Maximizing Remote Work in Flooding-based Peer-to-Peer Systems

Qixiang Sun, Neil Daswani, and Hector Garcia-Molina

Computer Science Department
Stanford University, Stanford, CA 94305, USA
{qsun, daswani, hector}@cs.stanford.edu

Abstract. In peer-to-peer (P2P) systems where individual peers must cooperate to process each other's requests, a useful metric for evaluating the system is how many remote requests are serviced by each peer. In this paper we apply this remote work metric to flooding-based P2P search networks such as Gnutella. We study how to maximize the remote work in the entire network by controlling the rate of query injection at each node. In particular, we provide a simple procedure for finding the optimal rate of query injection and prove its optimality. We also show that a simple prefer-high-TTL protocol in which each peer processes only queries with the highest time-to-live (TTL) is optimal.

1 Introduction

Flooding-based peer-to-peer systems like Gnutella [4] have been deployed and used by millions of users worldwide to share and exchange files. As of April 2003, Gnutella has over one million users (with at least one hundred thousand concurrent users [5]) and ten tera-byte of shared data. Also according to [3], there are over 10 vendors actively developing Gnutella-style clients for their applications.

While there is significant research interest in distributed hash tables [8] [9] [11] [14], Gnutella-style systems are used in practice for four reasons: 1) simple to implement, 2) easy to deploy, 3) extremely robust in handling frequent peer arrivals and departures, and 4) supports wild-card searches. Moreover, in ad-hoc wireless environments where unicast is just as expensive as broadcast, a flooding-based mechanism is more desirable.

Although a flooding-based search mechanism can be inefficient as a search query is forwarded to all nodes within a certain number of hops (e.g., 7 hops), Gnutella-style networks have, nevertheless, scaled to millions of users by using a super-node architecture where high speed (CPU and bandwidth) nodes act as proxies for regular (slower) nodes. Figure 1 shows a sample super-node network with 3 super-nodes and 16 regular nodes. Each super-node indexes the content of its attached regular nodes and performs the flooding-based search on

[0] This work is supported in part by NSF Graduate Fellowship and NSF Grants EIA 0085896, IIS-9817799, and CCR-0208683.

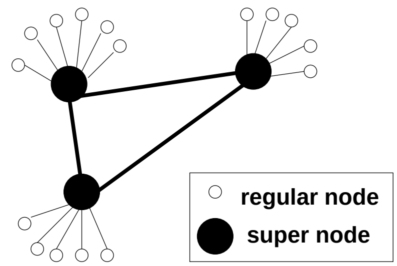


Fig. 1. A sample super-node network.

behalf of the regular nodes. In this architecture, a network with millions of users can be reduced to one with tens of thousands of super-nodes, where a flooding mechanism is adequate.

Even with this architecture, super-node networks are still susceptible to overloading when too many search queries are generated by users. In the extreme case, if every super-node uses all of its processing capacity to inject new search queries instead of answering and propagating existing queries, no "useful" work is done because queries are not answered by anyone. We define "useful" or *remote work* as a super-node processing a query that is not inject by itself or by its attached regular node. At the other extreme, if super-nodes inject too few new queries, they will have available capacity to process remote queries, but there will not be enough queries to keep the super-nodes busy. Thus, our goal is to pick a query-injection rate between these two extremes that maximizes the remote work performed.

We chose *remote work* as our objective metric because it succinctly captures the goal of users. The more remote super-nodes that process a given user query, the more potential answers the user will receive. From among the answers, the user can then select those he wants, and the larger the selection, the better. For instance, if the user searches for compositions by "Bach," he can then select titles that sound appealing, or files that have a good recording quality.

One approach to maximizing the remote work is to change the search protocol itself, e.g., using random walkers [7] or iterative deepening [13]. In this paper we attack the problem from a different angle: we control the rate of query injection at individual super-nodes. We address the following questions:

- How do we model query injection, processing, and propagation in a Gnutella-style system?
- What is the optimal number of new queries that each super-node should inject each round as to maximize the remote work done in a network?
- What is the impact of using different protocols to select which queries to process and propagate? Is there an optimal protocol?
- Should we enforce a fair policy where every super-node injects the same number of new queries into the network? Or should highly-connected super-nodes in the "critical" part of the network inject more queries (or less)?
- What is the penalty in terms of reduced remote work for using a fair policy?

– What are some heuristics for more complex systems that are outside of our simple model?

Daswani et al. in [1] conducted simulations to answer some of the above questions focusing on the impact of malicious super-nodes who purposely generate large number of bogus queries to reduce the amount of "useful" work done in a flooding-based peer-to-peer system. In the current paper, we do not consider malicious super-nodes doing denial-of-service (DoS) attacks using bogus queries. Instead, we assume all super-nodes are cooperating to maximize useful work in the network. The results in this paper provide a firm theoretical foundation for studying the effects of DoS attacks and establish a baseline of comparison. These results can be easily incorporated into [1] to further extend their results.

Knowing the theoretical optimal rate of query injection and the maximum remote work possible can improve the construction of the overlay network. For example, a super-node can use the optimal query-injection rate to dynamically decide whether it should accept more clients or disconnect existing ones. We can also use remote work as metric to evaluate different types of overlay topologies.

Although our work is specific to Gnutella-like systems, we do address an issue that we believe will be of growing importance in distributed systems, that of getting autonomous components to provide services for each other. Whether the system is a publish-subscribe one, or a sensor net, or an ad-hoc wireless network, nodes must balance their local needs (e.g., disseminate events or messages originating locally) with the services they provide to others (e.g., packet forwarding, resource discovery). As far as we know, this "distributed resource coordination" problem has not been studied in detail. Our paper is a first study of such coordination for autonomous systems.

2 Assumptions and a Model

We use a very simple model of Gnutella to capture key performance characteristics that are relevant to our goal of maximizing remote work. Given that regular nodes always access the network via a super-node, we only need to capture the activities of the super-nodes. Specifically, we model the super-node network as a graph $G = (V, E)$ where edges represent connections between super-nodes. For brevity, when we say "node" in the remainder of this paper, we mean super-node unless stated otherwise explicitly.

We model the P2P system as operating in rounds, where search queries are injected and processed during the round and forwarded to neighboring peers between rounds. Although the system does not have to be sychronous, we will assume sychrony for analysis purposes. Note that queries "injected" by a super-node are typically initiated by the regular nodes attached to it.

We assume each query has a time-to-live (TTL) field that is decremented by one each time when forwarded to other peers. When the TTL becomes negative, the query is removed from the network. For our purpose of maximizing remote work, we only model the propagation of search queries and ignore other communication such as search replies, ping-pong messages, and actual file transfers.

We also assume the bottleneck of the system is the processing capacity of the super-nodes rather than the network bandwidth. Furthermore, we assume receiving queries from the network has negligible processing cost as compared to the actual processing of a query. There are three reasons for these assumptions: (1) super-nodes have excellent network connectivities, e.g., 10 megabits or better; (2) backbone bandwidth is grossly over-provisioned; and (3) wild-card search queries are expensive to evaluate because simple hashing techniques do not work well.

We assign each super-node a processing capacity of C queries per round. A super-node may use its capacity in two ways: (1) accept and process a new search query from an attached regular node, or (2) process a remote query forwarded to it by a neighboring super-node. We refer to case 1 as a super-node *injecting new queries*, and refer to case 2 as *processing remote queries*. For clarification, processing a remote query involves two steps: one, match the query against the shared data indexed by this super-node; and two, forward this query to neighboring nodes. Obviously in a single round, the number of new queries injected plus the number of remote queries processed is at most C.

In most of our analysis in this paper, we assume all nodes have the same processing capacity to make the analysis tractable. Although Sariou et. al. [10] observed large variations among Gnutella clients, variations among super-nodes are much smaller. We will briefly outline the difficulties in handling super-nodes with different capacities as an open problem in Section 9.2.

Although each node can only process up to C queries per round, its neighboring nodes may send it more than C remote queries. Because we assumed that network bandwidth is not the limiting factor and that the cost of receiving data from the network is negligible, we allow each node to receive all the incoming remote queries even if it does not have the capacity to process them all. A node must then decide which remote queries to process this round and drop the remaining queries. We do not allow a node to "temporarily" buffer excess *remote* queries for processing at a later round because we are interested in the long-term system behavior where nodes are constantly overloaded.

The long-term behavior of a peer-to-peer system certainly depends heavily on how each node decides which queries to process and drop. For brevity, we use the term *protocol* to refer to a node's decision mechanism. As an example, a node is said to be using a random protocol if it picks which queries to process uniformly at random.

One important parameter of a protocol is how a node divides its capacity between injecting new queries and processing remote queries. We use a fraction ρ between 0 and 1 to denote this parameter. For example, $\rho = \frac{1}{3}$ implies one third of a node's capacity is allocated for injecting new queries while the other two third is used for processing remote queries. We assume that a super-node injects its full quota of ρC new queries each round, i.e., there is always an abundance of queries that regular nodes want to submit. This assumption is reasonable because our goal is to study the maximum amount of remote work possible which can only occur if nodes are generating sufficient number of new queries to keep the system busy. In practice, a super-node can inject new *local* queries at a fixed rate by buffering and delaying new search queries from its attached regular nodes.

Rather than trying to build an accurate model that can predict the actual performance of the peer-to-peer system, we have made many simplifying assumptions to make our study of the fundamental system behavior feasible. This simplified model retains all the important aspects of a flooding-based peer-to-peer protocol and does not restrict design decisions.

3 Notation and Problem Definition

- ρ_v denotes the fraction of processing capacity node v allocates for injecting new queries per round.
- $\bar{\rho} = \{\rho_v \mid v \in V\}$ denotes the set of ρ_v used by all nodes in network G.
- $\delta(u, v)$ denotes the minimum hop distance between nodes u and v in network G.
- $D(v, \tau)$ denotes the set of nodes u, excluding v, in G such that $\delta(u, v) \leq \tau$.
- $\hat{D}(v, \tau)$ denotes $D(v, \tau) \cup \{v\}$.
- $W_t^{\mathcal{P}}(v, \bar{\rho})$ denotes the set of queries processed by node v, using protocol $\mathcal{P}$ with settings $\bar{\rho}$ for the nodes, during round t. The set $W_t^{\mathcal{P}}(v, \bar{\rho})$ includes both new queries injected by v and processed remote queries. We drop the superscript $\mathcal{P}$ when the context is clear.
- $R_t^{\mathcal{P}}(v, \bar{\rho}) \subset W_t^{\mathcal{P}}(v, \bar{\rho})$ denotes the set of remote queries processed by node v at time t.
- $RW_t^{\mathcal{P}}(\bar{\rho}) = \sum_{v \in V} |R_t^{\mathcal{P}}(v, \bar{\rho})|$ denotes the number of remote queries processed by all nodes in network G at time t.

With the notation above, maximizing the remote work of a network G using protocol $\mathcal{P}$ can be stated formally as:

Problem: Given a graph $G = (V, E)$, maximum TTL τ, processing capacity C, and a protocol $\mathcal{P}$, find the optimal rate of injecting new queries $\bar{\rho} = \{\rho_v \mid v \in V\}$ such that $\sum_t RW_t^{\mathcal{P}}(\bar{\rho})$ is maximized.

The maximization problem is stated above as the cumulative number of remote queries processed over all nodes and all time. We chose to sum over all time to take into account of protocols with nondeterministic or irregular behaviors. However, as we will see, the protocols studied here all have some form of "steady-state" behavior.

4 Protocols

Before describing the protocols, we first need to discuss how to tag each query with an ID to avoid processing duplicate queries and to remove queries when their TTL expires. For a query q, we use a triplet (src, ttl, mid) where src is the

Deterministic Prefer-High-TTL Protocol $\mathcal{H}^{\mathcal{D}}$

During every round, each node $v \in V$ performs the following tasks in the order shown below:

1. Inject $\rho_v \cdot C$ new queries with the triplet identifiers $\{v, \tau, 1\}, \{v, \tau, 2\}, \ldots, \{v, \tau, \rho_v C\}$. Denote this set of local queries L_v. (For clarity in the presentation, we assume $\rho_v C$ is an integer. We can take the floor if it is not an integer.)
2. Sort all incoming queries from adjacent super-nodes in decreasing order of TTL, break ties in a deterministic manner that is independent of the current time, and remove queries that are duplicates or have already been processed at some previous time step. Denote this sorted list of new incoming queries I_v.
3. Take the first $(1 - \rho_v)C$ queries in I_v. Denote this set of remote queries R_v.
4. Service queries in L_v and R_v against local index.
5. Decrement the TTL of queries in L_v and R_v by 1.
6. Forward all queries in L_v and R_v that have $TTL \geq 0$ to all neighbors.

Fig. 2. An informal description of the deterministic prefer-high-TTL protocol.

node that injected the query, ttl is the current time-to-live of q as q moves around the network, and mid is an internal sequence number where $1 \leq mid \leq C$. We enforce three invariants about the IDs: (1) for any two queries injected by the same node in the same round, their mids are different; (2) $0 \leq ttl \leq \tau$ where τ is the maximum TTL; and (3) a query with ID (src, ttl, mid) at time t is injected at time $t - \tau + ttl$.

Note that when the query travels around the network, its ID changes as the ttl is decremented. To determine whether two query IDs q_1 and q_2 at times t_1 and t_2, respectively, refer to the same query, we check whether these two IDs have the same src node, the same mid, and were injected into the network at the same time. For example, assuming all queries initially have a TTL τ when injected, then a query with ID $q_1 = (u, 5, 2)$ at time step 8 is the same query as a query with ID $q_2 = (u, 3, 2)$ at time step 10 because both queries are injected by node u at time $8 + 5 - \tau = 10 + 3 - \tau = 13 - \tau$ with sequence number 2.

Using these IDs, we describe the operations of the deterministic prefer-high-TTL protocol $\mathcal{H}^{\mathcal{D}}$ in Figure 2. Essentially, after each node injects its new queries for the round, it then processes remote queries in decreasing TTL order until the processing capacity has been exhausted. If two queries have the same TTL, the tie is broken deterministically, e.g., lexicographically by source node ID and then the sequence number.

Similarly, the randomized prefer-high-TTL protocol $\mathcal{H}^{\mathcal{R}}$ performs the same steps as $\mathcal{H}^{\mathcal{D}}$ except ties are broken randomly. Though $\mathcal{H}^{\mathcal{R}}$ and $\mathcal{H}^{\mathcal{D}}$ are very similar, they exhibit different steady-state behavior as we will see in the next section. This distinction has significant impact on how efficiently we can simulate the protocols for experimental studies. A third protocol that we will use for illustrative purposes is the prefer-low-TTL protocol $\mathcal{L}$. Instead of sorting all the incoming queries in the set I_v in decreasing order of TTL during step 2 (of Figure 2), protocol $\mathcal{L}$ sorts the queries in increasing order of TTL.

5 Steady State

Regardless of the transient behavior at the beginning of time, a protocol that processes the most remote queries in the steady state will process the most remote work in the long run. Therefore, if two protocols have steady states, then we can simply compare their per-round performance in the steady state. It turns out that not all protocols have some form of steady state. In the technical report [12] , we trace out the execution of the prefer-low-TTL protocol $\mathcal{L}$ on a chain of five nodes where the per-round remote work oscillates with periodicity 2.

There are two flavors of steady state that are of particular interest because they distinguish between protocols $\mathcal{H}^{\mathcal{D}}$ and $\mathcal{H}^{\mathcal{R}}$. The first kind is a *strong* steady state where we can determine exactly which queries will be processed by every node. Formally,

Definition 1. *(Strong steady state) A protocol $\mathcal{P}$ has a* strong steady state *if given any $\bar{\rho}$, there exists t_0 such that for every node v and all $t > t_0$, $R_t^{\mathcal{P}}(v, \bar{\rho}) = R_{t_0}^{\mathcal{P}}(v, \bar{\rho})$.*

In other words, *strong steady state* guarantees that after time t_0, each node will process remote queries with the same triplet ID as the previous time step. For example, if node v processed a query with ID $(u, 5, 2)$ at time t_0, then v will process a query of the same ID from then on. Thus having a strong steady state makes simulation studies easier. Note that the same triple ID at two different times does not mean the same query because the two queries are created at different times.

An alternative is to relax the constraint of processing queries with the same triplet IDs.

Definition 2. *(Weak steady state) A protocol $\mathcal{P}$ has a* weak steady state *if given any $\bar{\rho}$, there exists t_0 such that for every node v and all $t > t_0$, $|R_t^{\mathcal{P}}(v, \bar{\rho})| = |R_{t_0}^{\mathcal{P}}(v, \bar{\rho})|$.*

A weak steady state only requires the number of remote queries processed to be the same rather than the query IDs to be the same. Since our objective is to maximize the total number of remote queries processed, having a weak steady state is sufficient for our analysis. Clearly, strong steady state implies weak steady state.

With these two notions of steady state, we now show protocol $\mathcal{H}^{\mathcal{D}}$ has a strong steady state. In particular, we show $\mathcal{H}^{\mathcal{D}}$ has a monotonicity property.

Proposition 1. *(Monotonicity) In protocol $\mathcal{H}^{\mathcal{D}}$, given $\bar{\rho}$, for any node v and a query ID $q = (src, ttl, mid)$,*

1. *if* $q \in W_{\tau - ttl_q}(v, \bar{\rho})$*, then* $q \in W_t(v, \bar{\rho})$ *for all* $t \geq \tau - ttl_q$*.*
2. *if* $q \in W_t(v, \bar{\rho})$ *for some* $t > \tau - ttl_q$*, then* $q \in W_{\tau - ttl_q}(v, \bar{\rho})$*.*

Monotonicity states that once a query ID q is in $W_{t_1}(v, \bar{\rho})$ for any node v and time t_1, the ID q can never disappear from $W_{t_2}(v, \bar{\rho})$ for all $t_2 > t_1$. It also guarantees the first appearance of q is at time $\tau - ttl_q$. The monotonicity is the result of breaking ties among queries of the same TTL in a deterministic fashion. Using this monotonicity, one can show that protocol $\mathcal{H}^{\mathcal{D}}$ has a strong steady state.

Theorem 1. *Protocol* $\mathcal{H}^{\mathcal{D}}$ *reaches a strong steady state in* τ *time steps.*

Our proofs of Proposition 1 and Theorem 1 are given in the technical report [12].

Unlike protocol $\mathcal{H}^{\mathcal{D}}$, the randomized version $\mathcal{H}^{\mathcal{R}}$ only has a weak steady state. Clearly $\mathcal{H}^{\mathcal{R}}$ does not have a strong steady state because the random selections do not guarantee a node will consistently choose remote queries with the same IDs. The fact that $\mathcal{H}^{\mathcal{R}}$ has a weak steady state is a directly corollary of a theorem in the next section that states both protocols $\mathcal{H}^{\mathcal{D}}$ and $\mathcal{H}^{\mathcal{R}}$ are "optimal" in the number of remote queries processed. Since $\mathcal{H}^{\mathcal{D}}$ and $\mathcal{H}^{\mathcal{R}}$ processes the same number of remote queries and $\mathcal{H}^{\mathcal{D}}$ reaches a strong steady state in τ time steps, then $\mathcal{H}^{\mathcal{R}}$ must reach a weak steady state in τ time steps.

6 Optimality of Protocol $\mathcal{H}^{\mathcal{R}}$

We now show that for any settings of $\bar{\rho}$, the two prefer-high-TTL protocols, $\mathcal{H}^{\mathcal{D}}$ and $\mathcal{H}^{\mathcal{R}}$, processes as much remote work as any other protocols using the same $\bar{\rho}$ settings, and hence are optimal. Since protocol $\mathcal{H}^{\mathcal{D}}$ is a special case of protocol $\mathcal{H}^{\mathcal{R}}$, we only show the optimality of protocol $\mathcal{H}^{\mathcal{R}}$. We prove this claim by first establishing an upper bound on the amount of remote work any protocol can process, and then showing protocol $\mathcal{H}^{\mathcal{R}}$achieves this upper bound.

For the upper bound, notice that regardless of which protocol we use, the number of remote queries a node v can process, $|R_t(v, \bar{\rho})|$, is limited by two factors: (1) node v's processing capacity, and (2) how many queries are injected by nodes within τ hops of v. At maximum capacity, a node v can process $(1-\rho_v)C$ queries per round. We call such a node *saturated*.

When a node v is not saturated, it can receive up to $K_v = C \cdot \sum_{u \in D(v,\tau)} \rho_u$ queries from nodes within τ hops. For protocols without steady state, the actual number of queries processed by node v may vary between rounds, (e.g., process no queries during one round, but a large amount the next round); however, the average number of queries processed per round, over time, is bounded by K_v.

We get our upper bound by combining the two limiting factors and taking the minimum number of remote queries processed in case 1 and case 2 (along with a special case when $t < \tau$).

Proposition 2. *For any protocol $\mathcal{P}$, any node v, and any setting $\bar{\rho}$,*

$$\sum_t |R_t(v,\bar{\rho})| \leq C \cdot \sum_t \min\left(1-\rho_v, \sum_{w \in D(v,\min(\tau,t))} \rho_w\right)$$

We now show in two steps that protocol $\mathcal{H}^{\mathcal{R}}$ achieves this upper bound. In the first step, we claim that if a node v's "neighbors" cannot inject enough queries to continuously saturate v, then node v will process every query injected by these "neighbors." Stated formally,

Lemma 1. *Consider protocol $\mathcal{H}^{\mathcal{R}}$ and any node v. Suppose for some hop count $h \leq \tau$, $\sum_{w \in D(v,h)} \rho_w \leq 1-\rho_v$. Then for all nodes $w \in D(v,h)$ and all i such that $1 \leq i \leq \rho_w C$, the query with triplet ID $(w, \tau-\delta(w,v), i) \in R_t(v,\bar{\rho})$ for all time $t \geq \delta(w,v)$.*

In the second step, we claim that if node v's "neighbors" are continuously injecting more queries than v can process, then node v processes exactly $(1-\rho_v)C$ queries each round. Formally,

Lemma 2. *In protocol $\mathcal{H}^{\mathcal{R}}$, for any node v and hop count h, if $\sum_{w \in D(v,h)} \rho_w > 1-\rho_v$, then node v is saturated after time h, i.e., $|R_t(v,\bar{\rho})| = (1-\rho_v)C$ for all $t \geq h$.*

This claim is not immediately obvious because the random selections in protocol $\mathcal{H}^{\mathcal{R}}$ may result in many duplicate queries arriving at a node v and reduce the number of remote queries processed. Fortunately, the prefer-high-TTL mechanism ensures "enough" non-duplicate queries arrive at v to saturate its processing capacity. The detailed proofs of these lemmas are given in the technical report [12].

Combining Lemmas 1 and 2 with $h = \tau$, we get that if node v's neighbors within τ hops do not inject enough queries to saturate v's processing capacity, then node v processes every query injected by them. On the other hand, if there is more than enough queries, then node v processes at maximum capacity $(1-\rho_v)C$. Consequently,

Theorem 2. *In protocol $\mathcal{H}^{\mathcal{R}}$, for any node v and any setting $\bar{\rho}$,*

$$|R_t(v,\bar{\rho})| = C \cdot \min\left(1-\rho_v, \sum_{u \in D(v,\min(\tau,t))} \rho_u\right)$$

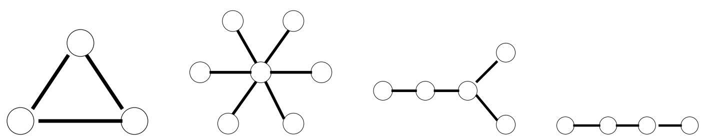
Applying Theorem 2, we immediately obtain that $\sum_t |R_t(v,\bar{\rho})|$ is equal to the upper bound established in Proposition 2. Hence,

Corollary 1. *No protocols can achieve more remote work than protocol $\mathcal{H}^{\mathcal{R}}$.*

Find_Optimal_Single_ρ:

1. order the vertex set $V = \{v_1, v_2, \ldots, v_n\}$ such that $|\hat{D}(v_i, \tau)| \leq |\hat{D}(v_{i+1}, \tau)|$.
2. construct the sequence of non-increasing real numbers $\{d_1, d_2, \ldots d_n\}$ where $d_i = \frac{1}{|\hat{D}(v_i,\tau)|}$.
3. find the smallest k such that $\sum_{i=1}^{k} |\hat{D}(v_i, \tau)| \geq n$.
4. return d_i.

Fig. 3. Procedure for finding the optimal $\hat{\rho}$ when all nodes have the same ρ.



(a) $\tau = 1$ (b) $\tau = 1$ (c) $\tau = 2$ (d) $\tau = 1$

Fig. 4. Four example topologies.

Another important consequence of Theorem 2 is that in computing remote work, we do not have to worry about which queries were duplicates or which path a query traveled on. Therefore, we can treat all queries as indistinguishable from each other and rewrite our optimization problem into a simple linear program (LP). The LP is given in the technical report [12]. However, solving the LP gives us little insight into the problem's structure. The next section builds such insights for a special case of the problem where each node has the same ρ setting, i.e., $\rho_v = \rho$ for all v.

7 Identical ρ for All Nodes

The instance of every node having the same ρ is of particular interest because it captures fairness in the super-node network. In other words, every super-node injects the same number of new queries into the network. This instance also arises when the software clients have a hard-coded and pre-determined capacity allocation. Clearly, finding the optimal $\hat{\rho}$ setting that maximizes the total remote work is dependent on the network topology. In addition to presenting a procedure for selecting the optimal $\hat{\rho}$, we also show that imposing this "fair" criterion of identical ρ for all nodes does not significantly reduce the maximum amount of remote work.

Figure 3 shows our procedure for selecting $\hat{\rho}$. To illustrate, consider examples (a), (b), and (c) in Figure 4. We first write out $|\hat{D}(v_i, \tau)|$ for all nodes v_i in a non-decreasing sequence, and then add the numbers in sequence from the beginning until the sum exceeds the number of nodes. When we stopped adding at node i, the optimal $\hat{\rho}$ is the corresponding $d_i = \frac{1}{|\hat{D}(v_i,\tau)|}$. In example (a), we get the sequence of $|\hat{D}(v_i, \tau)|$ as $\{3, 3, 3\}$. Because 3 is the number nodes in this network, we stop immediately at $i = 1$ and get the optimal $\hat{\rho} = \frac{1}{3}$, as expected. Moving to the more complicated examples, we see example (b) generates the sequence $\{2, 2, 2, 2, 2, 2, 7\}$. After adding the first four 2s, we get $8 > 7$, thus the optimal $\hat{\rho} = d_4 = \frac{1}{|\hat{D}(v_4, \tau)|} = \frac{1}{2}$. In example (c), we get the sequence $\{3, 4, 4, 5, 5\}$ which yields the optimal $\hat{\rho} = \frac{1}{4}$ when $3 + 4 > 5$.

The correctness of the *Find_Optimal_Single_*ρ procedure is the result of the following theorem.

Theorem 3. *In protocol* $\mathcal{H}^{\mathcal{R}}$ *with the same* ρ*, the optimal* $\hat{\rho} = d_k$ *where* k *is the smallest integer such that* $\sum_{i \leq k} |\hat{D}(v_i, \tau)| \geq n$.

The formal proof is given in the technical report [12]. Here we outline the general idea behind the theorem. Note that given any ρ, we can divide the nodes into two categories: the set of saturated nodes S and the set of unsaturated nodes U. Now consider using $\rho' = \rho + \epsilon$ for some $\epsilon > 0$. For all nodes $v \in S$, v's remote work is reduced by ϵ, i.e., we lose a total of $R^- = \epsilon|S|$. However, for all nodes $w \in U$, w's remote work has increased by $\epsilon|\hat{D}(w, \tau)|$, or we gain $R^+ = \epsilon \sum_{w \in U} |\hat{D}(w, \tau)|$. Thus intuitively, when $R^- = R^+$, we have found a candidate for the optimal $\hat{\rho}$. Fortunately, there is only one such candidate, which corresponds precisely to Theorem 3.

There is a special case for Theorem 3 when $\sum_{i \leq k} |\hat{D}(v_i, \tau)| = n$. In this situation, there are multiple optimal $\hat{\rho}$ for a single round in the steady state. Specifically,

Corollary 2. *If* $\sum_{i \leq k} |\hat{D}(v_i, \tau)| = n$ *for some* k*, then for all* ρ *where* $d_k \geq \rho \geq d_{k+1}$*,* $RW(\rho)$ *is optimal.*

Example (d) in Figure 4 illustrates this occurrence of multiple optimal $\hat{\rho}$. The sequence of $\{|\hat{D}(v_i, \tau)|\}_i$ in this case is $\{2, 2, 3, 3\}$. Notice that $|\hat{D}(v_1, \tau)| + |\hat{D}(v_2, \tau)| = 2 + 2 = 4$ which is the number of nodes. By Corollary 2, we can conclude for example (d), any ρ where $\frac{1}{3} \leq \rho \leq \frac{1}{2}$ yields the optimal amount of remote work in a single round of the steady state.

Now that we know how to find the optimal for this special case of identical ρ for each node, a natural question is how much remote work did we sacrifice in restricting to the special case instead of using arbitrary $\bar{\rho}$? To bound this amount of lost remote work, we use the following the theorem.

Theorem 4. *For any connected network* $G = (V, E)$ *where* $|V| = n \geq \tau + 1$*, compute the optimal* $\hat{\rho}$ *using the* Find_Optimal_Single_ρ *procedure. Then in steady state,* $RW_t(\hat{\rho}) \geq \frac{\tau}{\tau+1} nC$.

The proof follows from a lemma used in proving Theorem 3 and is given in the technical report [12]. The immediate consequence of Theorem 4 is that even with the restriction of identical ρ's, nodes in the network are processing at $\frac{\tau}{\tau+1}$ of the maximum capacity. Hence, the fraction of loss due to the restriction is at most $\frac{1}{\tau+1}$. A secondary consequence is that regardless of what kind of network G we use, we can always process remote work at $\frac{\tau}{\tau+1}$ of the capacity.

8 Different ρ for Each Node

If we have all nodes inject the same number of queries into the network, some nodes will not operate at their maximum capacities. Thus it is possible to achieve more remote work by allowing nodes to inject different amounts of work, i.e., use a different ρ for each node. To illustrate the difference in the amount of remote work, we reuse the examples in Figure 4. In (b), by setting the ρ for the center of the star to 1 and 0 for the other nodes, we can saturate every node and get a total remote work of $6C$. In contrast, the identical-ρ case only yields total remote work of $\frac{7}{2}C$. Similarly, we get $4C$ and $2C$ for examples (c) and (d) respectively by setting the ρ of the nodes with the highest degrees to 1 and 0 for the other nodes. Using identical ρ, we get $\frac{7}{2}C$ and $2C$ respectively for examples (c) and (d).

In this general case where nodes can have different ρ values, there are many possible optimal solutions. In particular, there is one subset of the optimal solutions that corresponds to the *minimum fractional dominating-set* (MFDS) of distance τ for the network topology graph $G = (V, E)$. In MFDS, each node v is assigned a weight w_v where $0 \leq w_v \leq 1$. The dominating set condition is that for every node v, the sum of the weights from nodes within τ hops of v is at least 1. The goal is to come up with a set of weights w_v that satisfies the dominating condition while minimizing the sum of the weights. The MFDS is a well understood problem. Reducing our problem to the MFDS exposes some underlying structure in finding the optimal ρ_v's and allows us to leverage many existing techniques for solving it. Fortunately, there is a simple mapping from an optimal solution of MFDS to our problem. Specifically,

Theorem 5. *For any optimal solution $\{w_v\}$ to the minimum fractional dominating set of G with distance τ, the solution $\bar{\rho}$ where $\rho_v = w_v$ maximizes the total remote work in the network G.*

To prove the above claim, we observe that when all nodes are saturated, maximizing remote work is equivalent to minimizing new-query injection (i.e., MFDS). Therefore we simply need to show that there exists an optimal $\hat{\rho}$ where all nodes are saturated. Intuitively, for any optimal $\hat{\rho}$ where some node v is not saturated, we can "boost" ρ_v until v is saturated without changing the amount of remote work. The details of this "boosting" step and the proof of Theorem 5 are given in the technical report [12] .

Although using different ρ's leads to more remote work, note that we are setting ρ to 0 for a large number of nodes, which means these nodes cannot inject any queries. In practice, a node that cannot inject any queries is not useful. Therefore a combination of using a small fixed ρ (e.g., using d_n from the previous section) to guarantee some fairness while allocating the remaining capacity through the dominating set is more practical.

Distributed ρC Estimation

For every 2τ rounds (say at time t), each node $v \in V$ does the following:

1. If $|W_t(v, \rho_v)| < C$ (i.e., not enough remote work),
2. broadcast an $inc(1 - \frac{|W_t(v,\rho_v)|}{C})$ message with TTL τ.
3. If $|W_t(v, \rho_v)| > C$ (i.e., too much remote work),
4. for every node w such that $\exists(w, ttl, mid) \in W_t(v, \rho_v)$
5. send a $dec(\frac{|W_t(v,\rho_v)|}{C} - 1)$ message to node w.

Upon receiving an $inc(p)$ or $dec(p)$ message, each node adjusts its ρC by 1 with probability p.

Fig. 5. An informal description of a distributed ρC estimation heuristic.

9 Open Problems

We now outline two open problems that are practical variations of the maximizing remote work problem we studied in this paper.

9.1 Distributed Algorithm

In Sections 7 and 8, we described centralized solutions for finding the optimal ρ for each node that maximizes the total remote work in the network. Our solutions require knowing the entire network topology in advance. However in a P2P environment, with nodes constantly joining and leaving, it is impractical for any node to gather the entire network topology information. Even if we could efficiently gather such information, the rapidly changing topology will quickly render a solution based on the current topology obsolete and sub-optimal. Nevertheless, the results about the centralized solutions are important because they form the basis of comparison for distributed solutions.

For the instance of using a different ρ for each node, distributed solutions are possible by adapting fractional dominating set algorithms [2], [6]. However, these algorithms have long running times for our problem, cannot handle different capacities at each node, and must be re-run each time as the network topology changes. Here, we propose a simple heuristic for estimating how many new queries each node should inject (i.e., the value of $\rho_v C$ for each node v) in a distributed fashion. Figure 5 outlines the steps in our distributed approach. Every node only makes local decisions. When a node does not have enough queries to saturate its processing capacity, it tells all of its neighbors to inject one more local query per round. If a node has too much remote work, it tells all the nodes that have sent remote work to it to inject one less local query per round. We have performed some initial simulations to compare our heuristic against the optimal solution. The heuristic performs very well when the capacity C, in number of queries, is large compared to the number of nodes within τ hops.

The randomization for $inc(p)$ and $dec(p)$ is necessary to avoid oscillation and to stabilize the system. However, the resulting stable setting may not be optimal.

Hence, a better solution is needed. However, note that the proposed heuristic is estimating the number of new queries ρC rather than the fraction of capacity ρ as in the fraction dominating set approach. Thus this heuristic does not assume all the nodes have the same capacity C.

9.2 Nodes with Different Capacities

In reality, super-nodes may have different processing capacities. The results from the previous sections no longer hold because we cannot determine, independent of the network topology, when a node is saturated. Recall that if nodes have the same capacity, then Lemma 2 guarantees that a node v is saturated when v's neighbors are injecting more queries than v's capacity. However, when nodes have different capacities, there is a simple counterexample.

Consider nodes u, x, and v connected in a line in that order. Now assign capacity $2C$ to nodes u and v and capacity C to x. Since all the work from u must travel through x to reach v, the amount of remote work at v is limited by the capacity at x. Even if node u is injecting $2C$ queries, at most C of them will reach v each round, which invalidates Lemma 2 for the case of different capacities. In this particular example, the extra capacities at nodes u and v are irrelevant.

Even for the simple case where only one node x has more capacity than the rest, the solution is non-obvious and topology dependent. For example, if x is in an area of the network where nodes are under-saturated, then it should use its extra capacity to inject more queries. On the other hand, if x is in an area where nodes are already saturated, then the extra capacity should only be used to increase the amount of remote work at node x.

Our current approach is an incremental heuristic that combines multiple optimal solutions. The basic idea is as follows: Suppose nodes have one of two possible capacities C_1 and C_2 where $C_1 < C_2$. Then our heuristic is to find the optimal $\bar{\rho}$ setting for the entire network assuming all the nodes have capacity C_1. We then create a subgraph of the original network that includes only nodes with capacity C_2. Note that the subgraph may be disconnected. We then compute another optimal $\bar{\rho}'$ setting on the subgraph assuming all the nodes have the capacity $C_2 - C_1$. For nodes with capacity C_1, their corresponding ρ' value is 0. To get the final solution, we let each node inject $\rho_v C_1 + \rho'_v(C_2 - C_1)$ queries.

10 Concluding Remarks

This paper uses a simple model to study remote work in a flooding-based peer-to-peer network. In particular, we showed

1. For any setting $\bar{\rho}$, protocol $\mathcal{H}^{\mathcal{R}}$ processes the most remote work.
2. Under protocol $\mathcal{H}^{\mathcal{R}}$ with all nodes using the same ρ, if we order the nodes $\{v_1, \ldots, v_k\}$ where $|\hat{D}(v_i, \tau)| \leq |\hat{D}(v_{i+1}, \tau)|$, then the optimal $\hat{\rho} = \frac{1}{|\hat{D}(v_k, \tau)|}$ where k is the smallest integer such that $\sum_{i=1}^{k} |\hat{D}(v_i, \tau)| \geq n$.

3. When nodes use different ρ, any optimal solution to the minimum fractional dominating-set of the network graph G is an optimal $\hat{\rho}$ solution.

We believe that our results can serve as a benchmark for more complex systems. For example, the proposed heuristic load management scheme of Section 9.1 can be compared against a system where $\bar{\rho}$ is selected using our optimal and centralized solutions. In addition, our solutions can form the basis for heuristics, as illustrated in Section 9.2.

Acknowledgment We thank Kamesh Munagala for valuable discussions on approximation algorithms for fractional bin-packing.

References

1. N. Daswani and H. Garcia-Molina. Query-flood DoS attacks in Gnutella. In *ACM Conference on Computer and Communications Security*, Washington, DC, November 2002.
2. N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *39th Annual Symposium on Foundations of Computer Science*, pages 300–309, Palo Alto, California, November 1998.
3. The Gnutella Developer Forum (GDF). Database of vendor codes. http://groups.yahoo.com/group/the_gdf/.
4. Gnutella. Website http://gnutella.wego.com.
5. Concurrent Gnutella Hosts. http://www.limewire.com/.
6. Fabian Kuhn and Roger Wattenhofer. Constant-time distributed dominating set approximation. In 22nd *ACM Symposium on Principles of Distributed Computing*, Boston, MA, July 2003.
7. Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th annual ACM International Conference on Supercomputing (ICS)*, 2002.
8. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, pages 149–160, San Diego, August 2001.
9. A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of SOSP '01*, 2001.
10. S. Sariou, P. K. Gummadi, and S. D. Gribble. Measuring and analyzing the characteristics of Napster and Gnutella hosts. In *Multimedia Computing and Networking (MMCN)*, San Jose, CA, January 2002.
11. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, pages 160–177, San Diego, August 2001.
12. Q. Sun, N. Daswani, and H. Garcia-Molina. Maximizing remote work in flooding-based peer-to-peer systems. Technical report, Stanford University, 2003. http://dbpubs.stanford.edu:8090/pub/2003-05.
13. B. Yang and H. Garcia-Molina. Efficient search in peer-to-peer networks. In *Proceedings of the 22nd IEEE International Conference on Distributred Computing Systems (ICDCS)*, Vienna, Austria, July 2002.
14. B. Y. Zhao, J. Kubiatowicz, and A. Joseph. An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California at Berkeley, 2001.