i.e. have a number of clients smaller than their capacity. The former is required by our definition of the target topology, while the latter is used by the next layer to obtain nodes candidates for client transfers and role changes.

Finally, the upper layer manages the client-superpeer relationship, by assigning superpeers with their current set of clients, and clients with their superpeer. This layer periodically samples the set of underloaded nodes to discover if a client transfer or a role change is required. For brevity, protocols below describe just how to manage the CLIENTS set stored at superpeers; the management of the corresponding superpeer link stored at clients is not discussed.

In the rest of this section, we present the gossip protocols that are executed at each of the layers. They are all obtained from the skeleton gossip protocol illustrated in Figure 2, by customizing the generic methods described below.

Each node is characterized by a local state, composed of a collection of node identifiers and associated data, and executes two different threads. The *active* one periodically initiates a *information exchange* with a node obtained through method RANDOMPEER by invoking method SENDSTATE and waiting for a response from the selected node. The *passive* one waits for messages sent by an initiator and replies using method RETURNSTATE. Both SENDSTATE and RETURNSTATE send a message representing the local state. Method UPDATE updates the local state based on the message received during the information exchange. The output of UPDATE depends on the specific topology implemented by the protocol.

Even though the system is not synchronous, we find it convenient to describe the protocol execution in terms of consecutive real time intervals of length δ called *rounds* that are enumerated starting from some convenient point. In each round, each node initiates *exactly one* exchange.

3.1 The Underlying Topology

Any protocol that generates a connected topology can be adopted to manage the CONNECTED set. Our choice is NEWSCAST, a gossip protocol that maintains an approximately random topology [6]. NEWSCAST has been used to implement several P2P protocols, including broadcast [6] and aggregation [7]. For space reasons, here we describe it very briefly; for additional information, please refer to [6].

In NEWSCAST, the state of a node is called *partial view* and it is constituted of a fixed-size set of *peer descriptors*. A peer descriptor contains the address of the node, along with a logical *timestamp* identifying the time when the descriptor was created. The size of a partial view is denoted by s.

Method RANDOMPEER returns an address selected randomly among those in the current partial view. Method UPDATE merges the partial views of the two nodes involved in an exchange and keeps the s freshest descriptors, thereby creating a new partial view.

New information enters the system when a node sends its partial view to a peer through SENDSTATE and RETURNSTATE. In this step, the node always inserts its own, newly created descriptor into the partial view. Old information is gradually and automatically removed from the system and gets replaced by new information. This feature allows the protocol to "repair" the overlay topology by forgetting crashed nodes, which by definition do not get updated because their owner is no longer active.

The resulting topology has several desirable features [6]: it has a very low diameter and is very close to a random graph with out-degree s. According to experimental results, choosing $s = 20$ is already sufficient for very stable and robust connectivity.

Since partial view are updated at each round, the topology is in a continuous state of flux. Apart from reflecting the natural dynamism of P2P networks, this feature has another benefit: partial views represent continuously fresh samples of the network, that improve the random selections performed by method RANDOMPEER implemented in the next layer.

Finally, it has also shown that, within a single round, the number of exchanges per node can be modeled by a random variable with the distribution $1 + \text{Poisson}(1)$. In other words, on the average, there are two exchanges per round (one is actively initiated and the other one is passively received) and the variance of this estimate is 1. This implies that no node is more important (or overloaded) than others.

3.2 Disseminating Superpeer Information

As explained above, the sets SUPERPEERS and UNDERLOADED maintained at each node contain samples of superpeers and underloaded superpeers, respectively. To disseminate information about these categories, two instances of NEWSCAST are executed, with the protocol slightly modified as follows. Method RANDOMPEER selects a random peer from CONNECTED, and not from SUPERPEERS or UNDERLOADED. Methods SENDSTATE and RETURNSTATE works as before, with the only exception that a newly created descriptor for the local node is added to the partial view only if the node belongs to the particular category for which the protocol is executed. Method UPDATE is unchanged. Finally, apart from identifier and time stamp, the descriptor of a node contains also the capacity parameter of that node, an information that will be used in the next layer.

Thanks to the modified RANDOMPEER, information is disseminated among all nodes, and not only those belonging to a particular category; in this way, even in the case of catastrophic failures involving all superpeers, the remaining nodes are able to disseminate information about the new elected superpeers. Thanks to the new SENDSTATE and RETURNSTATE nodes that stop to act as superpeers and nodes that are not underloaded any more are progressively removed from SUPERPEERS and UNDERLOADED.

3.3 Selecting Client and Superpeers

The gossip-based algorithm for establishing the client-superpeer relationships of the target topology is illustrated in Figure 3. Being more complex than the previous ones, this algorithm does not fit perfectly in the skeleton of Figure 2. Yet, we believe that presenting it by showing the basic methods discussed above greatly simplify the discussion.

The algorithm is executed only by superpeers: being more powerful, they can more easily pay the cost of their selection protocol. A joining node may either find a superpeer that is willing to accept it as client, or declare itself as superpeer, and start to participate in the selection protocol to see whether it deserves this role or not.

The rationale behind function RANDOMPEER is the following: all superpeers try to push clients towards more powerful nodes that are willing to accept more load. To do that, the function performs a random selection among those superpeers that are underloaded and whose capacity is larger or equal than the capacity of the local node. Since UNDERLOADED may contain obsolete information, multiple selections are made until a node is found whose capacity is effectively larger than the current number of clients. Ties (nodes with the same capacity) are broken by selecting the node with the larger number of clients. The process continues until such a node is found or no other nodes can be probed.

Once the random selection succeeds, function SENDSTATE sends how many of its clients as the chosen node is willing to accept. On the other side, the function RETURNSTATE sends an empty set, because the process of transferring clients is one-way.

In function UPDATE the local node (the one with more capacity) takes responsibility of the new clients by adding them to its local CLIENTS set. If the remote supernode turns out with an empty CLIENTS set, it becomes a client of the local node (if space permit). Otherwise, a last check is performed. If the local node has a client whose capacity is larger than the capacity of the remote one, the client and the remote node exchange their roles, by transferring the clients from the latter to the former.

```
RANDOMPEER()
    $S \leftarrow \{\ r \mid c_r \geq c_p \wedge r \in$ UNDERLOADED$\}$
    $q \leftarrow$ **null**
    **while** $(S \neq \emptyset \wedge q =$ **null**$)$
        $r \leftarrow \langle$pick a random node from $S\rangle$
        $S = S - \{r\}$
        $l_r \leftarrow \langle$request load from $r\rangle$
        **if** $(l_r < c_r \wedge (c_p < c_r \vee l_r > l_p)$      // Found
            $q \leftarrow r$
    **return** q

SENDSTATE($q$)
    $C \leftarrow \langle$select $min(c_q - l_q, l_p)$ local clients$\rangle$
    send $C$ to $q$

RETURNSTATE($q$):    send $\emptyset$ to $q$

UPDATE($C, q$)
    CLIENTS $\leftarrow$ CLIENTS $\cup\, C$
    **if** $(l_q == 0 \wedge l_p < c_p)$
        CLIENTS $\leftarrow$ CLIENTS $\cup \{q\}$
        $\langle q$ becomes a client$\rangle$
    **else if** $(\exists r \in$ CLIENTS $: c_r > c_q)$
        $\langle$transfer clients of $q$ to $r\rangle$
        CLIENTS $\leftarrow$ CLIENTS $\cup \{q\} - \{r\}$
        $\langle q$ becomes a client, $r$ becomes a server$\rangle$
```

Figure 3. The superpeer selection algorithm. Notations: p is the local peer; q and r are identifiers of the remote one; l_q denotes the number of clients of remote peer q, obtained by an explicit request; l_p denotes the local number of clients.

It is easy to see that in the absence of failures, this mechanism eventually produces the target topology. In fact, all superpeers continously try to discover nodes with larger capacity that are not completely utilized. These are selected from the union of the underloaded sets, that progressively shrinks until it becomes empty or reaches its minimum size.

So far, we have not discussed how to deal with failures. As discussed above, the lower layers based on NEWSCAST do not require any specific mechanism: crashed nodes cannot inject their descriptor in the network, so they are progressively forgotten. The only modification to be applied to the skeleton algorithm is the handling of timeouts: exchanges that are not completed inside a round are simply discarded.

In the case of the last protocol presented, the situation is slightly different. Superpeers and clients are responsible for periodically probing their counterparts, to detect failures. Faulty clients are simply removed from the CLIENTS set. When a client detects a faulty superpeer, it probes some of the nodes in UNDERLOADED to check whether they are willing to accept it as peer; if not, it becomes a superpeer and runs the protocol described here.

4 Experimental Results

To validate our approach, we have performed numerous experiments based on simulation. We were interested in three main questions: (i) what is the behavior of the protocol with respect to its parameters; (ii) what are the time and communication costs associated with its execution; and (iii) how robust the protocol is.

Two distinct settings have been analyzed, characterized by distinct distributions for the capacity parameter: power-law and uniform. In the power-law distribution, the probability $P[c_n = x]$ that a node n has

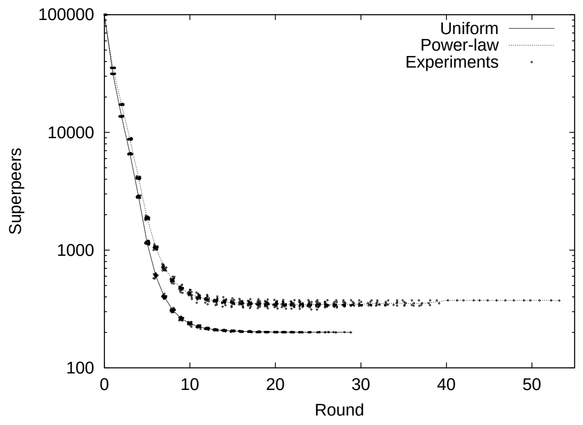

Figure 4. Total number of superpeers in the network after the execution of the specified number of rounds.

a capacity x, with x contained in a bounded range $[1, c_{max}]$, is equal to $x^{-\alpha}$, where α is the distribution exponent. In the uniform distribution, capacity is chosen uniformly from the range $[1, c_{max}]$. The former is motivated by prior measurements done over existing P2P networks [12], where most of the nodes have low capacity, while very few of them are able to support a large number of clients. Although not completely exact, the power-law distribution (with parameter $\alpha = 2$) provides a reasonable approximation of those results. The latter is proposed to double-check our results and show that they are not limited to a single distribution.

All experiments are performed using a custom-developed, round-driven simulator. In all figures, 20 independent experiments have been performed. Unless stated otherwise, most of the parameters are fixed in all experiments: the size of the network is 10^5 nodes; the maximum capacity c_{max} is 500; and the size s of partial views used in NEWSCAST is 30. All these values can be reasonably adopted or measured in realistic settings; yet, the behavior of the algorithm observed under variations of these parameters are analyzed in the following.

Figure 4 shows the behavior of our protocol over time. As initial configuration, we selected a topology that is the farthest from the target: a random topology where all nodes behave as superpeers, although none of them is responsible for any client. The curves represent the number of superpeers contained in the network after the specified number of rounds, averaged over 20 experiments. Individual experiments are shown; their x-coordinates has been shuffled with a small random increment to separate similar results. Although in some of the experiments several rounds are needed to reach the target topology (55 rounds in the worst case), the algorithm proves to be extremely fast, independently from the distribution considered: after less than 15 rounds, the resulting topologies approximate extremely well the target. We believe that such topologies are satisfactory for most applications (if not all).

The results shown in Figure 4 are relative to a network of fixed size (10^5 nodes). An interesting questions regards scalability. Figure 5 shows results about time complexity in networks with sizes comprised in the range $[10^3 - 10^6]$. Three different points in time are considered: (i) the time needed to reach the target topology; (ii) the time needed to reach a configuration where more than 80% of the total capacity possessed by superpeers is utilized by clients; and (iii) the same as (ii), with the threshold being 95%. Interestingly, the time needed to reach such utilization thresholds is independent from network size and located around 10 and 13 rounds, respectively. As above, the target configuration requires a larger number of rounds, that grows (approximately) logarithmically with respect to the size of the network. This is motivated by the fact that, after having reached an almost-optimal configuration, the problem of eliminating the few superpeers that should not be included in the target topology is equivalent to the epidemic broadcast of their identifiers, which requires logarithmic time.

Figures 6 and 7 show how the maximum capacity c_{max} and the size of partial views s exchanged by NEWSCAST influence the performance of our algorithm. Only the number of rounds needed to reach the 80% and 95% thresholds are plotted, for space reasons; the time need to obtain the target topology present

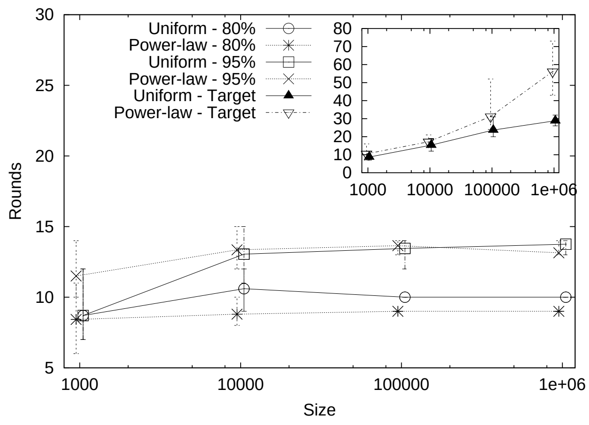

Figure 5. Number of rounds to obtain the 80%, 95% utilization thresholds and the target topology. Network size: $[10^3 - 10^6]$.

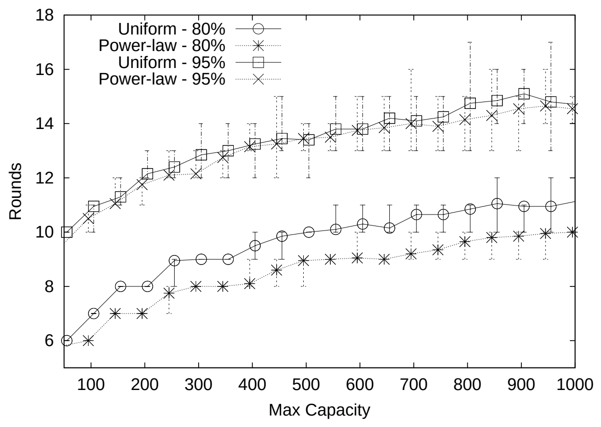

Figure 6. Number of rounds needed to reach the 80% and 95% utilization thresholds, with $c_{max} \in [50, 1000]$.

a similar behavior. In both cases, the results are as expected. The larger the capacity, the smaller is the set of superpeers included in the target topology, and the longer will be the number of rounds needed by the algorithm to reach such configuration. The curves, however, grow very slowly with capacity, peaking at 14 rounds when capacity is equal to 1000. Picking a larger view, on the other hand, implies a larger sample of nodes that are available for random selection, and thus better performance, but at the cost of larger communication costs.

So far, we have expressed our results in terms of rounds. The round length δ defines the time complexity of convergence. Choosing a short δ will result in proportionally faster convergence, but higher communication costs per unit time. The small number of rounds needed to obtain an almost-optimal configuration enables to choose large values for δ. For example, a value of 60 seconds enable the transformation of a random network in an almost-optimal superpeer topology in less than 15 minutes,, independently from the size of the network.

The overall communication cost of our algorithm is given by the total amount of messages exchanged by the multiple layers of our system. For the three sets managed directly by NEWSCAST, we inherit the good properties of that protocol. In NEWSCAST, the number of exchanges for each node per round can be described by the random variable $1 + \phi$ where ϕ has a Poisson distribution with parameter 1. Thus, on the average, there are two exchanges per node (one initiated and one coming from another node), with a very low variance. This means that these layers are all ”democratic”, requiring to send and receive few

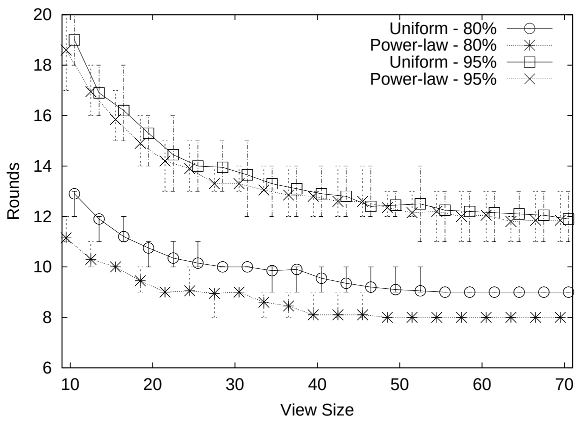

Figure 7. Number of rounds needed to reach the 80% and 95% utilization thresholds, with $c \in [10, 70]$.

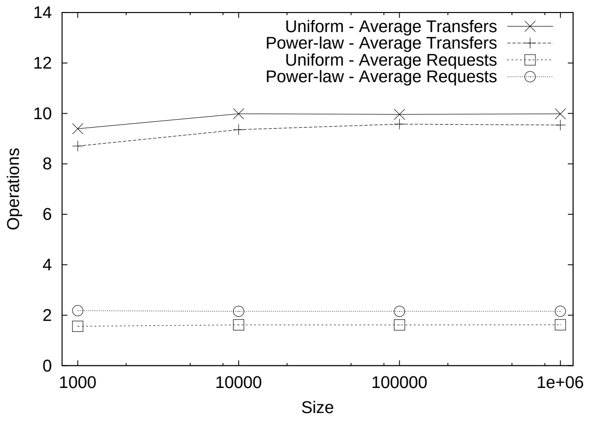

Figure 8. Total communication costs, averaged over all nodes of the network. Network size: $[10^3 - 10^6]$.

hundred bytes per round at all nodes. This is particularly important, because these layers are executed also by clients.

The fourth set, CLIENT, on the other hand, is managed just by superpeers. Two communication costs are to be considered: the total number of probes sent in function RANDOMPEER to discover the load of other superpeers, and the total number of client transfers performed. Figure 8 show such costs for networks of different size. To simplify the representation, the values have been averaged over the total number of nodes. Once again, the results are approximately independent from the size of the network, meaning that the total message complexity of our algorithm grows linearly with network size. The figure shows that only an average of two load probes are sent per node, while each client is transfered approximately 9 times. The latter may be considered a rather high value, considering the costs that may be associated with a client transfer. Yet, it is possible to use the algorithm to compute a plan of the distribution of clients, by transferring just their identifiers; and postpone the actual transfers later, after a predetermined number of rounds.

To complete our evaluation of communication costs, we have measured the maximum number of operations performed by a superpeer in a single round, divided by the capacity of the corresponding node. Figure 9 proves that none of the nodes is ever overloaded by communication costs.

The final figures demonstrate the robustness of our protocol. In Figure 10, a catastrophic scenario is shown: at round 30, 50% of the superpeers are removed. After the initial period when all clients whose superpeer has crashed become superpeer by themselves, the protocol behaves as usual and repair the overlay

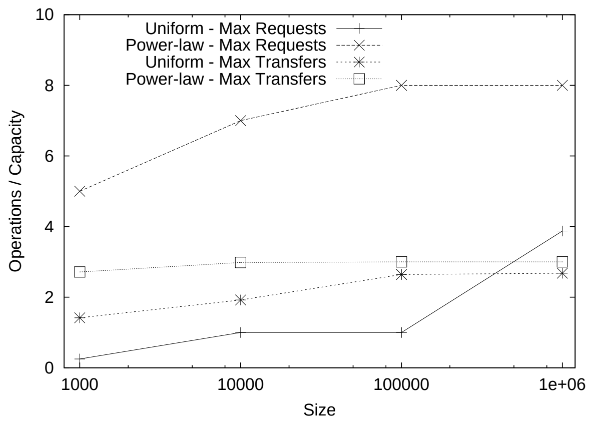

Figure 9. Maximum communication costs incurred by a single node in a single round. Network size: $[10^3 - 10^6]$.

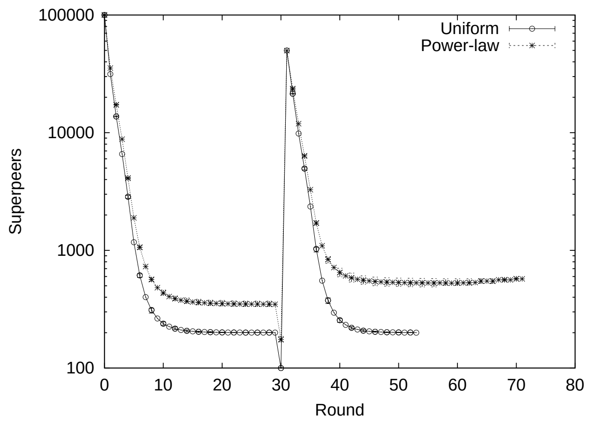

Figure 10. Catastrophic failure scenario. At round 30, 50% of the superpeers are removed from the network.

topology by selecting new superpeers among the remaining nodes. Figure 11 depicts a scenario where an increasing number of nodes are continuously substituted with new nodes. The plot has been obtained as follows. The algorithm has been run for several rounds, substituting a given percentage of nodes (drawn in the x-axis) at each round, producing an oscillating number of superpeers. Each point in the plot represents the number of superpeers at one of such rounds. As expected, the oscillation range grows with the percentage of substituted nodes; nevertheless, even for high values, such range is small.

5 Related Work and Conclusions

Heterogeneity of P2P networks can be leveraged by implementing topologies based on superpeers. Singh et al. have suggested the deployment of superpeers directly managed by content service providers [13]. In the mean time, popular file-sharing applications such as Kazaa [4] have explored the use of superpeers to improve search performance. Details of the Kazaa protocols and code are not publicly available, making it hard to draw comparisons to our work.

Yang and Garcia Molina et al. [15] have discussed general issues regarding the design of superpeer networks. Their main focus is on centralized design of such networks. A simple decentralized mechanism is proposed, based on superpeers splitting their clusters when overloaded, but they do not present any experimental result for it.

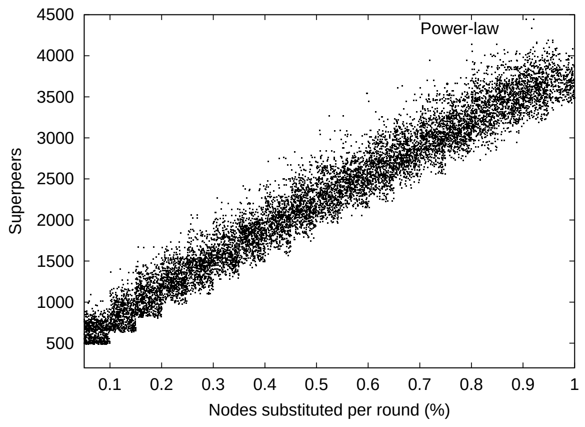

Figure 11. Behavior of the protocol with an increasing percentage of nodes are substituted at each round.

On the contrary, this paper has presented a novel approach for the construction of superpeer overlays. The resulting protocol is extremely efficient and robust, being able to tolerate both a continuous flux of nodes joining and leaving the system, and catastrophic scenarios where any percentage of the superpeers are removed from the system.

Future work will include exploring the dimensions that have not been considered here, including communication failures, locality awareness and more complex topologies with replicated superpeers.

References

[1] F. Dabek et al. Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service. In *Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, Schloss Elmau, Germany, May 2001. IEEE Computer Society.

[2] A. Demers et al. Epidemic Algorithms for Replicated Database Management. In *Proc. of 6th ACM Symp. on Principles of Dist. Comp. (PODC'87)*, Vancouver, August 1987.

[3] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié. From Epidemics to Distributed Computing. *IEEE Computer*, 21(4):341–374, Nov. 2003.

[4] Fasttrack Home Page. http://www.fasttrack.nu.

[5] I. Foster and C. Kesselman. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann, 1999.

[6] M. Jelasity, W. Kowalczyk, and M. van Steen. Newscast computing. Technical Report IR-CS-006, Vrije Universiteit Amsterdam, Dept. of Computer Science, Nov. 2003.

[7] M. Jelasity and A. Montresor. Epidemic-Style Proactive Aggregation in Large Overlay Networks. In *Proc. of the 24th International Conference on Distributed Computing Systems (ICDCS 2004)*, Tokyo, Japan, 2004.

[8] G. Kan. Gnutella. In A. Oram, editor, *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, chapter 8. O'Reilly & Associates, Mar. 2001.

[9] J. Kubiatowicz et al. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, Nov. 2000.

[10] D. S. Milojicic et al. Peer-to-Peer Computing. Technical Report HPL-2002-57, HP Labs, Palo Alto, 2002.

[11] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proc. of the 18th Int. Conf. on Distributed Systems Platforms*, Heidelberg, Germany, Nov. 2001.

[12] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.

[13] S. Singh et al. The Case for Service Provider Deployment of Super-Peers in P2P Networks. In *Proc. of the Workshop on Economics of P2P Systems*, Berkeley, California, June 2003.

[14] J. Smed, T. Kaukoranta, and H. Hakonen. Networking and Multiplayer Computer Games–The Story So Far. *International Journal of Intelligent Games & Simulation*, 2(2), 2003.

[15] B. Yang and H. Garcia-Molina. Designing a Super-peer Network. In *Proc. of the 19th Int. Conf. on Data Engineering (ICDE)*, Bangalore, India, Mar. 2003.

[16] B. Zhao et al. Tapestry: A Resilient Global-scale Overlay for Service Deployment. *To appear in IEEE Journal on Selected Areas in Communications*, 2003.