Learning to Route Queries in Unstructured P2P Networks: Achieving Throughput Optimality Subject to Query Resolution Constraints

Virag Shah and Gustavo de Veciana
ECE Dept, UT Austin

George Kesidis
EE & CSE Depts, Penn State

***Abstract*—Finding a document or resource in an unstructured peer-to-peer network can be an exceedingly difficult problem. In this paper we propose a query routing approach that accounts for arbitrary overlay topologies, nodes with heterogeneous processing capacity, e.g., reflecting their degree of altruism, and heterogenous class-based likelihoods of query resolution at nodes which may reflect query loads and the manner in which files/resources are distributed across the network. The approach is shown to be throughput optimal subject to a grade of service constraint, i.e., stabilizes the query load subject to a guarantee that queries' routes meet pre-specified class-based bounds on their associated a priori probability of query resolution. An explicit characterization of the capacity region for such systems is given and numerically compared to that associated with random walk based searches. Simulation results further show the performance benefits, in terms of mean delay, of the proposed approach. Additional aspects associated with reducing complexity, learning, and adaptation to class-based query resolution probabilities and traffic loads are studied.**

I. Introduction

Peer-to-peer (P2P) systems continue to find increasing and diverse uses as a distributed, scalable and robust framework to deliver services, e.g., file sharing, video streaming, expert/advice sharing, sensor networks, databases, etc. One of the basic functions of such systems is that of efficiently resolving queries or discovering files/resources. This is the problem addressed in this paper.

There is a considerable body of work exploring the design of efficient search/routing mechanisms in structured and unstructured P2P networks, see e.g., [1]–[10]. In structured networks, peers/files/resources are organized to form overlays with specific topologies and properties. Search mechanisms that perform name resolution based on distributed hash table (DHT) coordinate systems can be devised to achieve good forwarding-delay properties, see e.g., [2]. In such systems, the query traffic may depend on how keys are assigned. So, load balancing requires proactive/reactive assignments of keys to peers and data/service objects, e.g., [11], and possibly exploiting network hierarchies [10]. Fundamentally, in such networks the difficulty of search/discovery is shifted to that of maintaining the structural invariants required to achieve efficient query resolution particularly in dynamic settings with peer/content churn or when reactive load balancing is required.

This work was supported by the National Science Foundation under Award CNS-0915928.

Unstructured networks, by contrast, are easier to setup and maintain, but their mostly ad hoc overlay topologies make realizing efficient searches challenging. In a purely unstructured P2P network, a node only knows its overlay neighbors. With such limited information, search techniques for unstructured networks have mostly been based on limited-scope flooding, simulated random walks, and their variants [3]–[5]. Much research in this area has focused on evaluating these search techniques based on the contact time, i.e., number of hops required to find the target, using the spectral theory of Markov chains on (random) graphs, see e.g., [4]–[6]. Unfortunately in heterogenous settings where service capacity or resolution likelihoods vary across peers, such search techniques perform poorly under high query loads.

The inefficiencies of purely unstructured networks can be partially addressed by hybrid P2P systems, e.g., FastTrack and Gnutella2. Such systems use a simple two-level hierarchy where some peers serve as 'super-peers.' These are high degree nodes which are well connected to other super-peers and to a set of subordinate nodes in a hub-and-spoke manner [12]. Though such systems have advantages in terms of scalability, proposed search techniques are still based on variants of flooding and random walks.

The work of [7] proposes an approach where peers cache the outcomes of past queries as informed by reverse-path forwarding. The idea is to learn, from past experience, the best way to forward certain classes of queries, i.e., to intelligently "bias" their forwarding decisions by correlating classes of queries with neighbors who can best resolve them. This approach involves considerable overhead, is not load sensitive, and has not yet given guarantees on performance.

Although, as will be clear in the sequel, our results are not exclusive to hybrid P2P networks, these will serve as the focus of the paper. We assume each super-peer contributes a possibly heterogenous amount of processing resource for resolving queries for the network - incentives for doing so are outside of the scope of this paper, see e.g., [8], [9]. Super-peers serve their subordinates by resolving queries, or forwarding them to other super-peers. Super-peers can resolve queries by checking the files/resources they have, as well as those of their subordinate community. In our approach we also introduce a notion of query classes. These might, for example, represent types of content, such as music, films, animations, documents, or some other classification of files/resources relevant to the

application at hand. The idea is that such a grouping of queries into classes can be used as a low overhead approach to make useful inferences on how to relay queries.

Given a hybrid P2P topology and query classification we propose a novel query resolution mechanism akin to using backpressure-based scheduling/routing of traffic in data networks [13], [14]. Our approach contrasts with previous work on search for P2P networks in that it attempts to learn, using backpressure on query queues, how to schedule and forward queries in a manner that is sensitive to both the query loads and the resolution probabilities, towards achieving provable performance characteristics. Our approach differs from standard work on backpressure in that: (1) the destinations of commodities, i.e., where queries are to be resolved, are not predetermined; (2) we introduce probabilities for resolution for query classes at super-peers which can be estimated to guide the algorithm; and (3) we introduce a history dependent treatment of queries, i.e., the routes of queries evolve based on which nodes have they already visited.

Our Contributions. The main contributions of this paper are as follows. We propose a query forwarding mechanism for unstructured (hybrid) P2P networks with the following properties.

1. It dynamically accounts for heterogeneity in super-peer's 'service rate,' reflecting their altruism, and query loads across the network. To the best of our knowledge, this is the first work to rigorously account for such heterogeneity in devising a search mechanism for P2P networks.

2. It is based on classifying queries into classes. This classification serves as a type of name aggregation, which enables nodes to infer the likelihoods of resolving class queries, which, in turn, are used in learning how to forward queries.

3. Our approach is fully distributed in that it involves information sharing only amongst neighbors, and achieves *throughput optimality* subject to a Grade of Service (GoS) constraint on query resolution. Throughput optimality here refers to stabilizing the query queues if at all possible. The GoS constraint corresponds to guaranteeing that each query class follows a route over which it has a reasonable 'chance' of being resolved.

4. We provide and evaluate several interesting variations on our throughput optimal mechanism that help significantly improve the delay performance, and further reduce the complexity making it amenable to implementation. Specifically, we formally show that backpressure with aggregated queues, where aggregation is based on queries' histories, is throughput optimal for fully connected super-peer networks. This provides a basis for substantially reducing complexity by approximations, e.g., in the case where content is randomly placed.

Organization. In Section II, we set up our basic system model. We characterize the stability region of the network and provide the throughput optimal protocol and several modifications in Section III. We provide some numerical results in Section IV. We discuss estimation of query resolution probability and ways to reduce implementation complexity in Section V.

II. System Model

The overlay network is represented by a directed graph $G = (\mathcal{N}, \mathcal{L})$ where $\mathcal{N}$ are the super-peers and $\mathcal{L} \subset \mathcal{N} \times \mathcal{N}$ are overlay links, which are assumed to be symmetric, i.e., if $(i,j) \in \mathcal{L}$ then $(j,i) \in \mathcal{L}$. We let $N(i)$ denote the neighbors of super-peer i. Note that subordinate peers of the hybrid network are not explicitly represented, but are simply associated with the super-peer to which they are connected. We assume that time is slotted, and each super-peer i has an associated service rate μ_i, corresponding to a positive integer number of queries it is willing to resolve/forward in each slot.

Let $\mathcal{R}$ be the set of all files/resources that might be queried on the network. Let $\mathcal{C}$ be a predefined set of query classes. For each $c \in \mathcal{C}$, let $\mathcal{R}_c \subset \mathcal{R}$ be the files/resources of that class. For each $c \in \mathcal{C}$ and $i \in \mathcal{N}$, let R_i^c be the set of files/resources in class c which are available at super-peer i or its subordinate peers. Let $A_i(t)$ be a random variable denoting the number of queries arriving at super-peer i or its subordinates at time t and ν_r denote the *probability* a query is for file/resource $r \in \mathcal{R}$. The classification of queries induces random variables $A_i^c(t)$ denoting the number of class c queries that arrive at super-peer i or its subordinates at time t. We assume these random variables are rate ergodic, have finite second moments, are independent across slots, and thus have well defined arrival rates denoted by $\lambda \triangleq (\lambda_i^c : i \in \mathcal{N}, c \in \mathcal{C})$ where λ_i^c denotes the mean arrival rate of class c queries at node i

If a class c query at node i cannot be resolved it may be forwarded to one of its neighbors. The likelihood that a node can resolve such a query depends, not only on its class, but also on its *history*, i.e., the set of nodes it visited in the past. For example, suppose 3 nodes in a network partition files/resources $\mathcal{R}_c$ associated with class c. If two of these nodes attempted and failed to resolve a given class c query then it will for sure be resolved at the third node. In other contexts, if a search for a particular media file failed at many nodes, it is more likely that the file is rare, and the conditional likelihood that it is resolved at the next node might lower.

We capture such behavior for different classes by keeping track of the history of a query, i.e., the subset of nodes already visited, Thus a history is an element of powerset of $\mathcal{N}$, which we represent by $\mathcal{H}$. The 'type' τ of a query keeps track of its class c and its history H, i.e., $\tau = (c, H) \in \mathcal{T} \triangleq \mathcal{C} \times \mathcal{H}$; we define functions such that $c(\tau) = c$ and $H(\tau) = H$. Further, we let $e_i(\tau)$ represent the resulting type once a query of type τ is serviced by node i. Note a query that *revisits* a node does not change type, thus, $e_i(\tau) = (c(\tau), H(\tau) \cup \{i\})$ only if $i \notin H(\tau)$, otherwise $e_i(\tau) = \tau$. In turn, we let $E_i^{-1}(\tau)$ denote the inverse set of $e_i(\tau)$, i.e., $E_i^{-1}(\tau) = \{(c(\tau), H) : H \cup \{i\} = H(\tau)\}$. Thus, if $i \in H(\tau)$, $E_i^{-1}(\tau)$ contains τ and $(c(\tau), H(\tau) \setminus \{i\})$, otherwise $E_i^{-1}(\tau) = \emptyset$.

Query Resolution Probability. We model the probabilities of resolving queries across the network by a vector $p \triangleq (p_i^\tau : i \in \mathcal{N}, \tau \in \mathcal{T})$, where p_i^τ denotes the probability that a *typical* query of class $c(\tau)$ is resolved by i conditioned on failing attempts at nodes in $H(\tau)$. A node i can easily estimate p_i^τ

by keeping track of the fraction of queries of type τ that it was able to resolve.

Grade of Service on Query Resolution. A standard mechanism adopted in P2P systems is to remove a query from the network if it is unresolved after having traversed some fixed number of nodes, i.e., TTL threshold. Unfortunately, while this limits resource usage, it does not translate to a guaranteed grade of service on query resolution. We propose a different approach. Let $\phi(\tau)$ be the a priori probability that a typical query of class $c(\tau)$ is resolved upon visiting nodes in $H(\tau)$. As explained below we propose enforcing the removal of a query of type τ from the network if $\phi(\tau) \geq \gamma_{c(\tau)}$ where γ_c is the design parameter determining the GoS for class c. This guarantees that a *typical* class c query would have seen a chance of at least γ_c of being resolved. Note $\phi(\tau)$ does not depend on the order of the nodes traversed by the query, but can be computed recursively as a query traverses a sequences of nodes, e.g., if $H(\tau) = \{i_1, i_2, \ldots, i_k\}$, then $\phi(\tau) = 1 - \Pi_{l=1}^{k}(1 - p_{i_l}^{\tau_l})$, where $\tau_l = (c(\tau), \{i_1, i_2, \ldots, i_{l-1}\})$). For our purposes we model such an exit strategy directly in p itself. Specifically, if at node i we have $\tau' = (c(\tau), H(\tau) \cup \{i\})$ such that $\phi(\tau') \geq \gamma_{c(\tau)}$, then we set $p_i^{\tau} = 1$. Under this model a query of type τ exits the network after service at node i irrespective of the nodes' success or failure in resolving it since the GoS requirement has been satisfied.

To summarize, the vector p does not only reflect class-based probabilities of query resolution for various types of queries at the nodes but also the GoS requirement, or exit criterion, implemented by underlying query resolution protocol.

Network State and Routing Policies. We assume that arrivals occur at the end of each slot. We let $Q_i^{\tau}(t)$ denote number of queries of type τ waiting for service at node i at the start of slot t. $Q(t) \triangleq (Q_i^{\tau}(t) : i \in \mathcal{N}, \tau \in \mathcal{T})$ represents the network's state at the start of slot t. Queries are served sequentially in each slot according to some 'policy'. Policies are subject to the constraint that no more than μ_i[1] queries be serviced at node i in each slot. We say a policy is *ergodic* if sample paths of $Q(t)$ are ergodic and a steady state distribution exists. Note that 'service' here includes both the attempt to resolve the query as well as determining a routing (forwarding) strategy for the queries. Queries are forwarded at the end of the slot.

State dependent randomized policy: Given that $Q(t) = q(t)$, a randomized policy does the following for each node i:

1) It randomly chooses the types of the μ_i queries to be served. Queries of those types are resolved on a first come first serve basis, where if none is available a blank query is included.
2) For each unresolved (non blank) query, it randomly chooses a neighbor $j \in N(i)$ to which it should be forwarded.

Such a policy depends on specifying a vector $\pi(t) = (\pi_{ij}^{\tau}(t) : (i,j) \in \mathcal{L}, \tau \in \mathcal{C})$ for each slot t, where $\pi_{ij}^{\tau}(t)$ is the probability that a given query served by node i at time t belongs to type τ, and is forwarded to node j, if unresolved. In general, $\pi_{ij}^{\tau}(t)$ can depend on $q(t)$ and/or t explicitly. Also, $1 - \sum_{j,\tau} \pi_{ij}^{\tau}(t)$ is the probability that no type is chosen, in which case a blank query is served. These probability vectors determine the service rate allocations at each node. Indeed, let $\mu(t) \triangleq (\mu_{ij}^{\tau}(t) : (i,j) \in \mathcal{L}, \tau \in \mathcal{T})$, where, $\mu_{ij}^{\tau}(t) \triangleq \mu_i \pi_{ij}^{\tau}(t)$. Thus, determining $\pi_{ij}^{\tau}(t)$ is equivalent to determining $\mu_{ij}^{\tau}(t)$. Further, define $\mu_i^{\tau}(t) \triangleq \sum_j \mu_{ij}^{\tau}(t)$ for all $i \in \mathcal{N}$ and $\tau \in \mathcal{T}$. Note once again that, in general, the service rates $\mu_i^{\tau}(t)$ may be function of $q(t)$ and/or t.

For simplicity, we shall refer to such policies as randomized policies. Note that these include policies where the state deterministically determines the query-type to be serviced and the forwarding strategy at each node. Indeed, this corresponds to the case where $\pi_{ij}^{\tau}(t) = 1$ for some $\tau \in \mathcal{T}$ and $j \in N(i)$, and 0 for others. A randomized policy is called *fixed* if $\pi(t)$ does not depend on t or $q(t)$.

III. Throughput Optimal Query Resolution

In this section, we will propose a query scheduling and forwarding policy that ensures GoS for each class, is distributed, easy to implement, and is throughput optimal. We begin by defining the stability for such networks and the associated capacity region.

A. Stability & Capacity Region

We shall use the definition of network stability given in [13], which is general in that it includes non-ergodic policies. However for ergodic policies it is equivalent to standard of notions of stability given in [14], [15]. For a given queue process $Q_i^{\tau}(t)$, let $g_i^{\tau}(\alpha)$ denote its 'overflow' function

$$g_i^{\tau}(\alpha) = \limsup_{t \to \infty} \frac{1}{t} \sum_{t'=1}^{t} \mathbf{1}\{Q_i^{\tau}(t') > \alpha\} \tag{1}$$

associated with the fraction of time $Q_i^{\tau}(t)$ exceeds α.

Definition 1: A *queue $Q_i^{\tau}(t)$ is stable* if $g_i^{\tau}(\alpha) \to 0$ almost surely as $\alpha \to \infty$. The network is stable if each queue is stable.

Next we define the 'capacity region' for query loads on our network.

Definition 2: The *capacity region* Λ is set of query arrival rates λ, such that there exists a feasible solution to the following linear constraints on $f \triangleq \left(f_{ij}^{\tau} : (i,j) \in \mathcal{L}, \tau \in \mathcal{T}\right)$:
Capacity constraints: for all $i \in \mathcal{N}$,

$$\sum_{j,\tau} f_{ji}^{\tau} + \sum_{c} \lambda_i^c \leq \mu_i; \tag{2}$$

Flow conservation constraints with resolution at nodes: for all $i \in \mathcal{N}$ and $\tau \in \mathcal{T}$,

$$\sum_{j} f_{ij}^{\tau} = \sum_{\tau' \in E_i^{-1}(\tau)} (1 - p_i^{\tau'})\Big(\sum_{j} f_{ji}^{\tau'} + \lambda_i^{c(\tau')} \mathbf{1}\{H(\tau') = \emptyset\}\Big); \tag{3}$$

Non-negativity constraints: for all $(i,j) \in \mathcal{L}$ and $\tau \in \mathcal{T}$,

$$f_{ij}^{\tau} \geq 0. \tag{4}$$

[1]Although we have assumed μ_i is integer-valued, fractional service rates can be modeled by allowing random service rates per slot. The results herein hold with some additional technicalities in the proofs.

We refer to f as flow variables, where (2) ensures that the incoming flow to a node is less than its service rate and (3) ensures that the total flow of types $\tau' \in E_i^{-1}(\tau)$ reaching i which is not resolved at i (left hand side) equals the flow of type τ leaving node i.

We let Λ' denote the interior of Λ. The following theorem, proven in the Appendix A, makes the link between the capacity region and stabilizability of the network.

Theorem 1: (Capacity Region and Stabilizability)

(a) If for a given arrival rate vector λ there exists a policy under which the network is stable, then $\lambda \in \Lambda$.
(b) If $\lambda \in \Lambda'$, then there exists a fixed randomized policy under which the network is stable.

Note that this result is general in that even full knowledge of future events does not expand the region of stabilizable rates. Also, while our focus, for now, is on policies where p corresponds to the conditional probabilities of query class resolutions subject to the GoS modification, other modifications could be made. The only restrictions on p for the above result to hold is that each query should eventually leave the network, and that revisits to nodes (while allowable) have a zero probability of resolving the query.

B. Throughput optimal query resolution policies

In principle, given $\lambda \in \Lambda'$, a feasible set of network flows can be found and, as shown in the proof of Theorem 1.b, this can be used to devise a fixed randomized policy which stabilizes the network. However, such a centralized policy may not be practically feasible, moreover arrival rates λ may not be known a priori. Below we develop a distributed dynamic algorithm where each node i makes decisions based on its queue states and that of its neighbors and only needs to know (or estimate) p_i^τ for all $\tau \in \mathcal{T}$, i.e., local information.

Basic Backpressure Algorithm: For each t, given $Q(t) = q(t)$ each node, say i, carries out the following steps:
1) For each neighbor $j \in N(i)$ it determines

$$\begin{aligned} w_{ij}^*(t) &= \max_{\tau \in \mathcal{T}} \left\{ q_i^\tau(t) - q_j^{e_i(\tau)}(t)(1-p_i^\tau) \right\} \\ \tau_{ij}^*(t) &= \arg\max_{\tau \in \mathcal{T}} \left\{ q_i^\tau(t) - q_j^{e_i(\tau)}(t)(1-p_i^\tau) \right\} \end{aligned}$$

2) It finds $j_i^* = \arg\max_{j \in N(i)} w_{ij}^*(t)$, and lets $\tau_i^* = \tau_{ij_i^*}^*$.
3) It serves $\min[q_i^{\tau_i^*}, \mu_i]$ queries of type τ_i^*, and forwards the unresolved ones to node j_i^*. This is equivalent to a state dependent randomized algorithm with $\mu_{ij}^{*\tau}(t)$ equal to μ_i when $j = j_i^*$ and $\tau = \tau_i^*$, and 0 otherwise, in slot t.

Note that the weights used in above algorithm for each link (i, j) are different from those used in traditional multi-commodity backpressure algorithm [13], [14], where weights are found using differences between queue backlogs of each commodity at i and j. Here, for each type τ, one takes difference of the queue backlog at i from that of 'expected' queue backlog a query of type τ would see at j if forwarded by node i to j. To see this, observe that query of type τ would get resolved with probability p_i^τ and would thus leave the network. But, with probability $(1 - p_i^\tau)$ it would not be resolved, and would see queue backlog of $q_j^{e_i(\tau)}$ at node j. Thus, the weight taken is $w_{ij}^*(t) = \max_{\tau \in \mathcal{T}} \left\{ q_i^\tau(t) - q_j^{e_i(\tau)}(t)(1-p_i^\tau) \right\}$.

Theorem 2: The above backpressure algorithm is throughput optimal, i.e., it achieves stability for any $\lambda \in \Lambda'$.

A proof of Theorem 2 is given in Appendix B. This basic backpressure algorithm, though throughput optimal, is wasteful. In a slot, each node i serves only the queue with highest relative backlog. In case that particular queue has less than μ_i queries waiting in it, the spare services are provided to blank queries, even if other queues are non-empty. We now devise a more efficient protocol that serves blank queries only when all the queues are non-empty and is thus work-conserving; and is still throughput optimal. As we shall see, this provides large delay benefits over the above basic backpressure algorithm.

The idea is, if the number of queries in the queue with highest relative backlog is less than the total service rate, the work conserving policy serves the queries in the queue with second highest backlog, and so on, until either total of μ_i queries are served or all the queues are empty. We formally define the algorithm as follows.

Work Conserving Back-pressure Policy: Given $Q(t) = q(t)$, each node i does the following.
1) It finds the least positive integer k such that $\sum_{l=1}^{k} \max_{\tau,j}^{(l)} \left\{ q_i^\tau(t) - q_j^{e_i(\tau)}(t)(1-p_i^\tau) \right\} \geq \min\left[\mu_i, \sum_\tau q_i^\tau(t)\right]$, where $\max_\tau^{(l)}$ refers to the l^{th} largest value.
2) For $l = 1, 2, \ldots, k$, for each $j \in N(i)$, it finds

$$\begin{aligned} w_{ij}^{*l}(t) &= \max_{\tau}^{(l)} \left(q_i^\tau(t) - q_j^{e_i(\tau)}(t)(1-p_i^\tau) \right) \\ \tau_{ij}^{*l}(t) &= \arg\max_{\tau}^{(l)} \left(q_i^\tau - q_j^{e_i(\tau)}(t)(1-p_i^\tau) \right). \end{aligned}$$

3) For $l = 1, 2, \ldots, k$, it finds $j_i^{*l} = \arg\max_j w_{ij}^{*l}(t)$ and lets $\tau_i^{*l} = \tau_{ij}^{*l}(t)$ for $j = j_i^{*l}$.
4) For $l = 1, \ldots, k-1$, it serves all the queries of type τ_i^{*l} and forwards the unresolved queries to node j_i^{*l}. For queries of type τ_i^{*k}, it serves $\min\left(q_i^{\tau_i^{*k}}(t), \mu_i - \sum_{l=1}^{k-1} q_i^{\tau_i^{*l}}(t) \right)$ of them on an FCFS basis and forwards unresolved ones to j_i^{*k}.

Corollary 1: The above work conserving backpressure policy is throughput optimal.
The proof is provided in Appendix C.

IV. Numerical Results and Simulations

In this section, we numerically evaluate the gains in capacity achievable by our throughput optimal backpressure algorithms versus a baseline random walk policy. We consider a fully connected network with 6 super-peers, $\mathcal{N} = \{1, 2, \ldots, 6\}$, supporting two query-classes c_1 and c_2 with homogenous arrival rates across each nodes given by λ_1 and λ_2 respectively. This reduces the dimension of the capacity region from 12 to

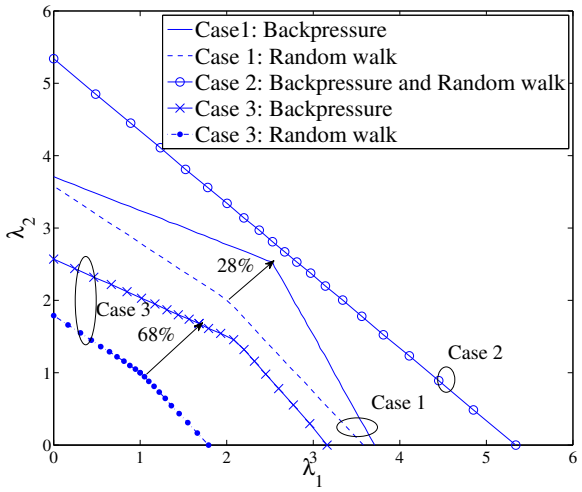

Fig. 1. Boundaries of capacity regions for the throughput optimal backpressure algorithm and random walk policy for the 3 cases.

2, making it easier to study. The GoS parameters for the two classes are set to 0.9, viz. $\gamma_1 = \gamma_2 = 0.9$. Note since super-peer networks are typically highly connected in practice, a fully connected network model might be representative.

In the baseline random walk policy, upon service, each node forwards an unresolved query to a randomly selected neighbor which was not been previously visited. To make a fair comparison, we also enforce an exit policy, i.e., a query exits the network once the GoS constraint has been satisfied, see Section II. As for backpressure, we can characterize the capacity region for this random walk policy, see [16].

We considered the following three cases, see Fig. 1.

Case 1: $\mu_i = 10$ for all $i \in \mathcal{N}$. For all types $\tau \in \mathcal{T}$ such that $c(\tau) = c_1$, $p_i^\tau = 0.6$ if $i \in \{1, 2, 3\}$ and $p_i^\tau = 0.1$ otherwise. Similarly, when $c(\tau) = c_2$, $p_i^\tau = 0.1$ if $i \in \{1, 2, 3\}$ and $p_i^\tau = 0.6$ otherwise.

Case 2: $\mu_i = 10$ for all $i \in \mathcal{N}$. $p_i^\tau = 0.5$ for all $i \in \mathcal{N}$ and $\tau \in \mathcal{T}$.

Case 3: $\mu_i = 15$ for $i \in \{1, 3, 5\}$ and $\mu_i = 5$ otherwise. p is the same as in Case 1.

Fig. 1 exhibits significant capacity gains for Cases 1 and 3. It also shows that, when μ_i and p_i^τ and traffic is homogenous over nodes as in Case 2, the random walk policy is able to balance loads achieving the capacity. However, with heterogeneity in nodes' query resolution probability, i.e., how they store the files/resources of various classes, backpressure significantly outperforms the random walk. For example, in Case 1, when λ_1 and λ_2 are constrained to be equal, a 28% gain in capacity is achieved. Further, for Case 3, the gain along the direction $\lambda_1 = \lambda_2$ of the capacity region increases to 68%. This shows that the advantages of load balancing by backpressure are significant, particularly when there is heterogeneity among nodes in their service rates, i.e., their altruism, as well. One might further expect that with heterogeneity in traffic across nodes the gains would also be good.

We now compare the delay performance of our backpressure algorithms to the random walk based policy in Case 1. Fig. 2

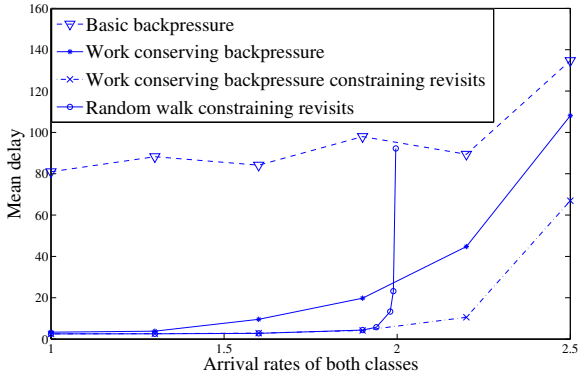

Fig. 2. Mean delay of resolved queries for various algorithms for Case 1.

exhibits the expected delay of for each query arrival as a function of the arrival rates for both the classes, keeping the arrival rates equal. It confirms our observation that the basic backpressure algorithm is throughput optimal, but wasteful as it is not work conserving. The work conserving algorithm significantly improves performance. Performance is further improved by constraining queries from revisiting nodes. With this modification, the backpressure algorithm has excellent delay performance as compared to the random walk policy with the same revisit constraints and same GoS, especially at higher loads.

V. Implementation and Complexity

A. Estimating query resolution probabilities

So far we have assumed that resolution probabilities for queries of different types are known. In practice they can be easily estimated. Due to space limitations we will only discuss this briefly. In order to ensure unbiased estimates be obtained at each node, suppose a small fraction ϵ of all queries is marked 'RW', forwarded via the random walk policy with a large TTL, and given scheduling priority over other queries. With a sufficiently large TTL this ensures that each node will see a random sample of all query types it could see. Suppose all other queries are treated according to our backpressure policy based on estimated query resolution probabilities. After a sufficiently long time the estimates should converge to a sufficient degree of accuracy, at which point marking can stop and the backpressure algorithm can ensure throughput optimality. Alternatively, one can *always* have a fraction ϵ of the queries marked and routed accordingly, to allow persistent tracking of changes in resolution probabilities or the popularities of files/resources. Under such circumstances it is easy to see that some of the system capacity will be used up by RW queries and the remaining query load would be routed in a throughput optimal manner with respect to a reduced capacity region depending on resources consumed by RW queries, which in turn are controlled by appropriately choosing ϵ and the TTL.

B. Reducing complexity

Not unlike standard backpressure-based routing, our policies suffer from a major drawback: each node needs to share the

state of its potentially large number of non-empty queues with its neighbors. For backpressure-based routing the number of queues per node corresponds to the number of flows (commodities) in the network. In our context, the number of queues per node corresponds to number of query types it could see, i.e., worst case $\Theta(|\mathcal{C}|2^{|\mathcal{N}|})$. In this section we propose simple modifications and approximations that considerably reduce the overheads, albeit with some penalty in the performance. The key idea is to define equivalence classes (called levels in the sequel) of query types that share a 'similar' history, in the sense that they have similar conditional probabilities of resolution, and have them share a queue. For example, all query types of class c which have visited the same number of nodes k, i.e., such that $c(\tau) = c$ and $|H(\tau)| = k$ might be grouped together, reducing the number of queues to $\Theta(|\mathcal{C}||\mathcal{N}|)$ or fewer. Alternatively, we will show that one can further reduce overheads by approximately grouping similar query types based on the cumulative number of class $c(\tau)$ files/resources stored at nodes in $H(\tau)$. By grouping them into L levels the number of queues get reduced to $\Theta(|\mathcal{C}|L)$. Intuitively, such queries have seen similar opportunities if files/resources are randomly made available in the network.

Network with random file/resource placement. To better understand when such aggregation makes sense, consider a network where files/resources are randomly and independently available at each node, i.e., at the super-peers and/or their associated subordinate peers. Such independence might make sense in an unstructured network where resources and subordinate associations might be ad hoc. Random placement of files/resources will be modeled as follows. The probability that node i has resource $r \in \mathcal{R}_c$ is given by $\rho^c_{a,i}(r) = \beta^c_i p^c_s(r)$ where $p^c_s(r), r \in \mathcal{R}_c$ is a probability measure capturing the relative availability of class c file/resource r and β^c_i is a number capturing the willingness of node i to store class c files/resources. Note that $(\rho^c_{a,i}(r) : r \in \mathcal{R}_c)$ is *not* a probability measure; instead we require that β^c_i be such that $\rho^c_{a,i}(r) \leq 1$ for all $r \in \mathcal{R}_c$. We let $p^c_q(r), r \in \mathcal{R}_c$ be a probability measure capturing the likelihood a query of type c is for file/resource r, i.e., in terms of our ν_r we have that for all $r \in \mathcal{R}_c$

$$p^c_q(r) = \frac{\nu_r}{\sum_{s\in\mathcal{R}_c} \nu_s}.$$

In summary $p^c_q()$ captures the relative popularity of various queries for resources in class c, while $p^c_a()$ captures the relative availability of various resources of class c and β^c_i the willingness of node i to store class c files/resources. Finally under this network with random file/resource placements the average number of class c resources at node i would be

$$\sum_{r\in\mathcal{R}_c} \rho^c_{a,i}(r) = \sum_{r\in\mathcal{R}_c} \beta^c_i p^c_s(r) = \beta^c_i.$$

Next let us compute $\bar{p^\tau_i}$ the probability that a query of type τ is resolved at node i which in this section will be averaged over random file/resource distributions. One can show that

$$\bar{p^\tau_i} = \frac{\sum_{r\in\mathcal{R}_c} \beta^c_i p^c_s(r) \left(\prod_{j\in H(\tau)}(1-\beta^c_j p^c_s(r))\right)}{\sum_{r\in\mathcal{R}_c} p^c_q(r) \left(1-\prod_{j\in H(\tau)}(1-\beta^c_j p^c_s(r)\right)} \quad (5)$$

Note that although this represents an average over network random file/resource distributions one can show that in a network with large number of files there is a concentration result where this probability is representative of a given realization of the random network. Further it is easy to see that if $\beta^c_i = \beta^c$ for all i then $\bar{p^\tau_i}$ is depends solely on the number of nodes in $H(\tau)$. Thus all queries for class c files/resources that have visited the same number of nodes can be grouped together. One can further roughly approximate the above expression to obtain

$$\bar{p^\tau_i} \approx \frac{\sum_{r\in\mathcal{R}_c} \beta^c_i p^c_s(r) \left(1 - p^c_s(r) \sum_{j\in H(\tau)} \beta^c_j\right)}{\left(\sum_{j\in H(\tau)} \beta^c_j\right) \left(\sum_{r\in\mathcal{R}_c} p^c_q(r) p^c_s(r)\right)}. \quad (6)$$

Note that under this approximation $\bar{p^\tau_i}$ is simply a function of $\sum_{j\in H(\tau)} \beta^c_j$ corresponding to the cumulative average number of files of class c seen at nodes in $H(\tau)$. β^c_j can be estimated by keeping track of the number of distinct queries resolved by node i. Thus as proposed in the sequel, one could conceivably aggregate query types which have seen similar numbers of files in their history and still roughly capture the correct probabilities of query resolutions in the network. This would lead to substantial reductions in complexity.

Realizing backpressure with aggregated types. We formally define aggregation of query types as follows.

Definition 3: A function $a : \mathcal{T} \to \mathcal{A}$ is a *valid aggregation* function if for any $\tau_1, \tau_2 \in \mathcal{T}$ such that $c(\tau_1) = c(\tau_2)$ and $a(\tau_1) = a(\tau_2)$ we have $p^{\tau_1}_i = p^{\tau_2}_i$, for all $i \in (H(\tau_1) \cap H(\tau_2)) \cup (H(\tau_1)^c \cup H(\tau_2)^c)$.

Thus, for example, $a(\tau)$ could denote the number of nodes visited by a query of a given class or number of files seen of a given class, or some other appropriate aggregation criterion.

In this section we focus on a fully connected network, and without loss of generality restrict nodes from forwarding queries to nodes that they have already visited. Since $i \in H(\tau)$ implies that $p^\tau_i = 0$, revisiting a node does not help resolve a query, and since the network is fully connected, it can not help find an alternate route. We partition $\mathcal{T}$ into sets $T_1, T_2, \ldots,$ such that, each $\tau \in T_\ell$ has exactly same $a(\tau)$ and $c(\tau)$, for each index ℓ. We call such indices 'levels'. Let Γ be set of all levels ℓ. Now, each node maintains a queue for each $\ell \in \Gamma$. Let $Q'^\ell_i(t)$ be the total number of queries in level ℓ waiting to be served at node i, at the beginning of each slot, and let $Q'(t) \triangleq (Q'^\ell_i(t) : i \in \mathcal{N}, \ell \in \Gamma)$ represent the network's queue states in slot t. One important outcome of constraining queries from revisiting nodes is that the probability of resolution for all the queries in $Q'^\ell_i(t)$ is the same, say p'^ℓ_i, since otherwise revisiting queries will have probability 0. By analogy to the definition of $e_i(\tau)$ and $E^{-1}_i(\tau)$, define $\psi_i(\ell)$ and $\Psi^{-1}_i(\ell)$ as, $\psi_i(\ell) = \ell'$ if $\forall \tau \in T_\ell, e_i(\tau) \in T_{\ell'}$, and $\Psi^{-1}_i(\ell)$ is its inverse set. We now provide our modified backpressure policy.

Back-pressure algorithm with aggregation: Below is a distributed dynamic throughput optimal policy for a fully connected network. Given $Q'(t) = q'(t)$, each node i does the following,

1) For each neighbor j, it determines

$$\begin{aligned} w^*_{ij}(t) &= \max_{\ell} \left({q'}_i^{\ell}(t) - {q'}_j^{\psi_i(\ell)}(t)(1-{p'}_i^{\ell}) \right) \\ \ell^*_{ij}(t) &= \arg\max_{\ell} \left({q'}_i^{\ell}(t) - {q'}_j^{\psi_i(\ell)}(t)(1-{p'}_i^{\ell}) \right). \end{aligned}$$

2) It finds $j_i^* = \arg\max_j w^*_{ij}(t)$ and lets $\ell_i^* = \ell^*_{ij}(t)$ for $j = j_i^*$,

3) It serves a maximum of μ_i queries from level ℓ_i^* which have not visited node j_i^* on FCFS basis and forwards the unresolved queries to node j_i^*. If the total number of such queries is less than μ_i, then it serves blank queries for the spare services.

Theorem 3: For a fully connected network and valid aggregation function $a()$ the backpressure algorithm with aggregation is throughput optimal.

For proof of the above theorem, see [16]. Note that, as with the basic backpressure policy, the above modified policy is wasteful and can be made work conserving along the lines of work conserving version of the basic backpressure algorithm in Section III-B. Further modifications are required for the case of a general network topology, since a case may arise where a query has already visited all the neighbors of its current node. For such conditions, we present a simple modification. After deciding on j_i^* and ℓ_i^*, node i serves not only queries in queue ${q'}_i^{\ell_i^*}(t)$ which have not visited node j_i^*, but also those queries in ${q'}_i^{\ell_i^*}(t)$ which have visited all its neighbors on FCFS basis. Such a scheme would perform well for networks with large enough degree, since cases where a query has visited all the neighbors would occur rarely.

Approximate aggregation: The total number of queues can be further lowered significantly by aggregating types with roughly similar $a(\tau)$, such that queries in a given level have similar p_i^τ. Here, each node can decide its own levels. Since queries in a coarsely aggregated level at node i may join different queues when forwarded to node j, a function of queue states can be used to compute the weights used by the backpressure algorithm. Note that the error due to approximate aggregation is only in deciding the scheduling and the forwarding policy; each query-class is still accurately provided its promised GoS of γ_c. For this, each node i updates the embedded $\phi(\tau)$ in the query by using $\phi(\tau') = \phi(\tau) + p_i^\tau(1-\phi(\tau))$. Also, instead of learning p_i^τ for each τ, nodes can simply learn and store resolution probability as a function of $a(\tau)$ in a form of look-up table.

VI. Conclusion

To summarize, we provided a novel, distributed, and efficient search policy for unstructured peer-to-peer networks with super-peers. Our backpressure based policy can provide capacity gains of as large as 68% over random walk policies. We also provided modifications to the algorithm that make it amenable to implementation.

Appendix

A. Proof of Theorem 1

To save space, the detailed proof of part *(a)* is given in [16]. It basically shows that when queues are 'well-behaved', the flows over links, in terms of average number of queries transmitted, converge to the desired values. We provide complete proof of part *(b)* of the theorem here. Before proving part *(b)*, we first provide Lemmas 1 and 2 which shall be useful.

Lemma 1: For any randomized policy, we have

$$\mathbf{E}\left[\sum_{\tau,i} (Q_i^\tau(t+1))^2 - (Q_i^\tau(t))^2 \Big| Q(t) \right] \le B - 2\sum_{\tau,i} Q_i^\tau(t) \times \left(\mu_i^\tau(t) - \sum_{j,\tau' \in E_j^{-1}(\tau)} \mu_{ji}^{\tau'}(t)(1-p_j^{\tau'}) - \lambda_i^{c(\tau)} \mathbf{1}\{H(\tau)=\emptyset\} \right)$$

where B is a constant.

Proof: For a given policy, let $F_{ij}^\tau(t)$ be unresolved queries of type τ received at j from i at time t. Thus,

$$\mathbf{E}\left[F_{ij}^\tau(t)\right] \le \sum_{\tau' \in E_i^{-1}(\tau)} \mu_{ij}^{\tau'}(t)(1-p_i^{\tau'}). \tag{7}$$

The evolution of queues for each type τ at node i can be given by,

$$Q_i^\tau(t+1) = \max(Q_i^\tau(t) - \mu_i^\tau(t), 0) + \sum_{j \in N(i)} F_{ji}^\tau(t) + A_i^{c(\tau)}(t)\mathbf{1}\{H(\tau)=\emptyset\}. \tag{8}$$

It can be easily checked that the above implies,

$$\begin{aligned} &(Q_i^\tau(t+1))^2 - (Q_i^\tau(t))^2 \\ &\le (\mu_i^\tau(t))^2 + \left(\sum_{j \in N(i)} F_{ji}^\tau(t) + A_i^{c(\tau)}(t)\mathbf{1}\{H(\tau)=\emptyset\} \right)^2 \\ &-2Q_i^\tau(t) \left(\mu_i^\tau(t) - \sum_{j \in N(i)} F_{ji}^\tau(t) - A_i^{c(\tau)}(t)\mathbf{1}\{H(\tau)=\emptyset\} \right) \end{aligned}$$

Summing over all τ and i, we get

$$\begin{aligned} &\sum_{\tau,i} \left((Q_i^\tau(t+1))^2 - (Q_i^\tau(t))^2 \right) \le B'(t) - 2\sum_{\tau,i} Q_i^\tau(t) \\ &\times \left(\mu_i^\tau(t) - \sum_{j \in N(i)} F_{ji}^\tau(t) - A_i^{c(\tau)}(t)\mathbf{1}\{H(\tau)=\emptyset\} \right) \end{aligned} \tag{9}$$

where

$$B'(t) = \sum_{\tau,i} (\mu_i^\tau(t))^2 + \left(\sum_{j \in N(i)} F_{ji}^\tau(t) \right)^2 + \sum_{\tau,i} \left(A_i^{c(\tau)}(t)\mathbf{1}\{H(\tau)=\emptyset\} \right)^2 \tag{10}$$

since $\sum_{j\in N(i)} F^\tau_{ji}(t)A^{c(\tau)}_i(t)\mathbf{1}\{H(\tau)=\emptyset\} = 0$ as $F^\tau_{ji}(t)\mathbf{1}\{H(\tau)=\emptyset\} = 0$. Here, $\sum_{\tau,i}(\mu^\tau_i(t))^2 \le \sum_i(\mu_i)^2$. Also,

$$\sum_{\tau,i}\left(\sum_{j\in N(i)} F^\tau_{ji}(t)\right)^2 \le \left(\sum_{\tau,i}\sum_{\tau'\in E_j^{-1}(\tau),j}\mu^{\tau'}_{ji}(t)\right)^2 \le \sum_i(\mu_i)^2.$$

Further, $\sum_{\tau,i}\left(A^{c(\tau)}_i(t)\mathbf{1}\{H(\tau)=\emptyset\}\right)^2 = \sum_{c,i}(A^\tau_i(t))^2$. Thus, $\mathbf{E}\left[B'(t)\right] \le 2\sum_i(\mu_i)^2 + \sum_{c,i}\mathbf{E}\left[(A^\tau_i(t))^2\right] \triangleq B$. Thus, by talking expectation on both sides of (9), we get,

$$\sum_{\tau,i}\mathbf{E}\left[(Q^\tau_i(t+1))^2 - (Q^\tau_i(t))^2\right] \le B - 2\sum_{\tau,i}Q^\tau_i(t) \times \mathbf{E}\left[\mu^\tau_i(t) - \sum_{j\in N(i)}F^\tau_{ji}(t) - A^{c(\tau)}_i(t)\mathbf{1}\{H(\tau)=\emptyset\}\right] \quad (11)$$

from which the lemma follows by using (7). ∎

Lemma 2: Given $\lambda \in \Lambda'$, one can obtain a fixed valid assignment $\mu(t) = (\tilde{\mu}^\tau_{ij})$ (and correspondingly, $\mu^\tau_i(t) = \tilde{\mu}^\tau_i$) such that, for some $\epsilon^\tau_i > 0$ and all for all $i \in \mathcal{N}$ and $\tau \in \mathcal{T}$,

$$\tilde{\mu}^\tau_i - \sum_{j,\tau'\in E_j^{-1}(\tau)}\tilde{\mu}^{\tau'}_{ji}(1-p^{\tau'}_j) - \lambda^{c(\tau)}_i\mathbf{1}\{H(\tau)=\emptyset\} = \epsilon^\tau_i.$$

Proof: The definition of Λ allows for exogenous arrivals $\lambda^{c(\tau)}_i$ only for the types τ such that $H(\tau)=\emptyset$. We first generalize it for the hypothetical case where exogenous arrivals are allowed for all types. Consider generalized arrival rates $\tilde{\lambda} = (\tilde{\lambda}^\tau_i : i \in \mathcal{N}, \tau \in \mathcal{T})$.

Definition 4: The *generalized capacity region* $\tilde{\Lambda}$ is set of generalized arrival rates $\tilde{\lambda}$, such that there exists a feasible solution to the following linear constraints on variables $\tilde{f} \triangleq (\tilde{f}^\tau_{ij} : ij \in \mathcal{L}, \tau \in \mathcal{T})$:

1) Capacity constraints: for all $i \in \mathcal{N}$,
$$\sum_{j,\tau}\tilde{f}^\tau_{ji} + \sum_\tau \tilde{\lambda}^\tau_i \le \tilde{\mu}_i \quad (12)$$
2) Flow conservation constraints with resolution at nodes: for all $i \in \mathcal{N}$ and $\tau \in \mathcal{T}$,
$$\sum_{\tau'\in E_i^{-1}(\tau)}(1-p^{\tau'}_i)\left(\sum_j \tilde{f}^{\tau'}_{ji} + \tilde{\lambda}^{\tau'}_i\right) = \sum_j \tilde{f}^\tau_{ij} \quad (13)$$
3) Non-negativity constraints: for all $(i,j) \in \mathcal{L}$ and $\tau \in \mathcal{T}$,
$$\tilde{f}^\tau_{ij} \ge 0. \quad (14)$$

Properties of $\tilde{\Lambda}$*:* a) For each $\lambda \in \Lambda$, we have a $\tilde{\lambda} \in \tilde{\Lambda}$ such that $\tilde{\lambda}^\tau_i = \lambda^{c(\tau)}_i\mathbf{1}\{H(\tau)=\emptyset\}$. b) $\tilde{\Lambda}$ is a convex set. c) For each node i and type τ, consider matrix $\bar{\delta}^\tau_i \in \mathcal{N}\times\mathcal{T}$ which has value 1 only in $(i,\tau)^{\text{th}}$ position, and 0 everywhere else. For each i and τ, there exist a constant $\gamma^\tau_i > 0$ such that $\gamma^\tau_i\bar{\delta}^\tau_i \in \tilde{\Lambda}$.

Proof of Properties of $\tilde{\Lambda}$*:* a) follows from definition of Λ, since putting $\tilde{\lambda}^\tau_i = \lambda^{c(\tau)}_i\mathbf{1}\{H(\tau)=\emptyset\}$ in the constraints for $\tilde{\Lambda}$, satisfies the constraints of Λ. b) follows from linearity of constraints of $\tilde{\Lambda}$. c) follows from our description of p, the fact that the network is connected, and that $\mu_i > 0$.

Now, consider $\lambda \in \Lambda'$. By definition, $\exists\epsilon > 0$ such that $(1+\epsilon)\lambda \in \Lambda$. Using property a), find $\tilde{\lambda}$ such that $(1+\epsilon)\tilde{\lambda} \in \tilde{\Lambda}$, and $\tilde{\lambda} = (\lambda^{c(\tau)}_i\mathbf{1}\{H(\tau)=\emptyset\})$. Thus, by convexity of $\tilde{\Lambda}$ and property c), $\tilde{\lambda} + \epsilon'\sum_{i,\tau}\gamma^\tau_i\bar{\delta}^\tau_i \in \tilde{\Lambda}$, where $\epsilon' = \frac{1}{|\mathcal{N}||\mathcal{C}|}(1-\frac{1}{1+\epsilon})$. Putting $\epsilon^\tau_i = \epsilon'\gamma^\tau_i$, we get $\tilde{\lambda} + \bar{\epsilon} \in \tilde{\Lambda}$, where $\bar{\epsilon} = (\epsilon^\tau_i)$.

Now, obtain a feasible solution $\tilde{f}' = (\tilde{f'}^\tau_{ij}, ij \in \mathcal{L}, \tau \in \mathcal{T})$ for constraints (12)-(14) for general arrivals $\tilde{\lambda} + \bar{\epsilon}$, where $\tilde{\lambda} = (\lambda^{c(\tau)}_i\mathbf{1}\{H(\tau)=\emptyset\})$. Using this solution, set for all i and τ

$$\tilde{\mu}^\tau_i = \sum_j \tilde{f'}^\tau_{ji} + \lambda^{c(\tau)}_i\mathbf{1}\{H(\tau)=\emptyset\} + \epsilon^\tau_i, \quad (15)$$

and

$$\tilde{\mu}^\tau_{ij} = \mu^\tau_i\frac{\tilde{f'}^{e_i(\tau)}_{ij}}{\sum_{j'}\tilde{f'}^{e_i(\tau)}_{ij'}}. \quad (16)$$

Note that, this assignment of $\tilde{\mu}^\tau_{ij}$ (and corresponding π^τ_{ij}) is valid, since from constraint (12) we get $\sum_{j,\tau}\mu^\tau_{ij} \le \mu_i$ for all i. Using these assignments and constraint (13), and one can check that, for all $i \in \mathcal{N}$ and $\tau \in \mathcal{T}$

$$\sum_{\tau'\in E_i^{-1}(\tau)}(1-p^{\tau'}_i)\tilde{\mu}^{\tau'}_{ij} = \tilde{f'}^\tau_{ij}. \quad (17)$$

Putting this in (15), we get that for all $i \in \mathcal{N}$ and $\tau \in \mathcal{T}$

$$\sum_j\sum_{\tau'\in E_j^{-1}(\tau)}\tilde{\mu}^{\tau'}_{ji}(1-p^{\tau'}_j) + \lambda^{c(\tau)}_i\mathbf{1}\{H(\tau)=\emptyset\} + \epsilon^\tau_i = \tilde{\mu}^\tau_i. \quad (18)$$

∎

Proof of Theorem 1 part (b): Under a fixed randomized policy, $Q(t)$ forms a Markov chain. Consider candidate Lyapunov function $L(Q) = \sum_{i,\tau}(Q^\tau_i)^2$. Thus, if we show that drift $\Delta Q(t) \triangleq \mathbf{E}\left[\sum_{\tau,i}(Q^\tau_i(t+1))^2 - (Q^\tau_i(t))^2\middle|Q(t)\right]$ is negative for all but finite set values of $Q(t)$, it would imply that $L(Q)$ is a Lyapunov function, thus proving that the Markov chain is positive recurrent, from which stability follows. Lemma 1 provides an upper-bound on $\Delta Q(t)$. Substituting the result of Lemma 2 in this bound, we get

$$\Delta Q(t) \le B - 2\sum_{\tau,i}Q^\tau_i(t)\epsilon^\tau_i \quad (19)$$

where B is a constant, and $\epsilon^\tau_i > 0, \forall i,\tau$. Thus, for the randomized policy given in Lemma 2, drift $\Delta Q(t)$ is negative for all but finite $Q(t)$, therefore obtaining stability. ∎

B. Proof of Theorem 2

Before proving Theorem 2, we first provide Lemma 3. We then use Lemmas 1 and 3 to prove the theorem.

Lemma 3: For the given back pressure algorithm, if λ is in the interior of Λ, then, for some $\bar{\epsilon} > 0$,

$$\sum_{\tau,i} Q_i^\tau(t) \left(\mu_i^{*\tau}(t) - \sum_{j,\tau' \in E_j^{-1}(\tau)} \mu_{ji}^{*\tau'}(t)(1-p_j^{\tau'}) \right) \geq \sum_{\tau,i} Q_i^\tau(t) \left(\lambda_i^{c(\tau)} \mathbf{1}\{H(\tau)=\emptyset\} + \epsilon_i^\tau \right) \quad (20)$$

Proof: From Lemma 2, there exists a stationary static policy, that does not depend on $Q(t)$, and determines valid fixed service rates $\tilde{\mu}_{ij}^\tau$ such that

$$\sum_{\tau,i} Q_i^\tau(t) \left(\tilde{\mu}_i^\tau - \sum_{j,\tau' \in E_j^{-1}(\tau)} \tilde{\mu}_{ji}^{\tau'}(1-p_j^{\tau'}) \right) = \sum_{\tau,i} Q_i^\tau(t) \left(\lambda_i^{c(\tau)} \mathbf{1}\{H(\tau)=\emptyset\} + \epsilon_i^\tau \right), \quad (21)$$

By rearranging terms of L.H.S., we get,

$$\begin{aligned} &\sum_{\tau,i} Q_i^\tau(t) \left(\sum_j \tilde{\mu}_{ij}^\tau - \sum_{j,\tau' \in E_j^{-1}(\tau)} \tilde{\mu}_{ji}^{\tau'}(1-p_j^{\tau'}) \right) \\ &= \sum_{(i,j)\in\mathcal{L},\tau} \tilde{\mu}_{ij}^\tau \left(Q_i^\tau(t) - Q_j^{e_i(\tau)}(t)(1-p_i^\tau) \right) \\ &\leq \sum_{(i,j)\in\mathcal{L},\tau} \tilde{\mu}_{ij}^\tau w_{ij}^*(t) \leq \sum_{(i,j)\in\mathcal{L},\tau} \mu_{ij}^{*\tau}(t) w_{ij}^*(t), \end{aligned} \quad (22)$$

where the last inequality follows from the choice of $\mu_{ij}^{*\tau}(t)$ by the back pressure algorithm that maximizes the upper bound by assigning the entire service rate of μ_i to a link that has maximum weight $w_{ij}^*(t)$. This also implies,

$$\begin{aligned} &\sum_{(i,j)\in\mathcal{L},\tau} \mu_{ij}^{*\tau}(t) w_{ij}^*(t) \\ &= \sum_{(i,j)\in\mathcal{L},\tau} \mu_{ij}^{*\tau}(t) \left(Q_i^\tau(t) - Q_j^{e_i(\tau)}(t)(1-p_i^\tau) \right) \\ &= \sum_{\tau,i} Q_i^\tau(t) \left(\mu_i^{*\tau}(t) - \sum_{j,\tau' \in E_j^{-1}(\tau)} \mu_{ji}^{*\tau'}(t)(1-p_j^{\tau'}) \right). \end{aligned} \quad (23)$$

From (21),(22) and (23), the lemma follows. ∎

Proof of Theorem 2: Since the basic backpressure algorithm is a state dependent randomized policy, Lemma 1 implies that,

$$\mathbf{E}\left[\sum_{\tau,i} (Q_i^\tau(t+1))^2 - (Q_i^\tau(t))^2 \Big| Q(t) \right] \leq B - 2\sum_{\tau,i} Q_i^\tau(t) \times \left(\mu_i^{*\tau}(t) - \sum_{j,\tau' \in E_j^{-1}(\tau)} \mu_{ji}^{*\tau'}(t)(1-p_j^{\tau'}) - \lambda_i^{c(\tau)} \mathbf{1}\{H(\tau)=\emptyset\} \right) \quad (24)$$

Note that $Q(t)$ forms a Markov chain for the back pressure algorithm since $\mu_{ij}^{*\tau}(t)$ are a function of $Q(t)$. Thus, again, if we show that drift $\Delta Q(t)$ is negative for all but finite values of $Q(t)$, it would imply that $L(Q) = \sum_{i,\tau} (Q_i^\tau)^2$ is a Lyapunov function, thus proving that system is stable. Lemma 3 shows that the bound on $\Delta Q(t)$ is only more negative compared to policy used in establishing stability for each $\lambda \in \Lambda'$ in Theorem 1. Thus, from Lemma 3 and (24), we get $\Delta Q(t) \leq B - 2\sum_{\tau,i} Q_i^\tau(t)\epsilon_i^\tau$, which is negative for all but finite values of $Q(t)$. ∎

C. Proof of Corollary 1

Consider all states of $Q(t)$ such that $Q_i^\tau(t) \geq \mu_i, \forall i, \tau$. For all these states, the work conserving back pressure policy is equivalent to the basic backpressure algorithm. Thus, from proof of Theorem 2, if $\lambda \in \Lambda'$, the work conserving back pressure policy has negative drift for all but finite values of $Q(t)$. ∎

REFERENCES

[1] Wikipedia, "Peer-to-peer — Wikipedia, the free encyclopedia." http://en.wikipedia.org/wiki/Peer-to-peer, 2011.

[2] I. Stoica *et al.*, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Trans. Networking*, vol. 11, no. 1, pp. 17–32, 2003.

[3] X. Li and J. Wu, "Searching techniques in peer-to-peer networks," in *Handbook of Theoretical and Algorithmic Aspects of Ad Hoc, Sensor, and Peer-to-Peer Networks, CRC Press*, 2004.

[4] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks," in *Proc. IEEE INFOCOM*, 2004.

[5] C. Gkantsidis, M. Mihail, and A. Saberi, "Hybrid search schemes for unstructured peer to peer networks," in *Proc. IEEE INFOCOM*, 2005.

[6] S. Ioannidis and P. Marbach, "On the design of hybrid peer-to-peer systems," in *Proc. ACM SIGMETRICS*, 2008.

[7] P. Patankar *et al.*, "Peer-to-peer unstructured anycasting using correlated swarms," in *Proc. ITC*, 2009.

[8] R. Gupta and A. Somani, "An incentive driven lookup protocol for chord-based peer-to-peer (p2p) networks," in *International Conference on High Performance Computing*, 2004.

[9] D. Menasche, L. Massoulie, and D. Towsley, "Reciprocity and barter in peer-to-peer systems," in *Proc. IEEE INFOCOM*, 2010.

[10] B. Mitra *et al.*, "How do superpeer networks emerge?," in *Proc. IEEE INFOCOM*, 2010.

[11] D. Karger and M. Ruhl, "Simple efficient load balancing algorithms for peer-to-peer systems," in *Proc. 16th ACM SPAA*, 2004.

[12] B. Yang and H. Garcia-Molina, "Designing a super-peer network," in *Proc. IEEE ICDE*, 2003.

[13] M. J. Neely, E. Modiano, and C. E. Rohrs, "Dynamic power allocation and routing for time varying wireless networks," in *IEEE INFOCOM*, 2003.

[14] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Trans. Automatic Control*, vol. 37, pp. 1936–1948, 1992.

[15] S. Asmussen, *Applied probability and queues*. Springer, 1987.

[16] V. Shah, G. de Veciana, and G. Kesidis, "Learning to route queries in unstructured p2p networks: Achieving throughput optimality subject to query resolution constraints." Technical Report, available at http://www.ece.utexas.edu/%7Egustavo/TechReportSVK11.pdf, 2011.