The Necessity of Supernodes

David Hadaller, Kevin Regan, Tyrel Russell

Abstract

Peer-to-Peer (P2P) networks have grown to such a massive scale that performing an efficient search in the network is non-trivial. Systems such as Gnutella [1] were initially plagued with scalability problems as the number of users grew into the tens of thousands. As the number of users has now climbed into the millions, system designers have resorted to the use of supernodes to address scalability issues and to perform more efficient searches.

While the use of supernodes has become commonplace in unstructured P2P networks, no theoretical bounds on search time have been established. We analyze search time and show that supernodes are required for efficient search. We also formulate the optimal reduction in search time that can be achieved by using supernodes and show a construction where search time of $O(loglogn)$ is realized.

1 Introduction

Peer-to-Peer networks have recently ballooned in popularity and currently use more bandwidth on the Internet than any other application [5]. File sharing P2P systems such as Kazaa [2] have millions of users sharing petabytes of data. The main problem which these systems must address is efficiently locating data. It is non-trivial to find an approach to perform an efficient search in such a large and diverse network. Determining theoretical bounds on search time would be an asset to those designing future P2P networks.

Supernodes are often used in unstructured P2P networks to improve search time and scalability. While their use has been empirically shown to improve search time (e.g. by Chawathe et al. [4]), no theoretical analysis has been done on the use of supernodes in unstructured networks.

In this work, we show that supernodes are necessary for efficient search. We proceed by first establishing a lower bound on the worst-case search time in a network without supernodes. We then construct a graph which illustrates that with supernodes, $O(loglog(n))$ search is possible. Additionally,

we establish a lower bound on the benefit of using supernodes in a network, and show that our construction is near optimal.

2 Problem Statement

We want to show that supernodes are necessary for efficient search. To show this, we first show that without supernodes, we cannot have efficient search. We then show that with supernodes, efficient search is possible.

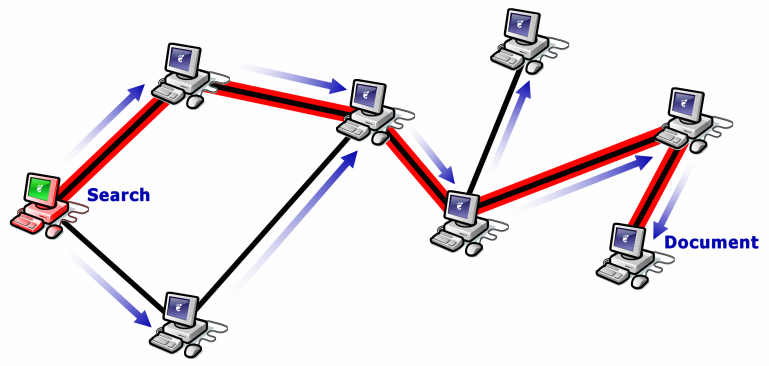

Figure 1: Worst case search time in a P2P network

2.1 Definitions

Search Time The number of hops required to reach a node containing the desired document using a flood search.

Worst-Case Search Time The maximum possible search time in a particular P2P network, where there is only one copy of the desired document and the search node and the document node are located as far apart as possible. Figure 1 is an example.

Efficient Search Searching in a P2P network with n nodes is efficient if the worst-case search time is $O(loglogn)$.

Ordinary Node A typical node in a P2P network, having a small number of neighbours (e.g. 6)

Supernode A node in a P2P network with a larger number of neighbours (e.g. 100). See figure 2 for an example.

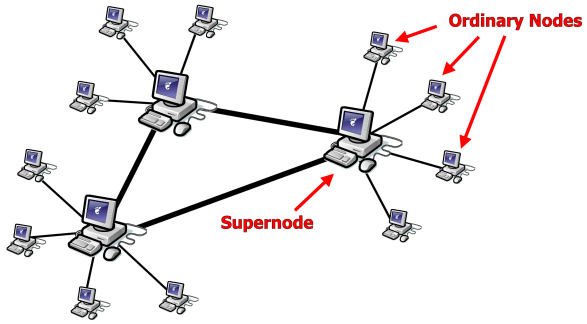

Figure 2: A P2P network with supernodes

Search time depends heavily on the topology of the P2P network. This can be seen by making a few simple observations about search time using pathological topologies.

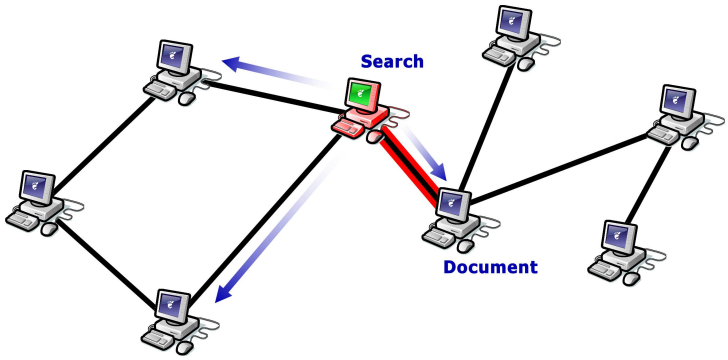

Figure 3: Best case search time in a P2P network

Best Case Search Time The search time in the best case for any network topology is the case where the desired document is one hop away from the search node. This would be a search time of 1. See Figure 3.

Worse Case Search Time with Worse Topology The search time in the worst case is the case where the nodes in the network are arranged in a single path and the search node is the first node in the path and the document node is the last node in the path. This would be a

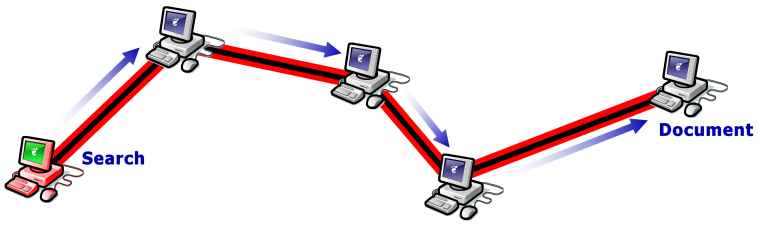

Figure 4: Worst case search time in a P2P network with the Worst Topology

search time of $n-1$, where n is the number of nodes in the network. See Figure 4.

The topology of the network directly affects the worst-case search time. In the following section, we define the problem of determining a lower bound on the worst-case search time.

2.2 Lower Bound on Search Time without Supernodes

It has been shown experimentally in the literature that peer-to-peer networks without supernodes are inefficient and scale poorly to a large number of nodes [4, 7]. We approach this problem analytically with the goal of showing that there is a lower bound on worst-case search time in a P2P network without supernodes.

2.2.1 Approach

In order to show that supernodes are required to achieve efficient search, we analyze the search time in P2P networks without supernodes. Since a supernode is a node with a large number of neighbours, we can eliminate supernodes by placing a limit on the maximum number of neighbours a node is permitted to have.

Goal: In a P2P network with n nodes and some bound on the number of neighbours $f(n)$ (e.g. $f(n) = logn$), we wish to show a lower bound on the worst-case search time.

Since worst-case search time is directly dependent on the topology of the P2P network, this translates into a problem of: given n nodes and some bound on the maximum number of neighbours, what is the lowest possible worst-case search time for any network topology.

To show this lower bound, we take an adversarial approach. Consider a game between a network designer and a document placer. Given a network topology, the job of the document placer is to choose the search node and the document node to be as far apart as possible (as is the case in Figure 1. The game begins with the network designer first designing the topology of the network, within the constraints of the problem. The document placer then takes his turn and chooses the search node and the document node to be as far apart as possible. The number of hops between these nodes is the score, and the network designer wins the game when the the score is as low as possible.

Taking this approach, the document placer is creating the worst search possible and the network designer is attempting to construct the network with least search time. If the network designer succeeds in designing a network with lowest possible worst-case search time (within the constraints of the problem), a lower bound on search time has been shown.

We now reformulate this problem into graph theory in order to determine this lower bound (see section **??**).

2.3 Graph Theory Representation

2.3.1 Definitions

Maximum Degree We denote Δ as the maximum degree over all vertices of a graph.

Minimum (u,v)-path We denote the minimum path between two vertices $u, v, u \neq v$, as $d(u, v)$.

Diameter We define diameter $\mathcal{D}$ as max $\{d(u, v) : \forall\, u, v \in \mathcal{V}\}$

2.3.2 Representation

Consider the graph representation of a P2P network. The peers are represented by vertices in the graph and neighbour-to-neighbour connections are represented by an edge connecting two vertices in the network. In order to eliminate supernodes, we bound the maximum degree of any node to Δ. The worst-case search time in a P2P network is equivalent to the diameter of the graph.

In graph theory terms, the problem becomes: Given a graph $\mathcal{G}$ with n vertices and maximum degree Δ, can we bound the diameter of the graph $\mathcal{D}$ in terms of $n = |\mathcal{V}|$ and Δ?

Section **??** shows our lower bound.

2.4 Assumptions

2.4.1 Search time Metric

In our work, we define search time to be the number of hops from the search node to document node. However, in a P2P network, other metrics may seem desirable. For example, it is important to consider the total number of messages generated by a flood search. It is also important to know the number of nodes which must perform a local search for the desired document.

While both of the above metrics are important in P2P implementations, they are inapplicable for our theoretical analysis of lower bounds on search time. To establish a lower bound on worst-case search time, we are considering the adversarial worst-case where there is just one desired document and it is placed as far from the search node as possible. In this case, because the search proceeds with no knowledge of where the document is (i.e. the network is unstructured), the entire network will have to be searched in the worst-case. Therefore, the number of nodes searched will always be n, for a network with n nodes. As well, minimizing the total number of messages conflicts with the goal of efficient search, since reducing the number of messages could possibly reduce the number of nodes that can be reached by a search.

We are interested in retrieving the search results as quickly as possible. Having every search flood the entire network is impractical. However, since we seek to show a lower bound on the worst-case search time, we must assume the case where every node is working to return the results as quickly as possible. Therefore, it can be seen that the time to retrieve the search results can only increase as the number of hops to the desired document increases. For example, it will necessarily take longer to retrieve the results if the document is four hops away from the search node than if the document were two hops away. Thus, reducing the diameter of the network necessarily improves worst-case search time.

2.4.2 Relation to DHTs

Two classes of P2P networks have emerged: structured and unstructured. A structured P2P network is one which imposes constraints on which nodes contain which documents. These are typically implemented as distributed hash tables (DHTs), for example Chord [9] and CAN [6]. Unstructured networks such as Gnutella [1] or Kazaa [2] do not have such constraints and documents are kept at their originating node.

The main benefit that a structured P2P network provides is that a search

proceeds with the knowledge of the location of the desired document. For example, in Chord, every hop of a query necessarily takes it closer to the document. This is possible because documents are placed in the network in a deterministic fashion. In the case of an unstructured network, there is no notion of document placement, and a flood-style search is the best that can be achieved.

Although DHTs provide numerous benefits, unstructured systems are known to provide more efficient lookup times for popular documents. For example, Loo et al. [3] suggest a hybrid flooding and DHT solution where flooding is used to locate popular documents and a DHT is used to locate rare documents. DHTs also require a large amount of maintenance overhead to maintain uniform distribution of documents. For these reasons, DHTs have yet to be adopted for general-purpose large-scale file-sharing applications and, therefore, we focus our analysis of search time in unstructured networks.

2.5 Effect of Adding Supernodes

Once we have established a lower bound on worst-case search time without supernodes, we show that with supernodes we can achieve efficient search. Using supernodes in unstructured P2P network has become common practice, especially in file sharing systems, such as Gnutella [1] and Kazaa [2].

The initial motivation behind using supernodes in a P2P network was to improve scalability [4]. Supernodes also add stability to the network and reduce overall execution time of the search. This is due to the observed heterogeneity in P2P networks; in particular, certain peers exhibit server-like characteristics. For example, Sariou et al. [8] found that 7% of Gnutella users reported bandwidths of T1 or greater, and that about 10% of sessions last more than 5 hours each. The existence of such nodes illustrates that it is not unreasonable to assign a small set of nodes more load and responsibility.

These experimental observations support our theoretical construction of a P2P network with supernodes, done in section 4.

3 Search without Supernodes

We desire to reduce worst-case search time in the P2P network by reducing the diameter of a equivalent graph. However, there is a limit to how small the diameter of a graph without supernodes can be. We proceed to give an analysis of how diameter is limited by the degree and number of nodes in a graph.

3.1 Bounds on Diameter

E.F. Moore established the following bound on the number of nodes in a graph given a diameter of $\mathcal{D}$ and a maximum degree of Δ, which holds for all $\Delta \geq 3$:

$$\begin{aligned} n &\leq 1 + \Delta\left\{1 + (\Delta - 1) + \cdots + (\Delta - 1)^{\mathcal{D}}\right\} && (1) \\ &\leq \frac{\Delta(\Delta - 1)^{\mathcal{D}} - 2}{\Delta - 2} && (2) \end{aligned}$$

We can express Equation 2 in terms of $\mathcal{D}$ to bound the diameter from below.

$$\mathcal{D} \geq \frac{log\left[\frac{n(\Delta-2)+2}{\Delta}\right]}{log(\Delta - 1)} \tag{3}$$

If we fix Δ, then we have the following bound on diameter in terms of n:

$$\mathcal{D} \in \Omega(logn) \text{ for } \Delta \text{ fixed to some constant} \tag{4}$$

Now consider if we were to allow Δ to grow as a function of n. If we allow Δ to grow no larger than $\log n$ we achieve the following bound on diameter:

$$\mathcal{D} \in \Omega\left(\frac{logn}{loglogn}\right) \text{ for } \Delta \in O(logn) \tag{5}$$

The derivation of this bound is left to appendix A. Moving from nodes of fixed degree to nodes of degree at most $logn$ only reduces the best case diameter of the graph from $logn$ to $\frac{logn}{loglogn}$. We can conclude that we cannot achieve our efficient search aim of $loglogn$ diameter without nodes having degree significantly higher than $logn$ (i.e. supernodes). Section 4 will give a construction of a graph which achieves loglogn diameter with a small set of supernodes, but first we note a simple graph that achieves diameter near $logn$.

3.2 Graph with Near $logn$ Diameter

A rooted balanced tree of n nodes where the branching factor is $\Delta - 1$ will have a depth of $log_{\Delta-1}n$. The longest path between any two nodes in the graph will use nodes in the leaves of the tree. This path will stretch from the root to each leaf, resulting in a diameter of $2 \cdot log_{\Delta-1}n$. So, while we

cannot expect to achieve diameters smaller than $logn$ with a fixed Δ, it is reasonable to construct graphs with near $logn$ diameter.

4 Search with supernodes

We can construct a graph with $O(loglogn)$ diameter by first defining the graph without supernodes as a set of $\frac{n}{logn}$ clusters of size $logn$. Given the results from the previous section we can reasonably expect to achieve diameter of $O(loglogn)$ in each cluster, since each cluster has only $logn$ nodes. However, the diameter of the entire graph will be at best $\Omega(logn)$.

We can significantly decrease the diameter by incorporating a single supernode. To do so, we connect the supernode to each of the $\frac{n}{logn}$ clusters. Now, we can imagine the longest distance between two nodes u, v in our graph will occur when u and v and are in separate clusters. In this case the path will involve traversing at most $loglogn$ edges to go from u to the node in the cluster connected to the supernode, then traversing one edge to the supernode and one edge into the cluster containing u, then at most another $loglogn$ edges to reach u. This gives us a diameter of $2 \cdot loglogn + 2$. Thus, we can achieve a diameter of $O(loglogn)$ with the addition of a singlenode. However, since our supernode must be attached to each cluster our single supernode will have degree $\frac{n}{logn}$.

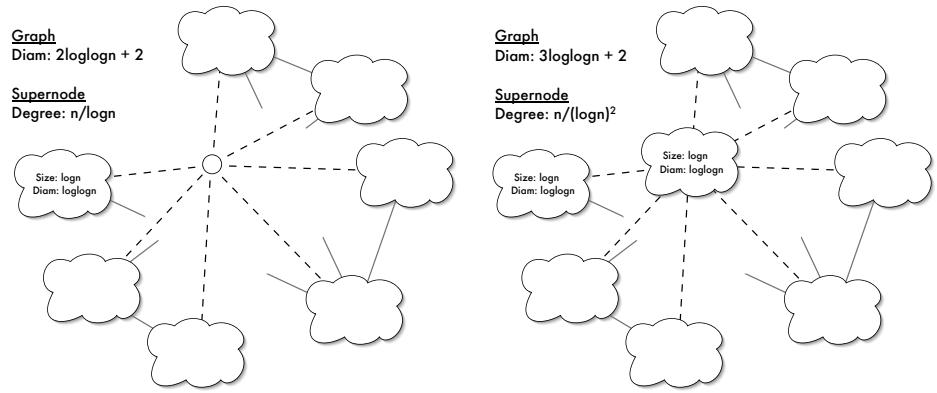

Figure 5: Constructions using single supernode and supernode cluster

We can use multiple supernodes to distribute the number of clusters each

supernode must be attached to. If we construct a cluster of $logn$ supernodes, we can again reasonably expect to achieve a diameter of $loglogn$ within this cluster. We can attach this supernode cluster to the rest of the original graph by attaching each individual supernode to $\frac{n}{logn \cdot logn}$ clusters. The longest distance between our u, v nodes will now pass through the cluster containing u, the supernode cluster and the cluster containing v, leading to a diameter of $3 \cdot loglogn + 2$ with supernodes of degree $\frac{n}{(logn)^2}$. It should be noted that we can further reduce degree of the supernodes by a factor of k by increasing the number of supernodes in the cluster by a factor of k. If we restrict this k value to a constant, then the diameter of the resulting graph, $(2 + k) \cdot loglogn + 2$, remains $O(loglogn)$.

5 Lower Bounds on Diameter with Supernodes

We will analyze how much a supernode can improve the diameter over graphs with a small maximum degree. We will show the that for graphs of small diameter a supernode will not affect the diameter of the graph. We will also show that the diameter of a graph with a supernode can be bounded below by a function in terms of the number of nodes n, the degree of the supernode k and the maximum degree of the regular nodes Δ.

First, we will show the simple case where adding a supernode does not affect the diameter of the graph.

Lemma 1. *Adding a supernode to a graph with a diameter of less than or equal to 2 will not reduce the diameter.*

Proof of Lemma 1. *This can be seen easily by noting that the path between any two nodes attached to a supernode is 2. This means that adding a supernode will not reduce the path length between any node.*

□

Next, we show that for graphs of diameter 3 adding a supernode will only reduce the diameter of the graph under specific conditions.

Lemma 2. *Adding a supernode to a graph with a diameter of 3 will only reduce the diameter of the graph if the supernode is attached to every pair of nodes where the distance between the nodes is 3.*

Proof of Lemma 2. *If the supernode is attached to every pair of nodes with distance 3 then the diameter of the graph will be at most 2. This is due*

to the fact that all nodes which used to have distance of 3 can reach each other through the supernode in 2 hops. If there exists a pair of nodes with distance 3 and the supernode is only attached to one of the pair, then even if the supernode is attached to every other node, the diameter will be 3. This is because the path with distance 3 is still the shortest path since traversing the supernode takes two hops and the node which is not attached requires one hop to reach the supernode.

□

Now, we show that the diameter of the graph is bounded when adding a single supernode.

Theorem 1. *If there exists a supernode with degree k in a graph G with n nodes, degree Δ, and the diameter of the subgraph not containing the supernode is $d \geq 4$ then the diameter of the entire graph d_n is*

$$d_n = min(2d_e + 2, d_l + 2), \text{ where} \tag{6}$$

$$d_e \geq \frac{log(\frac{(\frac{(n-k)}{k})(\Delta-2)+2)}{\Delta})}{log(\Delta-1)}$$

$$dl_l \geq \frac{log(\frac{(n-k)(\Delta-2)+2)}{\Delta})}{log(\Delta-1)}$$

Proof. see appendix □

From Theorem 1, we have shown that the diameter of the graph is bounded by a function of the number of nodes, the degree of the supernode and the degree of the ordinary nodes in the graph. We will now extend this result with several corollaries.

The first of these results shows that we can extend the single supernode case in to the m supernode case easily using the results obtained from Theorem 1.

Corollary 1. *The diameter of a graph with m supernodes of degree k is equal to*

$$d_n = min(2d_e + d_{sn} + 2, d_l + d_{sn} + 2) \tag{7}$$

where $d_e \geq \frac{log(\frac{(\frac{(n-mk)}{mk})(\Delta-2)+2)}{\Delta})}{log(\Delta-1)}$ and $d_l \geq \frac{log(\frac{(n-mk)(\Delta-2)+2)}{\Delta})}{log(\Delta-1)}$ and d_{sn} is the diameter of the supernode graph.

Proof of Corollary 1. *This proof follows from Theorem 1. First, we observe that the problem of finding the diameter of the graph is the same, since we must find the two trees of maximal depth which are the smallest given the number nodes in the graph. In this case, we now have have mk nodes attached to supernodes but if we substitute this value for k in Theorem 1 then we can use the results from Theorem 1. The only change that must be made is that instead of crossing a supernode in 2 hops, they must also traverse the supernode graph which has a diameter, d_{sn}, by definition. Therefore, using Theorem 1 and the diameter of the supernode graph, the diameter of the entire graph is,*

$$d_n = min(2d_e + d_{sn} + 2, d_l + d_{sn} + 2) \tag{8}$$

□

From Section 4, we show that graphs can be constructed with $O(loglog(n))$ diameter with supernodes. We will now show that for those graphs, this result is near optimal in terms of diameter and degree.

Corollary 2. *The construction from Section 4 has near optimal diameter.*

Proof of Corollary 2. *This can be observed by applying the results from Corollary 1. We observe that the construction has $log(n)$ supernodes and each node has a degree of $\frac{n}{(log(n))^2}$. We hypothesize that our construction can achieve a diameter equal to*

$$2\frac{log(\frac{(n(\Delta-2)+2)}{\Delta})}{log(\Delta-1)} + d_{sn} + 2$$

From Corollary 1, we know that the diameter of the graph, d_n, will be equal to

$$d_n = min(2d_e + d_{sn} + 2, d_l + d_{sn} + 2) \tag{9}$$

Therefore we substitute the values for m and k into the equations for d_e and d_s to show that the diameter is near optimal.

$$
\begin{aligned}
d_e &\geq \frac{log(\frac{(\frac{(n-mk)}{mk})(\Delta-2)+2)}{\Delta})}{log(\Delta-1)} \\
&= \frac{log(\frac{(n-(logn)\frac{n}{(logn)^2})(\Delta-2)+2)}{\Delta})}{log(\Delta-1)} \\
&= \frac{log(\frac{(n-\frac{n}{(logn)})(\Delta-2)+2)}{\Delta})}{log(\Delta-1)} \\
&= \frac{log(\frac{(logn-1)(\Delta-2)+2)}{\Delta})}{log(\Delta-1)} \qquad (10)
\end{aligned}
$$

and

$$
\begin{aligned}
d_l &\geq \frac{log(\frac{(n-mk)(\Delta-2)+2)}{\Delta})}{log(\Delta-1)} \\
&= \frac{log(\frac{(n-(logn)\frac{n}{(logn)^2})(\Delta-2)+2)}{\Delta})}{log(\Delta-1)} \\
&= \frac{log(\frac{(n-\frac{n}{logn})(\Delta-2)+2)}{\Delta})}{log(\Delta-1)} \qquad (11)
\end{aligned}
$$

As n grows, $2loglog(n)$ *will be smaller than* $log(n-\frac{n}{logn})$*. Therefore,* d_n *is minimal when the value is* $2d_e + d_{sn} + 2$ *where* d_e *is the equal to Equation 10. Therefore, the diameter of our construction is almost the same as the optimal diameter.*

□

Corollary 3. *To achieve diameter of* $O(loglogn)$ *the degree of the supernodes is bounded by* $\Omega\left(\frac{n}{(logn)^2}\right)$

Proof of Corollary 3. *If the diameter is* $\frac{log(\frac{(logn)(\Delta-2)+2)}{\Delta})}{log(\Delta-1)}$ *and from Corollary 1 the diameter of the graph will be* $\frac{log(\frac{(n-mk)(\Delta-2)+2)}{\Delta})}{log(\Delta-1)}$ *or* $\frac{log(\frac{(\frac{(n-mk)}{mk})(\Delta-2)+2)}{\Delta})}{log(\Delta-1)}$*. Setting the hypothetical diameter equal to the two lower bound results and reducing, we obtain,*

$$
log(n) = n - mk \qquad (12)
$$

or

$$log(n) \quad = \quad \frac{n - mk}{mk} \tag{13}$$

where m equals $log(n)$. Since the value of k is less for Equation 13, we find that $k = \frac{n}{(logn)^2}$

□

6 Conclusion

In this work, we have shown that supernodes are necessary for efficient search. We first established a lower bound on the worst-case search time in a network without supernodes. We then constructed a graph which illustrates that with supernodes, $O(loglog(n))$ search is possible.

Additionally, we established a lower bound on the benefit of using supernodes in a network, and show that our construction is near optimal. We show that to achieve efficient search with m supernodes, the degree of the supernodes must be $\Omega\left(\frac{n}{log(n)\cdot m}\right)$.

With these results, we can show that supernodes are necessary in P2P networks for efficient search.

A Diameter bound given $\Delta < logn$

When $\Delta < logn$, we can use the Moore bound as follows, to derive a bound on diameter simply in terms of n

$$\begin{aligned}
\mathcal{D} &\geq \frac{log\left[\frac{n(\Delta-2)+2}{\Delta}\right]}{log(\Delta-1)} \\
&\geq \frac{log\left[\frac{n\cdot(logn-2)+2}{logn}\right]}{log(logn-1)} \\
&\geq \frac{log\left[n\cdot(logn-2)\right] - loglogn}{log(logn-1)} \\
&\text{since } 1/2\cdot logn < logn - 2 \text{ when } n > 16 \\
&\geq \frac{log\left[n\cdot(\frac{1}{2}\cdot logn)\right] - loglogn}{loglogn} \\
&\geq \frac{logn - 1}{loglogn}
\end{aligned} \tag{14}$$

Thus we can say that when $\Delta < logn$, our diameter $\mathcal{D} \in \Omega\left(\frac{logn}{loglogn}\right)$.

B Proof of Theorem 1

Proof. *Given the set of nodes n, we would like to construct the graph with the smallest diameter with k nodes attached to a supernode. If the supernode s is attached to k nodes then there exists $n-k$ nodes which are not connected to the supernode. Since it requires one edge to reach a supernode and one edgee to return to a node, the maximum distance between any two nodes attached to supernodes is 2. The diameter of the entire graph is the sum of the two largest paths from a node attached to the supernode to its farthest leaf plus the 2 for the traversal across the supernode. This assumes that there are no branches traversing from a subgraph containing a node attached to the supernode to another subgraph with the same property except through the supernode. It will be seen that this assumption does hold due to the way in which we construct the subgraph.*

Following the intuition that drives the Moore Bound, the length of a path from a root node to a leaf is always minimal when every node in the subgraph has as many children as possible given its maximum degree. Since we wish to have as many children as possible, we will construct the graph so that each node only has one parent and therefore we have a complete tree. We can now characterized the diameter of our graph as the the sum of the depths of the two largest trees plus 2 edges which traverse across the supernode. There are two different cases which need to be addressed. Either the nodes which form a the two maximal trees are from two seperate trees or they are from a single tree.

We will discuss the case where there exists two maximal trees which make up the diameter of the graph. The idea is to minimize every tree simultaneously. This is done by spreading the nodes evenly between the k attached nodes and forcing them to form complete balanced trees.

Lemma 3. *If there exists a set of m nodes and they are attached to a root node, then the path length from the root node to the deepest leaf is minimal when the nodes form a complete tree.*

Proof of Lemma 3. *The proof follows by induction. Let $m = 0$ then the tree is complete and the path from root to leaf is minimal. Now we assume that when $m = k$ the path is minimal when nodes from a complete tree. Now,*

if $m = k+1$ then the one more node must be placed on the tree. If all of the leaves of complete tree with k nodes have a depth of h then adding a single node to any leave will create a complete tree of depth $h+1$ where the path length to the new node is minimal. If some of the leaves of the complete tree with k nodes have depth $h-1$ then placing the node at a node with depth $h-1$ will create a new complete tree with $k+1$ nodes and the path from the root to the deepest node is h which is minimal since it was the minimal path for the tree with k nodes. However, if the new node is placed on a node with depth h then the tree will no longer be a complete tree and the path length will be $h+1$ which is not minimal. So the lemma follows by induction.

□

Using Lemma 3 we know that the the depth of a tree is minimal when the tree is complete for a set of nodes m. We also know that the size of the tree grows with the number of nodes so we form k trees with an equally distributed number of nodes $\frac{n-k}{k}$. If $\frac{n-k}{k}$ nodes form a complete tree, then each tree has (at most) $\Delta(1+(\Delta-1)+\cdots+(\Delta-1)^{d_t-1})$ nodes, where d_t is the tree depth. This is the Moore Bound less one node since the root of the tree is actually part of the k nodes attached to the supernode. So, by the Moore Bound,

$$\frac{n-k}{k} \leq (\frac{\Delta(\Delta-1)^{d_e}+2}{(\Delta-2}) - 1 \tag{15}$$

Where d_e is defined to be the depth of the tree. This equation can be rearranged to show a bound on the depth of this tree in terms of the number of nodes and the maximum degree of the nodes.

$$d_e \geq \frac{log(\frac{(\frac{(n-k)}{k})(\Delta-2)+2)}{\Delta})}{log(\Delta-1)} \tag{16}$$

Lemma 4. *The diameter must stay the same or increase by removing nodes from any trees of depth d_e and placing them on at least two other trees.*

Proof of Lemma 4. *The proof for the lemma is straight forward. If nodes are removed from the any tree then the depth of that tree will either stay the same or decrease. The depth of this tree will only decrease if the number of nodes at depth d_e is less than the number of nodes removed. If a node is added to a tree of depth d_e then the depth of that tree will always stay the same or increase. Since we are adding the nodes to two trees of depth d_e, we know that the depth of these trees will either stay the same or increase.*

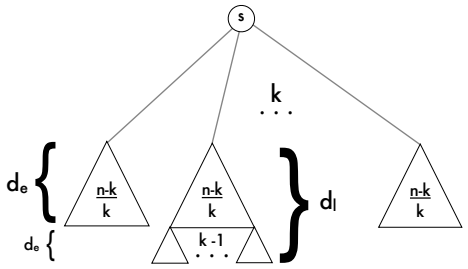

Figure 6: Diameter of large tree and evenly distributed trees

Since our diameter is the sum of the two maximal trees, we know that the diameter of this graph where nodes are being added to at least two trees will always increase or stay the same since the sum of the depths has the same property.

□

Using Lemma 4, the diameter of the graph is minimal when the nodes are evenly distributed for the all cases where nodes are rearranged to more than one tree. This leaves a case where nodes are only added to a single tree. Since the number of nodes grows exponentially with the depth of the tree, it can be shown that in specific cases the depth of the single tree is less than the depth of two trees.

Let all node removed from $k-1$ trees be added to a single tree. Let the depth of the single tree be d_l and the largest of the $k-1$ remaining trees be d_s. Since all nodes removed from trees of depth d_e are added to only d_l, the value of $d_s \leq d_e$ for all trees. Since d_s is the depth of the largest remaining tree, nodes should be removed evenly because no advantage is gained by removing all nodes from a single tree since this does not change d_s.

We will start by showing the simple case and then proving that the general case is bounded by the simple case. First, we let $d_s = 0$ and all $n-k$ nodes are part of a single tree. Since the tree of depth d_l is constructed by adding nodes to a tree of depth d_e then $d_e \leq d_l$. Each leaf on the trees of depth d_e have a max degree Δ and has at most $(\Delta - 1)$ children. There were $(k-1)$ trees with Δ branches with $1 + (\Delta - 1) + \cdots + (\Delta - 1)^{d_e - 1}$ nodes on

each branch. Observe that the trees with depth d_e have $\Delta(\Delta-1)^{d_e-1}$ leaves, which can have at most $(\Delta-1)$ children each. Therefore, we cannot just simply add the trees with Δ branches to the edges of the leaves to create a large tree. A single branch must be pruned from the trees and this forms $(k-1)$ branches. These branches can be formed to create $\frac{(k-1)}{(\Delta-1)}$ new trees that can be added. This gives a total number of $(k-1)+\frac{(k-1)}{(\Delta-1)}$ sub-trees which can be added to the leaves of the large tree. Since the depth of the large tree is the sum of the base tree, d_e, and the sub-trees, d_e, then the depth of the large tree is smaller than or equal to 2 trees of depth d_e only when the number of sub-trees to be added is less than or equal to the number of leaves in the large tree. More formally, if $\Delta(\Delta-1)^{d_e-1} \geq (k-1)+\frac{(k-1)}{(\Delta-1)}$ then $d_l \leq 2d_e$. Our tree is constructed by evenly distributing the nodes at each level and, if there are leaves to which a subtree is not fixed, the nodes at the lowest depths should be distributed to those leaves to balance the tree. The depth of the tree after these nodes are arranged will always be the same as or less than $2d_e$.

This leaves the more general case of the problem where there exists a tree with depth d_s such that $0 < d_s < d_e$. There can be $k-1$ trees of depth d_s without affecting the diameter of the entire graph and each of these trees has $\Delta(1+(\Delta-1)+\cdots+(\Delta-1)^{d_s-1})$ nodes. Lets assume that there exists $k-1$ trees with depth d_s and a large tree with depth d_l such that $d_s+d_l \leq 2d_e$. If it is possible to remove the nodes from all of the $k-1$ small trees and distribute them onto the large tree without breaking the inequality then it is sufficient to construct only the single large tree.

Lemma 5. *If $(\Delta-1)^{d_l-d_s+1} \geq (k-1)$ then adding the leaves from the $k-1$ trees with depth d_s to the large tree with depth d_l will only increase the depth of the large tree by at most 1.*

Proof of Lemma 5. *This is true if the number of leaves in the $(k-1)$ small trees is less than the number of leaves at the bottom of the tree with depth d_l. If there are $k-1$ trees of size d_s, then there are $(k-1)\Delta(\Delta-1)^{d^s-1}$ leaves at depth d_s. There are $\Delta(\Delta-1)^{(d_l)}$ possible children for the leaves in the large tree. Therefore, $(k-1)\Delta(\Delta-1)^{d^s-1} \leq \Delta(\Delta-1)^{d_l}$ must hold. This can be simplfied to $(k-1) \leq (\Delta-1)^{d_l-d_s+1}$. So if the inequality is true then the number of leaves in the $k-1$ small trees is less than the number of leaves in the large tree and adding these nodes to that tree will only increase the depth by at most that new row of nodes.*

□

From Lemma 5, we can see that adding new leaves to the large tree from all of the small trees only increases the depth of the large tree by the height of the row removed when $(\Delta-1)^{d_l-d_s+1} \geq (k-1)$*. This means that the value of the tree with all the nodes must have a depth less than or equal to the sum of the depths of a tree with depth* d_s *and a tree with depth* d_l*. Therefore when the inequality holds, it is sufficient to only have a single large tree with all* $n-k$ *nodes. Now we are left with the case where the inequality does not hold, it can be shown that this case collapses to the evenly distributed case. We will first present a lemma that will help us show this case.*

Lemma 6. *If* $(\Delta-1)^{d_l-d_s-1} \leq (k-1)$ *then adding the leaves from the large tree with depth* d_l *to the* $k-1$ *trees with depth* d_s *will only increase the depth of the* $k-1$ *trees by at most 1.*

Proof of Lemma 6. *The proof of this lemma is similar to the proof for Lemma 5. If the number of leaves in the large tree is not greater than the number of leaves in* $k-1$ *small trees then adding the leaves to the small tree will not increase the depth of the* $k-1$ *trees by at most 1. If there are* $k-1$ *trees of size* d_s*, then there are* $(k-1)\Delta(\Delta-1)^{d^s}$ *possible children for the leaves at depth* d_s*. There are* $\Delta(\Delta-1)^{(d_l-1)}$ *leaves in the large tree. Therefore,* $(k-1)\Delta(\Delta-1)^{d^s} \geq \Delta(\Delta-1)^{d_l-1}$ *must hold. This can be simplfied to* $(k-1) \geq (\Delta-1)^{d_l-d_s-1}$*. So if the inequality holds then the number of leaves in the large tree is less than the number of leaves in the big tree.*

□

From Lemma 6, we can see that if the inequality holds we can remove nodes from the large tree and distribute them on the $k-1$ *small trees without increasing the depth of the* $k-1$ *smaller trees by the depth of the row removed from the large tree. This means that when the inequality holds the depth of the even tree is always less than or equal to the sum of the depths of the large tree and the small tree.*

Since the inequalities overlap, this means that for all cases either the diameter of the graph is either the depth of two trees where the nodes are evenly distributed or the depth of a single tree which contains all nodes plus the time to traverse across a supernode. The diameter is the minimum of these two cases. Since the evenly distributed trees have $\frac{n-k}{k}$ *nodes and the large tree has* $n-k$ *nodes then by Moore's Bound, the depth of these trees will be,*

$$d_e \geq \frac{log(\frac{(\frac{(n-k)}{k})(\Delta-2)+2)}{\Delta})}{log(\Delta-1)} \quad (17)$$

$$d_l \geq \frac{log(\frac{(n-k)(\Delta-2)+2)}{\Delta})}{log(\Delta-1)} \quad (18)$$

And the diameter of the graph is

$$d_n = min(2d_e + 2, d_l + 2) \quad (19)$$

□

References

[1] Gnutella. http://www.gnutelliums.com/.

[2] Kazaa. http://www.kazaa.com.

[3] I. Stoica B. T. Loo, R. Huebsch and J. Hellerstein. The case for a hyrid p2p search infrastructure. In *IPTPS*, 2004.

[4] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418. ACM Press, 2003.

[5] D. Plonka. An analysis of napster and other ip flow sizes. http://moat.nlanr.net/natimes/april2001.pdf, April 2001.

[6] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.

[7] Jordan Ritter. Why Gnutella can't scale. No, really., 2001.

[8] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems, 2002.

[9] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.