A Stable Approach for Routing Queries in Unstructured P2P Networks

Virag Shah, Gustavo de Veciana, and George Kesidis

***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 stabilize the query load subject to a grade of service constraint, i.e., 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, estimating parameters, and adaptation to class-based query resolution probabilities and traffic loads are studied.**

***Index Terms*—peer-to-peer, search, stability, backpressure, random walk.**

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.

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 that 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

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

Virag Shah and Gustavo de Veciana are with ECE Department at The University of Texas at Austin, Austin, TX 78712 USA (e-mail: virag@utexas.edu, gustavo@ece.utexas.edu).

George Kesidis is with EE and CSE departments at The Pennsylvania State University, University Park, PA 16802 USA (e-mail: gik2@psu.edu)

This is an extended version of a paper presented at INFOCOM 2012, Orlando, Florida, USA.

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 which stabilizes the system for all query loads within a 'capacity region', i.e., the set of loads for which stability is feasible. Essentially, our policy is a biased random walk where forwarding decision for each query is based on instantaneous query loads at super-peers. To balance the load across heterogeneous super-peers, the policy aims at reducing the differential backlog at neighboring super-peers, while taking into account the class and history information to improve the query's resolvability.

Our policy draws upon standard backpressure routing algorithm, which is used to achieve stability in packet switching networks, e.g., see [13], [14]. In previously studied backpressure based systems, the goal is to deliver packets to the corresponding destinations. By contrast, our aim is to provide a grade of service in resolving queries with no fixed destinations. The random nature of the location of query resolution in the network leads us to deal with expected queue backlog instead of current queue backlog. Further, in P2P systems, the probability of resolution of a query at a given node depends on the query's history, i.e., the path that led it to the current node. These characteristics of P2P systems are not captured in previous works on backpressure by Tassiulas and Ephremides [13] and the subsequent enhancements, see e.g., [14]–[21].

To summarize, our approach differs from standard work on backpressure in that we incorporate the following different issues that arise in P2P search: (a) we model the uncertainty in the locations where a query may be resolved depending upon where the file/object of interest are placed, (b) we guarantee a grade of service to each query under such uncertainties, (c) we incorporate the information about a query's resolvability available through the knowledge of its history.

We also propose several natural enhancements to our backpressure based query routing policy. By contrast to previous works on backpressure such as [15]–[19] and citations therein, these enhancements are also driven by P2P query routing setting. For example, in order to reduce delays previous works develop algorithms which prefer shorter paths over longer ones by explicitly accounting for the hop length of various paths [15], or by finding good routes towards destinations using artificial 'shadow' queues which operate at larger loads to build gradients [17]. In our P2P query routing setting the destination of a query is not known a priori. We reduce delays via a simple 'work conserving' policy which efficiently uses available resources in routing queries at each node. We further propose a state aggregation policy aimed at reducing the complexity arising from the need to track the history of currently unresolved search queries.

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

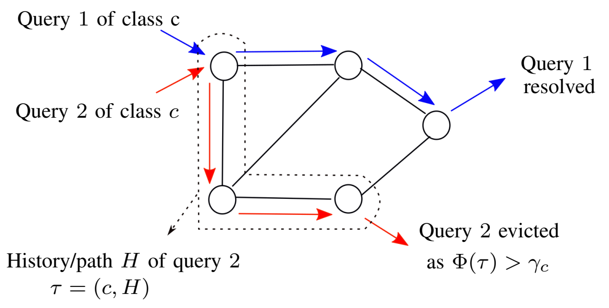
Query 1 of class c
Query 2 of class c
Query 1 resolved
Query 2 evicted as $\Phi(\tau) > \gamma_c$
History/path H of query 2 $\tau = (c, H)$

Fig. 1. A network of super-peers $G = (\mathcal{N}, \mathcal{L})$. Queries of a given class traverse potentially different routes. A query either gets resolved or gets evicted from the network upon receiving a grade of service.

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 stability subject to a Grade of Service (GoS) constraint on query resolution. 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 stable 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 stable 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 stable 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 $\mathcal{N}$ (nodes) 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 set of neighbors of super-peer i. Note that subordinate peers of the hybrid network are not explicitly represented, but 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 positive integer number of queries it is willing to resolve/forward in each slot.

We assume that super peers keep a record of files/resources available at subordinate peer. This information is communicated to super peers when a subordinate peer joins a super peer. Subordinate peers may initiate a query request at a super peer, but do not participate in forwarding or query resolution. Let

$\mathcal{R}$ be the set of all files/resources that might be queried on the network, and $\mathcal{C}$ a predefined set of resource classes. For each $c \in \mathcal{C}$, let $\mathcal{R}^c \subset \mathcal{R}$ be the files/resources belonging to class c. 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 $r \in \mathcal{R}$. We say a query is a class c query if the resource it is seeking is in $\mathcal{R}^c$. Let $A_i^c(t)$ denote 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, with finite second moments and independent across slots, thus we have well defined arrival rates denoted by $\lambda \triangleq (\lambda_i^c : \forall i \in \mathcal{N}, c \in \mathcal{C})$ 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 a node can resolve such a query depends not only on its class but also its *history*, i.e., the set of nodes it visited in the past. Note that the history is not ordered. 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 in resolving 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.

Notation for tracking history and class of a query: We capture such behavior for different classes by keeping track of the history of a query, i.e., the subset of nodes already visited, or equivalently an element of $\mathcal{H}$ which is the powerset of $\mathcal{N}$. Note, history captures only the set of visited nodes and not the order in which they are visited. The 'type' τ of a query keeps track of both, its class c and its history H, i.e., $\tau = (c, H) \in \mathcal{T} \triangleq \mathcal{C} \times \mathcal{H}$. Let $c(\tau) = c$ and $H(\tau) = H$ represent class and history of the associated query.

Further, we let $e_i(\tau)$ represent the resulting type once a query of type τ is serviced by node i, i.e., $e_i(\tau) = (c(\tau), H(\tau) \cup \{i\})$. Also, we let $E_i^{-1}(\tau)$ denote the inverse set of $e_i(\tau)$, i.e., $E_i^{-1}(\tau) = \{(c(\tau), H) : H \cup \{i\} = H(\tau)\}$. $E_i^{-1}(\tau)$ captures the set of all possible histories H that lead to τ. Note that, since history H is unordered, if a query revisits a node its history is unchanged. Similarly, if a query revisits a node its type is unchanged, i.e., if $i \in H(\tau)$ then $e_i(\tau) = \tau$.

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 by the 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 is able to resolve. In the sequel it will be useful to formally relate these quantities to, (1) ν_r the fractions of queries for resource $r \in \mathcal{R}$, (2) $\mathcal{R}^c$ the resources of class $c \in \mathcal{C}$, and (3) R_i^c the resources in class c held by node i. Indeed the probability a type τ query is resolved at i is given by

TABLE I
A SUMMARY OF NOTATIONS

$G = (\mathcal{N}, \mathcal{L})$	Network represented by graph G with nodes $\mathcal{N}$ representing super-peers and $\mathcal{L}$ representing overlay links
$N(i)$	Neighbors of node i
μ_i	service rate/altruism of node i
c; $\mathcal{C}$	A class of resources; set of all classes
$\mathcal{R}$; $\mathcal{R}^c$; R_i^c	Set of all files/resources; set of resources belonging to class c; set of resources in class c available at super-peer i
$A_i(t)$; $A_i^c(t)$	Arrival process of queries at node i; arrival process of class c queries at node i
λ_i^c; λ	Arrival rates; $\lambda = (\lambda_i^c : \forall i \in \mathcal{N}, c \in \mathcal{C})$
ν_r	popularity of resource r
H; τ	History: set of visited nodes; query type: $\tau = (c, H)$
$e_i(\tau)$; $E_i^{-1}(\tau)$	$E_i^{-1}(\tau) = (c(\tau), H(\tau) \cup \{i\})$; inverse set of $e_i(\tau)$
p_i^τ	Probability that a query of type τ exits the network upon service at node i, either due to query resolution or due to eviction upon receiving the grade of service
$\phi(\tau)$	a priori probability that a typical query of class $c(\tau)$ is resolved at a node in $H(\tau)$
γ_c	Grade of service: a query is evicted if $\phi(\tau) > \gamma_c$
$Q_i^\tau(t)$; $Q(t)$	Number of waiting queries of type τ at node i in slot t; $Q(t) = (Q_i^\tau(t) : i \in \mathcal{N}, \tau \in \mathcal{T})$
$\pi_{ij}^\tau(t)$; $\pi(t)$	Probability that a query served by node i at time t belongs to type τ, and is forwarded to node $j \in N(i)$, if unresolved; $\pi(t) = (\pi_{ij}^\tau(t) : (i,j) \in \mathcal{L}, \tau \in \mathcal{C})$
$\mu_{ij}^\tau(t)$; $\mu(t)$	$\mu_{ij}^\tau(t) = \mu_i \pi_{ij}^\tau(t)$; $\mu(t) = (\mu_{ij}^\tau(t) : (i,j) \in \mathcal{L}, \tau \in \mathcal{T})$
f_{ij}^τ; f	Flow of type τ from node i to node j; $f = \Big(f_{ij}^\tau : (i,j) \in \mathcal{L}, \tau \in \mathcal{T}\Big)$
Λ; Λ'	Capacity region; interior of capacity region

$$p_i^\tau = \frac{\sum_r \nu_r \mathbf{1}\Big\{r \in R_i^{c(\tau)}\Big\} \mathbf{1}\Big\{r \notin \cup_{j \in H(\tau)} R_j^{c(\tau)}\Big\}}{\sum_r \nu_r \mathbf{1}\Big\{r \in \mathcal{R}^{c(\tau)}\Big\} \mathbf{1}\Big\{r \notin \cup_{j \in H(\tau)} R_j^{c(\tau)}\Big\}}, \qquad (1)$$

i.e., the ratio of the sum demand, i.e., ν_r, for type τ class $c(\tau)$ files/resources which are present at node i and were not present at nodes in its history $H(\tau)$ and that for $c(\tau)$ resources that were not present at nodes in its history $H(\tau)$.

Recall, type of a query does not change upon revisits. Further, from (1), if a query has already visited a node i in past, i.e., if $i \in H(\tau)$, then its probability of resolution at node i, i.e., p_i^τ, is equal to 0. This is intuitive, since if a query has already visited a node and it is still not resolved then the corresponding file is not present at the node. This further reinforces the history dependent nature of a query's resolvability.

Grade of Service on Query Resolution: A standard mechanism adopted in P2P systems is to evict 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 a priori probability that a typical query of class $c(\tau)$ is resolved upon visiting nodes in $H(\tau)$, i.e.,

$$\phi(\tau) = \frac{\sum_r \nu_r \mathbf{1}\Big\{r \in \cup_{j \in H(\tau)} R_j^{c(\tau)}\Big\}}{\sum_r \nu_r \mathbf{1}\Big\{r \in \mathcal{R}^{c(\tau)}\Big\}} \qquad (2)$$

the ratio of weighted class $c(\tau)$ files/resources seen in $H(\tau)$ over the total weighted documents in $\mathcal{R}^{c(\tau)}$. We propose removing a query 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. Other possible GoS metrics will be discussed in the Section V-C. Note $\phi(\tau)$ does not depend on the path 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 $\phi(\tau) \geq \gamma_{c(\tau)}$, 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 simply reflect class-based probabilities of query resolution for various types of queries at the nodes but also the GoS requirement, or eviction criterion, implemented by underlying query resolution protocol.

Network State and Routing Policies: We assume that arrivals occur at the end of each slot and 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' as described below. 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.

Recall that each super-peer node i has an associated service (altruism) rate[1] μ_i which the policy can use in each slot of as follows:

1) It chooses no more than μ_i queries currently at node i for service on the slot;
2) If a query is unresolved at node i, it determines which neighbor $j \in N(i)$ the query should be forwarded to.

We say a policy is *ergodic* if sample paths of $Q(t)$ are ergodic and a steady state exists.

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.
2) For each unresolved 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 query, among μ_i queries served at node i, 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. 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.

[1]We have assumed that μ_i is an integer. One way to deal with fractional service rate is to keep service random. For example, $\mu_i = 2.6$ can be modeled as randomly choosing 2 and 3 with probability 0.4 and 0.6 respectively, in each slot. All the results herein hold invariably with some additional technicalities in the proofs.

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 $j \in N(i)$ and $\tau \in \mathcal{T}$, and 0 for others.

A randomized policy is called *fixed* if $\pi(t)$ does not depend on t or $q(t)$.

III. Stable Query Forwarding Policy

In this section, we will propose a query scheduling and forwarding policy that ensures the GoS for each class, is distributed, easy to implement, and is stable. 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 [14], 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 [13], [22]. For a given queue process $Q_i^\tau(t)$, let $g_i^\tau(\alpha)$ denote its 'overflow' function

$$g_i^\tau(\alpha) = \limsup_{t\to\infty} \frac{1}{t} \sum_{t'=1}^{t} \mathbf{1}\{Q_i^\tau(t') > \alpha\} \tag{3}$$

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 is 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 $i \in \mathcal{N}$

$$\sum_{j,\tau} f_{ji}^\tau + \sum_c \lambda_i^c \leq \mu_i; \tag{4}$$

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

$$\sum_j f_{ij}^\tau = \sum_{\tau' \in E_i^{-1}(\tau)} (1 - p_i^{\tau'})(\sum_j f_{ji}^{\tau'} + \lambda_i^{c(\tau')} \mathbf{1}\{H(\tau') = \emptyset\}); \tag{5}$$

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

$$f_{ij}^\tau \geq 0. \tag{6}$$

We refer to f as flow variables, where (4) ensures that the incoming flow to a node is less than its service rate and (5) 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. These are different than the standard multicommodity flow conservation laws in the sense that our conservation equations are designed to capture the following aspects arising in P2P search systems: (a) history dependent probability of query resolution at each node, (b) updates in 'types' of queries as they get forwarded to different nodes, (c) computing the quality of service received by query via its history and designing an appropriate exit strategy upon receiving enough service.

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

Theorem 1. *(Capacity Region)*

(a) If for a given arrival rate vector λ there exists a state dependent randomized 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 above result is that each query should eventually leave the network, and revisits to nodes (while allowed) have a zero probability of resolving the query.

B. Stable 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. Further, designing a stable search algorithm is now a challenge since, while the routing decisions are to be based on instantaneous queue loads at the neighbors, the decisions themselves affect the type/queue to which a query belongs. 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) $p_i^\tau, \tau \in \mathcal{T}$, 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, instead, 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 achieves stability for any $\lambda \in \Lambda'$.*

The proof of the above theorem is provided in Appendix B. The proof handles the evolution of query types and the randomness in resolution of queries by incorporating expected queue backlogs into Lyapunav drifts. The basic backpressure algorithm, though stable, is highly 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 the 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 stable as well. 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 total service rate, the work conserving policy serves the queries in second highest backlogged queue, 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

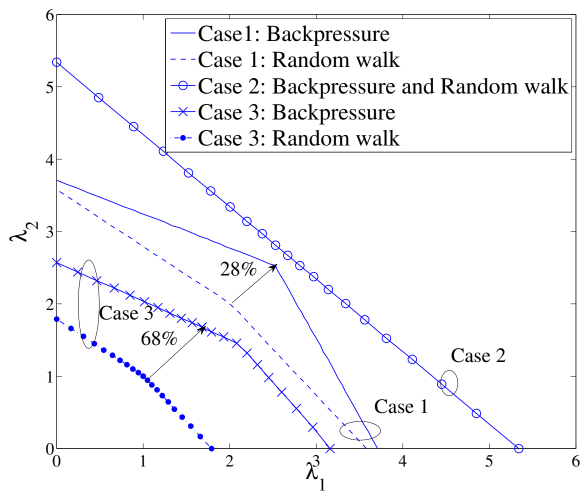

Fig. 2. Boundaries of capacity regions for the stable backpressure algorithm and random walk policy for the 3 cases.

them on an FCFS basis and forwards unresolved ones to j_i^{*k}.

Corollary 1. *The above work conserving backpressure policy is achieves stability for any $\lambda \in \Lambda'$.*

The proof is provided in Appendix C.

IV. Numerical Results and Simulations

In this section, we numerically evaluate the gains in the capacity region achievable by our stable backpressure algorithms versus that a baseline random walk policy. We consider a fully connected network with 6 nodes. Let $\mathcal{N} = \{1, 2, \dots, 6\}$. Since a super-peer network is designed to be highly connected in practice, a fully connected network might be a good representative of the practice. We consider two query-classes, c_1 and c_2. We assume that arrival rates for a given class is same at all the nodes, say λ_1 for class c_1 and λ_2 for class c_2. This reduces the dimension of the capacity region from 12 to 2, making it easier to study. Further, the parameters for the GoS, viz., γ_c, are set to 0.9 for both the classes.

In the baseline random walk policy, upon service, each node forwards an unresolved query to one of the neighbors chosen uniformly at random. Since, in a fully connected network, allowing queries to revisit nodes provides no advantages, queries are forwarded to only those nodes which are not previously visited. As with backpressure, whose achievable capacity region is given by Definition 2, we can characterize the achievable capacity region for the random walk policy. It is the set of arrival rates λ that satisfy the constraints (4)-(6), along with additional constraints that ensure that the outgoing flows of each type at each node are uniformly divided among unvisited nodes. Formally, these are given by,

Random-forwarding constraints: for all $i \in \mathcal{N}$, $\tau \in \mathcal{T}$ and for all $j_1, j_2 \in N(i)$ and $j_1, j_2 \notin H(\tau)$, $f_{ij_1}^{\tau} = f_{ij_2}^{\tau}$.

Constraints for avoiding revisits: for all $i \in \mathcal{N}$, $\tau \in \mathcal{T}$ and $j \in H(\tau)$, $f_{ij}^{\tau} = 0$.

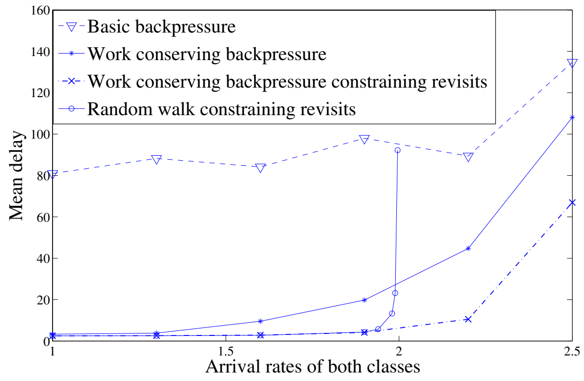

Fig. 3. Delay performance of the backpressure algorithms and random walk for Case 1.

We consider the following three cases, see Fig. 2.

Case 1: $\mu_i = 10$ for all $i \in \mathcal{N}$. For all the types of class c_1, $p_i^{\tau} = 0.6$ for $i \in \{1, 2, 3\}$ and $p_i^{\tau} = 0.1$ for the remaining nodes. For all the types of class c_2, $p_i^{\tau} = 0.1$ for $i \in \{1, 2, 3\}$ and $p_i^{\tau} = 0.6$ for the remaining nodes.

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 $\mu_i = 5$ for the remaining nodes. p are same as in Case 1.

We assume the same exit strategy for both policies to ensure the same GoS, which is captured in the vector p itself – see Section II. Figure 2 shows significant capacity gains for Cases 1 and 3. It also shows that, when μ_i and p_i^{τ} are homogenous over nodes, as in Case 2, the random walk is sufficient to balance the load among the nodes and achieve capacity, a fact that can be easily proven analytically as well. 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 backpressue are significant, particularly when there is heterogeneity among nodes in their service rates, i.e., their altruism, as well.

We now compare the delay performance of the backpressure algorithms and random walk under Case 1. Fig. 3 exhibits the mean delay 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 stable, 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. In order to ensure unbiased estimates can 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 and types it could see and thus allow for unbiased estimates. All queries which are not marked 'RW' are treated according to our backpressure policy based on the estimated query resolution probabilities. A node i receives $O(t\epsilon)$ 'RW' marked samples in time t. Thus, standard deviation in the estimation error is $O(\sqrt{\frac{1}{\epsilon t}})$. Thus the error is small for large enough t. If the contents are static, one may discontinue the estimation process after large enough time t, in which case the time-averaged performance of the policy remains unchanged.

Alternatively, to allow persistent tracking of changes in resolution probabilities, we may estimate the query resolution probabilities via samples provided from a control algorithm, without using a separate unbiased random walk. The convergence of estimation and stability of the system can be jointly obtained via stochastic approximation framework [23] under time scale separation between content dynamics and search dynamics.

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 modification and approximations that considerably reduce the overheads, albeit with some penalty in the performance. The key idea is to define equivalence classes 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 might be grouped together, reducing the number of queues to $\Theta(|\mathcal{C}||\mathcal{N}|)$ or better. Alternatively we will show one can further reduce overheads by approximately grouping similar query types based on their classes c and the cumulative number of class c files/resources they have seen in nodes in $H(\tau)$, reducing the number of queues to $\Theta(|\mathcal{C}|L)$ where L is a set of quantization levels. Intuitively such queries have seen similar opportunities if files/resources are randomely 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 superpeers 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 $\rho^c_{a,i}(r), r \in \mathcal{R}^c$ is *not* a probability measure but we require that β^c_i be such that $\rho^c_{a,i}(r) \leq 1$. 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 $p^c_s()$ 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. Let R^c_j be the random set of files of class c stored at node j. Thus, the probability that a query R of type τ is resolved at node i, given that it could not be resolved at nodes $H(\tau)$ is

$$\begin{aligned}\bar{p^\tau_i} &= \Pr\left(R \in R^c_i | R \notin \cup_{j\in H(\tau)} R^c_j\right)\\ &= \frac{\Pr\left(R \in R^c_i, R \notin \cup_{j\in H(\tau)} R^c_j\right)}{\Pr\left(R \notin \cup_{j\in H(\tau)} R^c_j\right)}.\end{aligned}$$

By conditioning with respect to event $R = r$ for each $r \in \mathcal{R}^c$, we get

$$\bar{p^\tau_i} = \frac{\sum_{r\in\mathcal{R}^c} \begin{array}{r}\Pr(R=r)\Pr\left(R\in R^c_i|R=r\right)\\ \times\Pr\left(R \notin \cup_{j\in H(\tau)} R^c_j | R\in R^c_i, R=r\right)\end{array}}{\sum_{r\in\mathcal{R}^c}\Pr(R=r)\Pr\left(R \notin \cup_{j\in H(\tau)} R^c_j|R=r\right)}. \quad (7)$$

By substituting values for each probabilities we get

$$\bar{p^\tau_i} = \frac{\sum_{r\in\mathcal{R}^c} p^c_q(r)\beta^c_i p^c_s(r)\prod_{j\in H(\tau)}\left(1-\beta^c_j p^c_s(r)\right)}{\sum_{r\in\mathcal{R}^c} p^c_q(r)\prod_{j\in H(\tau)}\left(1-\beta^c_j p^c_s(r)\right)}. \quad (8)$$

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

$$\bar{p_i^\tau} \approx \frac{\sum_{r\in\mathcal{R}^c} p_q^c(r)\beta_i^c p_s^c(r)\left(1-p_s^c(r)\sum_{j\in H(\tau)}\beta_j^c\right)}{\sum_{r\in\mathcal{R}^c} p_q^c(r)\left(1-p_s^c(r)\sum_{j\in H(\tau)}\beta_j^c\right)}. \quad (9)$$

Note that under this approximation $\bar{p_i^\tau}$ is simply a function of $\sum_{j\in H(\tau)}\beta_j^c$ corresponding to the cumulative average number of files of class c seen at nodes in $H(\tau)$. 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.

Given guidelines on how to aggregate query-types, we now provide modifications to the backpressure algorithm, needed to perform well under aggregation.

Assumption 1. *For τ such that $i \in H(\tau)$, p_i^τ depends on $H(\tau)$ only through $f(\tau)$.*

Here, $f(\tau)$ could be number of nodes visited or number of files seen or some other aggregation technique. For now, we focus on a fully connected network. First, we restrict nodes from forwarding queries to nodes that they have already visited, since $i \in H(\tau)$ implies that $p_i^\tau = 0$, revisiting a node does not help resolve a query, except by possibly finding an alternate route. Next, we partition $\mathcal{T}$ into sets $T_1, T_2, \ldots$, such that, each $\tau \in T_\ell$ has exactly same $f(\tau)$ 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 ${Q'}_i^\ell(t)$ be the total number of queries in level ℓ waiting to be served at node i, at the beginning of each slot, and let $Q'(t) \triangleq ({Q'}_i^\ell(t) : i \in \mathcal{N}, \ell \in \Gamma)$ 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 ${Q'}_i^\ell(t)$ is the same, say ${p'}_i^\ell$, since otherwise revisiting queries will have probability 0. By analogy to the definition of $e_i(\tau)$ and $E_i^{-1}(\tau)$, define $\psi_i(\ell)$ and $\Psi_i^{-1}(\ell)$ as, $\psi_i(\ell) = \ell'$ if $\forall \tau \in T_\ell, e_i(\tau) \in T_{\ell'}$, and $\Psi_i^{-1}(\ell)$ is its inverse set. We now provide our modified backpressure policy.

Back-pressure algorithm with aggregation: Below is a distributed dynamic stable 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 spare services.

Theorem 3. *Under Assumption 1 the modified backpressure algorithm achieves stability for a fully connected network for any $\lambda \in \Lambda'$.*

The proof of the above theorem is provided in Appendix D. 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.

Quantization based aggregation towards implementation: The total number of queues can be further lowered significantly by aggregating types with coarsely similar $f(\tau)$. For example, if $f(\tau)$ is the number of files seen, its range can be quantized into fewer values define a level for each value. Each node maintains a queue for each of these quantized levels. Upon arrival of a query, a node checks its $f(\tau)$ (which is embedded in the query) and appropriately puts it into a queue associated with the level closest to $f(\tau)$. It then runs the work conserving backpressure algorithm for aggregation for the general networks as described above. Since each node can decide its own levels, queries in a queue of node i may join different queues when forwarded to node j. Thus, a function of queue states can be used to compute the weights used by the algorithm.

Clearly, the above aggregation scheme may result in a reduction in capacity region, especially if the number of quantization levels is small. Thus there is an interesting tradeoff between complexity and the capacity region; we defer the analysis of such a tradeoff as a possible avenue for future work. Note that the quantization error is only in deciding the scheduling and the forwarding policy based on queues; each query-class is still accurately provided its promised GoS of γ_c. For this, each node i, before forwarding a query, updates its embedded τ to τ' by adding i to $H(\tau)$ and also updates its embedded $\phi(.)$ 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 $f(\tau)$ in a form of look-up table.

C. Alternate grades of service strategies

Till now we provided grade of service based on a priori probability $\phi(\tau)$. We provide below, in brief, some alternate strategies for providing GoS which may be more suited to some applications. Each strategy is simply a different exit policy and can be implemented by appropriately modifying vector p, as was done in Section II.

Fairness to each query: For application where each query is equally important, including the rare ones, GoS based on $\phi(\tau)$ can be unfair to queries which are rare and have lower demand ν_r. To see this, notice that in expression (2) for $\phi(\tau)$, the contribution of each each file is weighted by ν_r. Thus, files/resources with larger demand will drive $\phi(\tau)$ to higher value for a given $H(\tau)$. The actual probability of resolution for files with lower ν_r may be lower. This can be rectified by using a priori probability of resolving a 'given query' over given $H(\tau)$, given as $\phi'(\tau) = \frac{\left|\cup_{j\in H(\tau)} R_j^{c(\tau)}\right|}{|\mathcal{R}^{c(\tau)}|}$, instead of a priori probability of resolving a typical query of a 'given class', viz., $\phi(\tau)$.

Similar to p_i^τ, let ${\theta'}_i^\tau$ be the probability that a given query of class $c(\tau)$ is resolved at node i, conditioned on failing attempts by the nodes in $H(\tau)$. Formally, ${\theta'}_i^\tau = \frac{\left|R_i^{c(\tau)} \cap_{j\in H(\tau)} \{\mathcal{R}^{c(\tau)} \setminus R_j^{c(\tau)}\}\right|}{|R_i^{c(\tau)}|}$. Using ${\theta'}_i^\tau$, nodes can recursively update $\phi'(.)$, using $\phi'(\tau') = \phi'(\tau) + {\theta'}_i^\tau(1-\phi'(\tau))$, where $\tau' = (c(\tau), H(\tau)\cup\{i\})$. However, estimating ${\theta'}_i^\tau$ needs more work than p_i^τ. An unbiased estimate of p_i^τ is simply the ratio of the total number of queries of type τ resolved by node i to the number of such queries that arrived at node i, computing which does not require keeping track exactly what these queries where. Estimating ${\theta'}_i^\tau$, however, does require keeping track of this information, since it's unbiased estimate is the ratio of number of distinct queries resolved at node i, to the number of distinct queries that arrived at node i. Low complexity Bloom filters can be used to keep track of such information.

Multiple responses for each query: For certain applications, it may be beneficial to provide multiple responses to the source generating the query. For example, for applications requiring downloading a huge file, among the available options, the client may want to choose the server which is least loaded or is physically closest. For such applications, one way to provide GoS in such systems is to set exit strategy based on $\phi''(\tau)$, which is a priori expected number of responses for a query of class $c(\tau)$ from nodes in $H(\tau)$. A query of type τ exits the network if $\phi''(\tau) \geq \gamma''_{c(\tau)}$. If ${\theta''}_i^c$ is the a priori probability that a query of class $c(\tau)$ is resolved at node i, $\phi''(\tau) = \sum_{i\in H(\tau)} {\theta''}_i^c$. Further, the probability of resolution of a query used by back pressure algorithm for such a scheme, say ${p''}_i^\tau$, is simply the probability that the query of type τ exits the network. Thus, ${p''}_i^\tau = \mathbf{1}\left\{\phi''(\tau') \geq \gamma''_{c(\tau)}\right\}\mathbf{1}\{i \in H(\tau)\}$, where $\tau' = (c(\tau), H(\tau)\cup\{i\})$.

VI. Conclusion

To summarize, we provided a novel, distributed, and reliable 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 traditional random walk techniques. We also provided modifications to the algorithm that make it amenable to implementation.

Appendix

A. Proof of Theorem 1

We first prove part *(a)* of the theorem. Then, we provide Lemmas 2 and 3. Using these lemmas, we then prove part *(b)* of the theorem.

Proof of Theorem 1 part (a): Assume the system is stable under λ with a state dependent randomized policy $\pi(t)$. Let $X_{ij}^\tau(t)$ be total number of queries of type τ transmitted on link (i,j) up to time t under $\pi(t)$. Let $X_{ij}^{s,\tau}(t)$ be the number of such queries that where resolved at node j, and where thus removed from the system. Further, let $A_i^{s,c}(t)$ be exogenous arrivals of class c at node i that were successfully resolved at node i at time t. Thus the following holds for all time, and all $i \in \mathcal{N}$.

$$\sum_{j\in N(i),\tau\in\mathcal{T}} X_{ji}^\tau(t) + \sum_c \sum_{t'=1}^{t} A_i^c(t') \leq t\mu_i + \sum_{\tau\in\mathcal{T}} Q_i^\tau(t+1). \tag{10}$$

Also, for all $i \in \mathcal{N}$ and $\tau \in \mathcal{T}$,

$$\sum_{j\in N(i),\tau'\in E_i^{-1}(\tau)} X_{ji}^{\tau'}(t) + \sum_{t'=1}^{t} A_i^{c(\tau)}(t')\mathbf{1}\{H(\tau') = \emptyset\} = Q_i^\tau(t+1)$$
$$+ \sum_{j\in N(i),\tau'\in E_i^{-1}(\tau)} X_{ji}^{s,\tau'}(t) + \sum_{t'=1}^{t} A_i^{s,c(\tau)}(t')\mathbf{1}\{H(\tau') = \emptyset\} + \sum_j X_{ij}^\tau(t). \tag{11}$$

Also, for all $(i,j) \in \mathcal{L}$ and for all $\tau \in \mathcal{T}$,

$$X_{ij}^\tau(t) \geq 0 \tag{12}$$

Here, (10) follows because total query arrivals at any node is equal to number of the queries waiting in the queue plus number of those already serviced. (11) follows because total queries resolved at any node should be less than or equal total service provided. We now use the following lemma from [14], to show that above equations imply that a solution to constraints (4)-(6) exists.

Lemma 1. *If the network is stable, then there exists a finite value α such that the event $Q_i^\tau(t) \leq \alpha\ \forall i, \tau$ occurs infinitely often.*

Thus, there exist some α such that $Q_i^\tau(t) \leq \alpha$ at arbitrarily large times. Thus, with high probability, one can find a large enough time $\tilde{t}$ such that, for some $\epsilon > 0$, and for all $i \in \mathcal{N}$, $c \in \mathcal{C}$ and $\tau \in \mathcal{T}$, we get

$$Q_i^\tau(t) \leq \alpha \tag{13}$$

$$\frac{\alpha}{\tilde{t}} \leq \epsilon \tag{14}$$

$$\left|\sum_{t'=1}^{\tilde{t}} \frac{A_i^{c(\tau)}(t')}{\tilde{t}} - \lambda_i^{c(\tau)}\right| \leq \epsilon \tag{15}$$

$$\left|\frac{\sum_{t'=1}^{\tilde{t}} A_i^{s,c(\tau)}(t')}{\tilde{t}} - p_i^\tau \frac{\sum_{t'=1}^{\tilde{t}} A_i^{c(\tau)}(t') - Q_i^{(c(\tau),\emptyset)}(t+1)}{\tilde{t}}\right| \leq \epsilon \tag{16}$$

$$\left|\frac{\sum_j X_{ij}^{s,\tau}(\tilde{t})}{\tilde{t}} - p_i^\tau \frac{\sum_j X_{ij}^\tau(\tilde{t}) - Q_i^\tau(\tilde{t}+1)}{\tilde{t}}\right| \leq \epsilon \tag{17}$$

where (16) and (17) follow from definition of p_i^τ and the law of large numbers. Then, setting $f_{ij}^\tau(t) = X_{ij}^\tau(t)/t$

and using (13)-(17) with triangle inequality one can show that resulting variables $f(t)$ are arbitrarily close to satisfying constraints (4)-(6) at time $t = \tilde{t}$. Thus, input rates λ are arbitrarily close to Λ. It is not difficult to show that Λ is compact and thus contains its limit points. Thus, $\lambda \in \Lambda$. $\square$

Lemma 2. *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,

$$\mathbf{E}\left[F_{ij}^\tau(t)\right] \le \sum_{\tau' \in E_i^{-1}(\tau)} \mu_{ij}^{\tau'}(t)(1-p_i^{\tau'}). \tag{18}$$

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

$$Q_i^\tau(t+1) = \max(Q_i^\tau(t) - \mu_i^\tau(t), 0) + \sum_{j\in N(i)} F_{ji}^\tau(t) + A_i^{c(\tau)}(t)\mathbf{1}\{H(\tau)=\emptyset\}. \tag{19}$$

It can be easily checked that the above implies,

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

Summing over all τ and i, we get

$$\sum_{\tau,i}\left((Q_i^\tau(t+1))^2 - (Q_i^\tau(t))^2\right) \le B'(t) - 2\sum_{\tau,i} Q_i^\tau(t)$$
$$\times\left(\mu_i^\tau(t) - \sum_{j\in N(i)} F_{ji}^\tau(t) - A_i^{c(\tau)}(t)\mathbf{1}\{H(\tau)=\emptyset\}\right) \tag{20}$$

where

$$B'(t) = \sum_{\tau,i}(\mu_i^\tau(t))^2 + \left(\sum_{j\in N(i)} F_{ji}^\tau(t)\right)^2 + \sum_{\tau,i}\left(A_i^{c(\tau)}(t)\mathbf{1}\{H(\tau)=\emptyset\}\right)^2 \tag{21}$$

since $\sum_{j\in N(i)} F_{ji}^\tau(t) A_i^{c(\tau)}(t)\mathbf{1}\{H(\tau)=\emptyset\} = 0$ as $F_{ji}^\tau(t)\mathbf{1}\{H(\tau)=\emptyset\} = 0$. Here, $\sum_{\tau,i}(\mu_i^\tau(t))^2 \le \sum_i(\mu_i)^2$. Also, $\sum_{\tau,i}(\sum_{j\in N(i)} F_{ji}^\tau(t))^2 \le \left(\sum_{\tau,i}\sum_{j\in N(i),\tau'\in E_j^{-1}(\tau)} \mu_{ji}^{\tau'}(t)\right)^2 \le \sum_i(\mu_i)^2$. Further, $\sum_{\tau,i}\left(A_i^{c(\tau)}(t)\mathbf{1}\{H(\tau)=\emptyset\}\right)^2 = \sum_{c,i}(A_i^\tau(t))^2$. Thus, $\mathbf{E}\left[B'(t)\right] \le 2\sum_i(\mu_i)^2 + \sum_{c,i}\mathbf{E}\left[(A_i^\tau(t))^2\right] \triangleq B$. Thus, by talking expectation on both sides of (20), we get,

$$\sum_{\tau,i}\mathbf{E}\left[(Q_i^\tau(t+1))^2 - (Q_i^\tau(t))^2\right] \le B - 2\sum_{\tau,i} Q_i^\tau(t)$$
$$\times\mathbf{E}\left[\mu_i^\tau(t) - \sum_{j\in N(i)} F_{ji}^\tau(t) - A_i^{c(\tau)}(t)\mathbf{1}\{H(\tau)=\emptyset\}\right] \tag{22}$$

from which the lemma follows by using (18). $\square$

Lemma 3. *Given $\lambda \in \Lambda'$, one can obtain a fixed valid assignment $\mu(t) = (\tilde{\mu}_{ij}^\tau)$ (and correspondingly, $\mu_i^\tau(t) = \tilde{\mu}_i^\tau$) such that, for some $\epsilon_i^\tau > 0$,*

$$\tilde{\mu}_i^\tau - \sum_{j,\tau'\in E_j^{-1}(\tau)} \tilde{\mu}_{ji}^{\tau'}(1-p_j^{\tau'}) - \lambda_i^{c(\tau)}\mathbf{1}\{H(\tau)=\emptyset\} = \epsilon_i^\tau \ \forall i,\tau$$

Proof. The definition of Λ allows for exogenous arrivals $\lambda_i^{c(\tau)}$ 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 $\tilde{\lambda} = (\tilde{\lambda}_i^\tau : i \in \mathcal{N}, \tau \in \mathcal{T})$.

Definition 3. *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 $\tilde{f} \triangleq (\tilde{f}_{ij}^\tau : ij \in \mathcal{L}, \tau \in \mathcal{T})$:*

1) *Capacity constraints: for all $i \in \mathcal{N}$,*
$$\sum_{j,\tau} \tilde{f}_{ji}^\tau + \sum_\tau \tilde{\lambda}_i^\tau \le \tilde{\mu}_i \tag{23}$$
2) *Flow conservation constraints with resolution at nodes: For all $i \in \mathcal{N}$ and $\tau \in$ types,*
$$\sum_{\tau'\in E_i^{-1}(\tau)} (1-p_i^{\tau'})\left(\sum_j \tilde{f}_{ji}^{\tau'} + \tilde{\lambda}_i^{\tau'}\right) = \sum_j \tilde{f}_{ij}^\tau \tag{24}$$
3) *Non-negativity constrants: for all $(i,j) \in \mathcal{L}$ and $\tau \in \mathcal{T}$,*
$$\tilde{f}_{ij}^\tau \ge 0. \tag{25}$$

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

Proof of Properties of $\tilde{\Lambda}$: a) follows from definition of Λ, since putting $\tilde{\lambda}_i^\tau = \lambda_i^{c(\tau)}\mathbf{1}\{H(\tau)=\emptyset\}$ 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 $\tilde{\lambda} = (\lambda_i^{c(\tau)}\mathbf{1}\{H(\tau)=\emptyset\})$. Thus, by convexity of $\tilde{\Lambda}$ and property c), $\tilde{\lambda} + \epsilon'\sum_{i,\tau}\gamma_i^\tau\bar{\delta}_i^\tau \in \tilde{\Lambda}$, where $\epsilon' = \frac{1}{|\mathcal{N}||\mathcal{C}|}(1-\frac{1}{1+\epsilon})$. Putting $\epsilon_i^\tau = \epsilon'\gamma_i^\tau$, we get $\tilde{\lambda} + \bar{\epsilon} \in \tilde{\Lambda}$, where $\bar{\epsilon} = (\epsilon_i^\tau)$.

Now, obtain a feasible solution $\tilde{f}' = (\tilde{f}'^{\tau}_{ij}, ij \in \mathcal{L}, \tau \in \mathcal{T})$ for constraints (23)-(25) for general arrivals $\tilde{\lambda} + \bar{\epsilon}$, where $\tilde{\lambda} = (\lambda_i^{c(\tau)} \mathbf{1}\{H(\tau) = \emptyset\})$. Using this solution, set for all i and τ

$$\tilde{\mu}_i^{\tau} = \sum_j \tilde{f}'^{\tau}_{ji} + \lambda_i^{c(\tau)} \mathbf{1}\{H(\tau) = \emptyset\} + \epsilon_i^{\tau}, \tag{26}$$

and

$$\tilde{\mu}_{ij}^{\tau} = \mu_i^{\tau} \frac{\tilde{f}'^{e_i(\tau)}_{ij}}{\sum_{j'} \tilde{f}'^{e_i(\tau)}_{ij'}}. \tag{27}$$

Note that, this assignment of $\tilde{\mu}_{ij}^{\tau}$ (and corresponding π_{ij}^{τ}) is valid, since from constraint (23) we get $\sum_{j,\tau} \mu_{ij}^{\tau} \leq \mu_i$ for all i. Using these assignments and constraint (24), and one can check that,

$$\sum_{\tau' \in E_i^{-1}(\tau)} (1 - p_i^{\tau'}) \tilde{\mu}_{ij}^{\tau'} = \tilde{f}'^{\tau}_{ij} \; \forall i, \tau. \tag{28}$$

Putting this in (26), we get

$$\sum_j \sum_{\tau' \in E_j^{-1}(\tau)} \tilde{\mu}_{ji}^{\tau'} (1 - p_j^{\tau'}) + \lambda_i^{c(\tau)} \mathbf{1}\{H(\tau) = \emptyset\} + \epsilon_i^{\tau} = \tilde{\mu}_i^{\tau} \; \forall i, \tau \tag{29}$$

□

Proof of Theorem 1 part (b): Under a fixed randomized policy, $Q(t)$ forms a Markov chain. Consider candidate Lyapunov function $L(Q) = \sum_{i,\tau} (Q_i^{\tau})^2$. Thus, if we show that drift $\Delta Q(t) \triangleq \mathbf{E}\left[\sum_{\tau,i} (Q_i^{\tau}(t+1))^2 - (Q_i^{\tau}(t))^2 \middle| Q(t)\right]$ 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 2 provides an upper-bound on $\Delta Q(t)$. Substituting the result of Lemma 3 in this bound, we get

$$\Delta Q(t) \leq B - 2 \sum_{\tau,i} Q_i^{\tau}(t) \epsilon_i^{\tau} \tag{30}$$

where B is a constant, and $\epsilon_i^{\tau} > 0, \forall i, \tau$. Thus, for the randomized policy given in Lemma 3, 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 4. We then use Lemmas 2 and 4 to prove the theorem.

Lemma 4. *For the given back pressure algorithm, if λ is in the interior of Λ, then, for some $\bar{\epsilon} > 0$,*

$$\sum_{\tau,i} Q_i^{\tau}(t) \left(\mu_i^{*\tau}(t) - \sum_{j,\tau' \in E_j^{-1}(\tau)} \mu_{ji}^{*\tau'}(t)(1 - p_j^{\tau'}) \right) \geq \sum_{\tau,i} Q_i^{\tau}(t) \left(\lambda_i^{c(\tau)} \mathbf{1}\{H(\tau) = \emptyset\} + \epsilon_i^{\tau} \right) \tag{31}$$

Proof. From Lemma 3, there exist a stationary static policy, that does not depend on $Q(t)$, and determines valid fixed service rates $\tilde{\mu}_{ij}^{\tau}$ such that

$$\sum_{\tau,i} Q_i^{\tau}(t) \left(\tilde{\mu}_i^{\tau} - \sum_{j,\tau' \in E_j^{-1}(\tau)} \tilde{\mu}_{ji}^{\tau'} (1 - p_j^{\tau'}) \right) = \sum_{\tau,i} Q_i^{\tau}(t) \left(\lambda_i^{c(\tau)} \mathbf{1}\{H(\tau) = \emptyset\} + \epsilon_i^{\tau} \right), \tag{32}$$

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

$$\begin{aligned} &\sum_{\tau,i} Q_i^{\tau}(t) \left(\sum_j \tilde{\mu}_{ij}^{\tau} - \sum_{j,\tau' \in E_j^{-1}(\tau)} \tilde{\mu}_{ji}^{\tau'} (1 - p_j^{\tau'}) \right) \\ &\quad = \sum_{(i,j) \in \mathcal{L}, \tau} \tilde{\mu}_{ij}^{\tau} \left(Q_i^{\tau}(t) - Q_j^{e_i(\tau)}(t)(1 - p_i^{\tau}) \right) \\ &\quad \leq \sum_{(i,j) \in \mathcal{L}, \tau} \tilde{\mu}_{ij}^{\tau} w_{ij}^{*}(t) \leq \sum_{(i,j) \in \mathcal{L}, \tau} \mu_{ij}^{*\tau}(t) w_{ij}^{*}(t), \end{aligned} \tag{33}$$

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,

$$\begin{aligned} &\sum_{(i,j) \in \mathcal{L}, \tau} \mu_{ij}^{*\tau}(t) w_{ij}^{*}(t) \\ &\quad = \sum_{(i,j) \in \mathcal{L}, \tau} \mu_{ij}^{*\tau}(t) \left(Q_i^{\tau}(t) - Q_j^{e_i(\tau)}(t)(1 - p_i^{\tau}) \right) \\ &\quad = \sum_{\tau,i} Q_i^{\tau}(t) \left(\mu_i^{*\tau}(t) - \sum_{j,\tau' \in E_j^{-1}(\tau)} \mu_{ji}^{*\tau'}(t)(1 - p_j^{\tau'}) \right). \end{aligned} \tag{34}$$

From (32),(33) and (34), the lemma follows. □

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

$$\mathbf{E}\left[\sum_{\tau,i} (Q_i^{\tau}(t+1))^2 - (Q_i^{\tau}(t))^2 \middle| Q(t) \right] \leq B - 2 \sum_{\tau,i} Q_i^{\tau}(t) \times \left(\mu_i^{*\tau}(t) - \sum_{j,\tau' \in E_j^{-1}(\tau)} \mu_{ji}^{*\tau'}(t)(1 - p_j^{\tau'}) - \lambda_i^{c(\tau)} \mathbf{1}\{H(\tau) = \emptyset\} \right) \tag{35}$$

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 4 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 4 and (35), 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 $Q(t)$.

D. Proof of Theorem 3

The proof is along the lines similar to that of Theorem 2. First, we show that there exist a 'modified randomized policy' that depends on $Q'(t)$ and achieves stability, and then show that the modified backpressure policy can only do better.

State dependent modified randomized policy: Given that $Q'(t) = q'(t)$, under a modified randomized policy, each node i does the following for each of the μ_i services, in each slot:

1) It randomly chooses level ℓ and node j with probability ${\pi'}_{ij}^{\ell}(t)$.
2) Node i serves a query of level ℓ which has not visited node j on FCFS basis. If unresolved, it forwards it to node j.
3) If no such query is waiting, it serves a blank query.

For ease of notation, define $\mu'(t) \triangleq ({\mu'}_{ij}^{\ell}(t) : (i,j) \in \mathcal{L}, \ell \in \Gamma)$, where, ${\mu'}_{ij}^{\ell}(t) \triangleq \mu_i {\pi'}_{ij}^{\ell}(t)$. Thus, determining ${\pi'}_{ij}^{\ell}(t)$ is equivalent to determining ${\mu'}_{ij}^{\ell}(t)$. Further, define ${\mu'}_{i}^{\ell}(t) \triangleq \sum_j {\mu'}_{ij}^{\ell}(t)\ \forall i, \ell$. Note that $\mu'(t)$ may be function of $q(t)$.

Consider constraints (23)-(25). We showed in the proof of Lemma 3 that a solution to these constraints, say $\tilde{f}' = (\tilde{f'}_{ij}^{\tau}, (i,j) \in \mathcal{L}, \tau \in \mathcal{T})$, exists for any $\lambda \in \Lambda'$. For the case of complete graph, we now show that, for each $\lambda \in \Lambda'$ we can modify $\tilde{f}'$ such that $\tilde{f'}_{ij}^{\tau} = 0$ for all τ such that $j \in H(\tau)$. This would basically imply that one can obtain a stable randomized policy for complete graph that avoids revisits by queries. Consider any (i,j) and τ such that $\tilde{f'}_{ij}^{\tau} > 0$ and $j \in H(\tau)$. Note that this also implies that $p_j^\tau = 0$ and $p_i^\tau = 0$, since now i is also visited. Assume without loss of generality, $\tilde{f'}_{ij}^{\tau} \geq \tilde{f'}_{ji}^{\tau}$. (Else, we flip i and j). Now, set $\tilde{f'}_{ij}^{\tau} = \tilde{f'}_{ij}^{\tau} - \tilde{f'}_{ji}^{\tau}$ and then $\tilde{f'}_{ji}^{\tau} = 0$. One can check that $\tilde{f}'$ still satisfies the constraints. Now, for all $j' \neq i, j$, set $\tilde{f'}_{ij'}^{\tau} = \tilde{f'}_{ij'}^{\tau} + \tilde{f'}_{jj'}^{\tau}$ and then set $\tilde{f'}_{jj'}^{\tau} = 0$ and $\tilde{f'}_{ij}^{\tau} = 0$. One can again check that $\tilde{f}'$ still satisfies the constraints. Repeat this process for all such pairs of (i,j) and τ. We have thus obtain a solution $\tilde{f}'$ such that $\tilde{f'}_{ij}^{\tau} = 0$ for all τ such that $j \in H(\tau)$.

Now, just as in Lemma 3, use the modified $\tilde{f}'$ in (26) and (27) to obtain $\mu(t) = (\tilde{\mu}_{ij}^\tau)$ such that, for some $\epsilon_i^\tau > 0$,

$$\tilde{\mu}_i^\tau - \sum_{j,\tau' \in E_j^{-1}(\tau)} \tilde{\mu}_{ji}^{\tau'}(1 - p_j^{\tau'}) - \lambda_i^{c(\tau)} \mathbf{1}\{H(\tau) = \emptyset\} = \epsilon_i^\tau\ \forall i, \tau \tag{36}$$

Now, obtain fixed valid assignment $\mu'(t) = (\tilde{\mu'}_{ij}^{\ell})$ for the modified randomized policy by setting $\tilde{\mu'}_{ij}^{\ell} = \sum_{\tau \in T_\ell} \tilde{\mu}_{ij}^\tau$. For this fixed modified randomized policy, we obtain the following identity by summing both sides of (36) over all $\tau \in T_\ell$ for each ℓ. For some $\epsilon_i^\ell > 0$, and for all $i \in \mathcal{N}$ and $\ell \in \Gamma$,

$$\tilde{\mu'}_i^{\ell} - \sum_{j,\ell' \in \Psi_j^{-1}(\ell)} \tilde{\mu'}_{ji}^{\ell'}(1 - {p'}_j^{\ell'}) - \lambda_i^{(\ell)} = \epsilon_i^\ell \tag{37}$$

where $\lambda_i^{(\ell)} = \lambda_i^{c(\tau)}$ for $\ell \ni \tau$ such that $H(\tau) = \emptyset$. Note that our definition of ℓ provides a seperate level for each such τ.

Now, proceed along the lines of Lemma 2 to obtain the following, for any modified randomized policy:

$$\mathbf{E}\left[\sum_{\ell,i} ({Q'}_i^{\ell}(t+1))^2 - ({Q'}_i^{\ell}(t))^2 \Big| Q'(t)\right] \leq B - 2\sum_{\ell,i} {Q'}_i^{\ell}(t) \times \left({\mu'}_i^{\ell}(t) - \sum_{j,\ell' \in \Psi_j^{-1}(\ell)} {\mu'}_{ji}^{\ell'}(t)(1 - {p'}_j^{\ell'}) - \lambda_i^{(\ell)}\right), \tag{38}$$

for some constant B.

Note that even under a fixed modified randomized policy, $Q(t)$ forms a Markov chain (but not $Q'(t)$). Consider candidate Lyapunov function $L'(Q) = \sum_{i,\ell}\left(\sum_{\tau \in T_\ell} Q_i^\tau\right)^2 = \sum_{i,\ell}\left({Q'}_i^{\ell}\right)^2$. Thus, if we show that the drift $\mathbf{E}\left[\sum_{\ell,i}({Q'}_i^{\ell}(t+1))^2 - ({Q'}_i^{\ell}(t))^2 \big| Q'(t)\right]$ is negative for all but finite set values of $Q'(t)$, and thus $Q(t)$, it would imply that $L'(Q)$ is a Lyapunov function, thus proving that Markov chain is positive recurrent, from which stability follows. This is evident by substituting (37) in (38).

Now, observe that the modified backpressure policy is equivalent to a modified randomized policy with ${\mu'}_{ij}^{\ell} = \mu_i$ for $j = j_i^*$ and $\ell = \ell_i^*$ and 0 otherwise. Now, follow steps along the lines of proof of Lemma 4 to show that the drift of the modified backpressure policy is only more negative to above fixed modified randomized policy, thus proving its stability.

References

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

[2] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Transactions on 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, Boca Raton, USA), 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*, (Annapolis, MD), June 2008.

[7] P. Patankar, G. Nam, G. Kesidis, T. Konstantopoulos, and C. Das, "Peer-to-peer unstructured anycasting using correlated swarms," in *Proc. ITC*, (Paris), Sept 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*, (Bangalore, India), December 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, A. K. Dubey, S. Ghose, and N. Ganguly, "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 Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2004.

[12] B. Yang and H. Garcia-Molina, “Designing a super-peer network,” in *Proc. IEEE ICDE*, 2003.
[13] L. Tassiulas and A. Ephremides, “Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks,” *IEEE Transactions on Automatic Control*, vol. 37, pp. 1936–1948, 1992.
[14] M. J. Neely, E. Modiano, and C. E. Rohrs, “Dynamic power allocation and routing for time varying wireless networks,” in *Proc. IEEE INFOCOM*, 2003.
[15] L. Ying, S. Shakkottai, A. Reddy, and S. Liu, “On combining shortest-path and back-pressure routing over multihop wireless networks,” *IEEE/ACM Transactions on Networking*, vol. 19, pp. 841–854, June 2011.
[16] M. Alresaini, M. Sathiamoorthy, B. Krishnamachari, and M. Neely, “Backpressure with adaptive redundancy (bwar),” in *Proc. IEEE INFOCOM*, pp. 2300–2308, March 2012.
[17] L. Bui, R. Srikant, and A. Stolyar, “A novel architecture for reduction of delay and queueing structure complexity in the back-pressure algorithm,” *IEEE/ACM Transactions on Networking*, vol. 19, pp. 1597–1609, Dec 2011.
[18] B. Ji, C. Joo, and N. Shroff, “Delay-based back-pressure scheduling in multihop wireless networks,” *IEEE/ACM Transactions on Networking*, vol. 21, pp. 1539–1552, Oct 2013.
[19] Y. Cui, E. Yeh, and R. Liu, “Enhancing the delay performance of dynamic backpressure algorithms,” *IEEE/ACM Transactions on Networking*, 2015.
[20] L. Georgiadis, M. J. Neely, and L. Tassiulas, “Resource allocation and cross-layer control in wireless networks,” *Foundations and Trends in Networking*, vol. 1, no. 1, pp. 1–144, 2006.
[21] Y. Cui, V. Lau, R. Wang, H. Huang, and S. Zhang, “A survey on delay-aware resource control for wireless systems – large deviation theory, stochastic lyapunov drift, and distributed stochastic learning,” *IEEE Transactions on Information Theory*, vol. 58, pp. 1677–1701, March 2012.
[22] S. Asmussen, *Applied probability and queues*. Springer, 1987.
[23] H. J. Kushner and G. Yin, *Stochastic approximation and recursive algorithms and applications*. Springer, 2003.

Virag Shah received his Ph.D. at Electrical and Computer Engineering department at The University of Texas at Austin. He received his B.E. degree from University of Mumbai in 2007. He received his M.E. degree from Indian Institute of Science, Bangalore in 2009.

He is currently a Simons Postdoc at UT Austin. He was a Research Fellow at Indian Institute of Technology, Bombay from 2009 to 2010. His research interests include designing algorithms for content delivery systems, cloud computing systems, and internet of things; performance modeling; applied probability and queuing theory.

He is a recipient of two best paper awards: IEEE INFOCOM 2014 at Toronto, Canada; National Conference on Communications 2010 at Chennai, India.

Gustavo de Veciana (S’88-M’94-SM’01-F’09) received his B.S., M.S, and Ph.D. in electrical engineering from the University of California at Berkeley in 1987, 1990, and 1993 respectively.

He joined the Department of Electrical and Computer Engineering where he is currently a Cullen Trust Professor of Engineering. He served as the Director and Associate Director of the Wireless Networking and Communications Group (WNCG) at the University of Texas at Austin, from 2003-2007. His research focuses on the analysis and design of communication and computing networks; data-driven decision-making in man-machine systems, and applied probability and queueing theory. Dr. de Veciana served as editor and is currently serving as editor-at-large for the IEEE/ACM Transactions on Networking.

He was the recipient of a National Science Foundation CAREER Award 1996 and a co-recipient of five best paper awards including: IEEE William McCalla Best ICCAD Paper Award for 2000, Best Paper in ACM TODAES Jan 2002-2004, Best Paper in ITC 2010, Best Paper in ACM MSWIM 2010, and Best Paper IEEE INFOCOM 2014. In 2009 he was designated IEEE Fellow for his contributions to the analysis and design of communication networks. He currently serves on the board of trustees of IMDEA Networks Madrid.

George Kesidis received his MS (in 1990) and PhD (in 1992) from UC Berkeley in EECS. Following eight years as a professor of ECE at the University of Waterloo, he has been a professor of EECS at the Pennsylvania State University since 2000. His research interests include many aspects of networking, cyber security and machine learning, particularly intrusion detection based on large-scale network datasets, and, more recently, energy efficiency and the impact of economic policy. His work has been supported by NSF and DARPA research grants and Cisco Systems URP gifts.