Towards Topology Aware Networks

Christos Gkantsidis
Microsoft Research
Cambridge, UK
email: chrisgk@microsoft.com

Gagan Goel, Milena Mihail
College of Computing
Georgia Institute of Technology
Atlanta, GA, USA
Email: {gagang,mihail}@cc.gatech.edu

Amin Saberi
Management Science
and Engineering
Stanford University
Palo Alto, CA, USA
email: saberi@stanford.edu

***Abstract*— We focus on efficient protocols that enhance a network with topology awareness. We discuss centralized algorithms with provable performance, and introduce decentralized asynchronous heuristics that use only local information and local computations. These algorithms are based on distributed solutions of convex programs expressing optimization of various spectral properties of the matrix associated with the graph of the network topology. For example, these algorithms assign special weights to links crossing or directed towards small cuts by minimizing the second eigenvalue. Our main technical ingredient is to perform the decentralized asynchronous computations is a manner that preserves critical invariants of the exact second eigenvalue of the adjacency matrix associated with the network topology. To further demonstrate the utility of our algorithms we show how their output can enhance the performance in the context of peer-to-peer networks.**

I. Introduction

Overlay networks is the main design strategy by which we try to evolve the Internet, adapt it to changing requirements, and accommodate emerging technologies. Several widely used file sharing and content distribution applications involve unstructured P2P networks.

Clearly, an overlay network should maintain good topology connectivity, both for the purposes of providing good quality of service at the virtual overlay network layer, as well as for the purpose of imposing a low and balanced load at the underlying network layer. We believe that critical to both these goals is that the overlay network protocols are aware of some information concerning the topology, both of the virtual overlay network, as well as special features of the underlying layer.

In this paper we focus on the questions of maintaining good topology connectivity and topology awareness in unstructured P2P networks. In particular, we focus on methods that:
(1) Identify critical network links, such as links across a sparse network cut. See Figures 1 and 2. In very rough terms, we wish for network links to become aware of their criticality. We base our heuristics for identifying critical network links in an asynchronous distributed computation. This computation aims to "mimic" the computation of the second eigenvalue of the adjacency matrix of the graph representing the network topology. The key technical ingredient is that each local computation maintains a vector over the entire network that is perpendicular to the first eigenvector, thus tends to converge to a vector close to the second eigenvector.
(2) Improve the performance of network protocols by taking advantage of information about link criticality.
(3) Improve the connectivity of the network, for example by rewiring according to efficient and reliable heuristics.

Towards the above goal we present two (related) classes of algorithms. In Section II we present convex optimization programs which assign weights to links that optimize the performance of protocols running on the underlying network topologies. These novel algorithms need centralized control, so it is not readily applicable in today's decentralized environments. In addition, even though the convex programs imply that the corresponding problems are in P, in practice, solutions can be obtained for graphs with a few tens of nodes. Therefore, this initial theoretical approach is currently impractical. Obtaining a decentralized approximation to this algorithm is a fundamental open problem.

In Section III we present a fully decentralized heuristic to identify critical links. This heuristic is based on distributed computation of the second eigenvalue of a suitable normalization of the adjacency matrix of the network topology. The heuristic is that links whose endpoints are assigned substantially different weights are "critical" links. (Indeed, this heuristic can be viewed as the first step of the formal algorithm of Section II.) In Section IV we discuss how the spectral method can be applied in an asynchronous way.

In Section V we give examples of how awareness of link criticality improves protocol performance. The example that we use is in the sense of realistic hybrid search schemes which extend the usual method of flooding. In Section VI we show how link awareness can improve the configuration of the network topology

using rewiring methods. In Section VII we show how link awareness can improve routing congestion.

II. Optimization Formalization and Results

To the best of our knowledge, the first effort to enhance a graph with topology awareness is due to [1]. They started with a connected graph $G(V,E)$ and a symmetric Markov chain defined on this graph. That is, a matrix P of transition probabilities on the links E of the graph, such that $p_{ij}=p_{ji}$, for all nodes i and j. They wished to assign weights on the links so as to preserve the limiting stationary distribution of the Markov chain, but speed up the convergence rate of the corresponding Markov chain as much as possible. (For our purposes, that is the construction of well connected network topologies, it is clear that a graph on which the corresponding Markov chain has optimal convergence must have good conductance and hence good connectivity). [1] showed that their problem is in P by writing it as the following semidefinite program:

$$\begin{array}{ccc} & \min s & \\ -sI \leq & P-\frac{1}{n}11^T & \leq sI \\ P \geq 0 & P1=1 & P=P^T \\ & p_{ij}=0 & \{i,j\} \notin E \end{array}$$

For more practical purposes, [1] proposed an iterative sugradient algorithm, which can be sketched as follows. Let $\vec{p}$ be some initial vector on E of transition probabilities. Let $\vec{e}$ be the eigenvector corresponding to the second in absolute value eigenvalue of P. Realize that $\vec{e}$ assigns one value e_i to each node $i \in V$. Let $g^{(0)}$ be a vector on E (i.e. assignments of weights on the links of E, with

$$g^0_{ij}=(e_i-e_j)^2$$

Then, iteratively, [1] repeated a subgradient step $\vec{p}=\vec{p}-\alpha_k g^{(k)}/||g^{(k)}||_2$, where α_k was determined heuristically. Of course, this is a centralized algorithm and impractical to implement in a network with thousands of nodes. For our purposes, what we maintain from the above formalization, is the significance of the last equation, that is, considering the difference in value that the second eigenvector gives to a link. At least from the first step of this iterative algorithm, it appears that critical links will be links for which $|e_i-e_j|$ will be large.

In another effort, [2] wrote a semidefinite program for assigning weight in a graph that minimize the effective resistance and the average commute time between all pairs of nodes. Thus this problem is also in P. Their formulation is

$$\begin{array}{ccc} & \min nTrY & \\ & s.t. \quad 1^T g=1 & g \geq 0 \\ \left[\begin{array}{cc} G+11^T/n & I \\ I & Y \end{array}\right] & \geq & 0 \end{array}$$

They also gave a subgradient algorithm whose iterative steps involve the difference $|e_i-e_j|$ given by the second eigenvector and, in particular, averages of $\frac{e_i+e_j}{2}$.

The significance of assigning weights so as to minimize the average commute time in networking is that it expresses the following scenario. Suppose that a random node initiates a query that is located in another random node, and performs the search for that other random node by random walk. Then the average search time is precisely the average commute time.

Like the algorithm of [1], the algorithm of [2] is centralized and can be applied, in practice, to graphs with few ten of nodes. However, the results are very interesting. For demonstration, in Figures II and II, we give the relative weights that the algorithm of [2] gave in two graphs with clear critical cuts. It is obvious that the algorithm identified these cuts with remarkable accuracy.

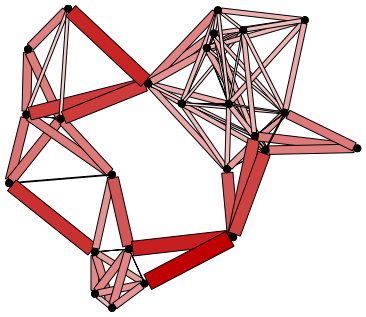

Fig. 1: In this random graph, weights have been assigned according to the algorithm of [2]. The thickness of links is proportional to the assigned weights. We clearly see that links appearing with larger weights are links crossing sparse cuts.

In the rest of this paper, we shall try to convert a few aspects of these exact algorithms to heuristics that identify critical links, but are at the same time practical for use in real distributed and asynchronous networks.

III. Spectral Algorithms

In this section we review the spectral method for clustering and identification of link awareness and criticality. As we said in Section II, we are, in some sense, trying to implement the first step of the subgradient algorithm of [1]. In addition, the spectral method has been well

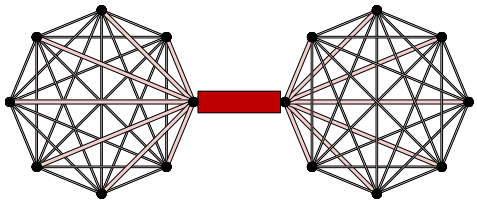

Fig. 2: In this bellbar graph, weights have been assigned according to the algorithm of [2]. Again, the thickness of the links is proportional to the assigned weights. We clearly see that the most critical link connecting the two clusters has the largest weight. We also see that links connected to this most critical link, thus leading to the sparsest cut also have larger weights.

recognized for clustering and link criticality, e.g. in [3]–[5].

We note that, even though the spectral method is well known, the particular way that we break the action of this method, expressed in equations (3) and (4) is new here. This is precisely the adaptation that we needed to later make the spectral method distributed and asynchronous.

Let $G(V, E)$ be an undirected simple graph representing the topology of a network. Let $|V|=n$ and $|E|=m$, as usual. We typically deal with sparse networks, i.e. $|E| = O(|V|)$ or $n = O(m)$. Let A be the adjacency matrix of G, that is, A is an $n \times n$ 0-1 matrix, with $a_{ij} = 1$ if $\{i, j\} \in E$ and $a_{ij} = 0$ if $\{i, j\} \notin E$. Let P be the stochastic normalization of A, i.e. P is a stochastic matrix representing the lazy random walk on G, where a particle moving on the node of the graph G determines its next move by staying, with probability 1/2 in its current position, while with probability 1/2 the particle moves to one of the neighboring nodes chosen uniformly at random. In particular, if d_i is the degree of node i, $1 \leq i \leq n$, then P is an $n \times n$ stochastic matrix, with $p_{ii} = 1/2$, and, for $i \neq j$, $p_{ij} = 1/2d_i$ if $\{i, j\} \notin E$. It is well known that P has n real eigenvalues, $1 = \lambda_1 \geq \lambda_2 \geq \ldots \lambda_n \geq 0$, where $\lambda_2 < 1$ if the graph is connected, and with n corresponding real eigenvectors $\vec{e_1}, \ldots \vec{e_n}$. That means that $\vec{e_i} = \lambda_i \vec{e_i}$. In particular, each eigenvector assigns a real value to each node of the graph. By e_{ij} we mean the value that eigenvector $\vec{e_i}$ assigns to node j.

When $G(V, E)$ is a regular graph, i.e. all the degrees are equal, then the following properties hold: (a)The first eigenvector, corresponding to eigenvalue 1, assigns the same value $1/n$ to all the nodes of the graph (this is the stationary distribution of the lazy random walk on G). (b)The eigenvectors $\vec{e_i}$, $1 \leq i \leq n$, are all orthogonal to each other and span the n dimensional space. In particular, all eigenvectors $\vec{e_2}, \ldots, \vec{e_n}$ are vertical to the first eigenvector $\vec{e_1}$, so all corresponding inner products $< \vec{e_1}, \vec{e_i} >= 0$. In addition, assuming that we pick any vector $\vec{x}$ vertical to $\vec{e_1}$, that is $< \vec{e_1}, \vec{x} >= 0$, or $\sum_{i=1}^{n} e_{1i} x_i = 0$, then, if we apply P repeatedly to $\vec{x}$, we get

$$\vec{x} = \alpha_2 \vec{e_2} + \ldots + \alpha_n \vec{e_n}$$

therefore

$$\vec{x} P^k = \alpha_2 \lambda_2^k \vec{e_2} + \alpha_3 \lambda_3^k \vec{e_3} + \ldots + \alpha_n \lambda_n^k \vec{e_n} \simeq \alpha_2 \lambda_2^k \vec{e_2}. \quad (1)$$

assuming that $1 > \lambda_2 > \lambda_3 \geq \ldots \geq \lambda_n$, and hence $\lambda_2^k >> \lambda_3^k \geq \ldots \geq \lambda_n^k \geq 0$. Therefore, we can compute approximately the relative values that the second eigenvector $\vec{e_2}$ assigns to each node, by repeated applications of the matrix P to some initial values expressed by a vector $\vec{x}$ that satisfies $\sum_{i=1}^{n} e_{1i} x_i = 0$.

Of course, realize that, if, on the other hand, the chosen vector $\vec{x'}$ is not vertical to the first eigenvector $\vec{e_1}$, then repeated application of P on $\vec{x'}$ will result in

$$\vec{x'} P^k = \alpha'_1 \lambda_1^k \vec{e_1} + \alpha'_2 \lambda_2^k \vec{e_2} + \ldots \alpha'_n \lambda_n^k \vec{e_n} \simeq \alpha'_1 \frac{1}{n},$$

since $\lambda_1 = 1 > \lambda_2$ and the first eigenvector is $1/n$ on all nodes. It is therefore critical, for the computation of the second eigenvector, to start from a $\vec{x}$ that is perpendicular to the first eigenvector.

Let us now present how the calculation of (1) can be done in a distributed way.

First, we have to pick the vector $\vec{x}$, with $\sum_{i=1}^{n} e_{1i} x_i = 0$. Realize that $e_{i1} = 1/n$ for all nodes, so the requirement translates to $\sum_{i=1}^{n} x_i = 0$. It is now very easy to pick a suitable $\vec{x}$, if, certain links $\vec{\{u, v\}}$ (say chosen uniformly at random) contribute a value of $f_u = +\alpha$ to one of their endpoints, say u, and a value of $f_v = -\alpha$ to the other of their endpoints, say v. Then each link u considers as its initial value the sum of the values that have been contributed to u by all its neighbors: $x_u = \sum_{v:\{u,v\} \in E} f_u$. Realize that this calculation can be done in a completely distributed way. In fact, realize that synchronization is also not necessary.

Now let us explain the computation of one step of applying the matrix P to a vector $\vec{x}$. Let

$$\vec{y} = \vec{x} P. \quad (2)$$

For a d-regular graph, we have:

$$y_u = \frac{1}{2} x_u + \sum_{v:\{u,v\} \in E} \frac{x_v}{2d} \quad (3)$$

$$= \sum_{v:\{u,v\} \in E} \frac{x_u + x_v}{2d} \quad (4)$$

Now realize that equations (3) and (4) involve only local quantities of the graph G, which is essentially averaging of values of neighbors or averaging of the values of the endpoints of each link. Therefore, computing one application of the matrix P as in (2), that is, computing $\vec{y}$ from $\vec{x}$, can be done in a distributed way also. However, this computation clearly requires synchronization. We can think of a global clock, at each tick of which all averages of equations (3) or (4) take place.

We now proceed to discuss the practical significance of the relative values e_{2i} that the second eigenvector $\vec{e_2}$ assigns to the nodes i, $1 \leq i \leq n$. This is a heuristic that has been shown to hold true for special kinds of graphs (such as planar [6]) and, in general graphs under worse approximation factors.

In order to explain the significance of the values assigned to nodes by the second eigenvector, we need to introduce the notion of "cut sparsity". So, for a graph $G(V, E)$, and for some bipartition of its vertices $S \subseteq V$ and $\bar{S} \subseteq V$ with $S \cup \bar{S} = V$, and with $S' = S$ if $|\{\{u, v\} \in E : u \in S, v \in S\}| \leq |\{\{u, v\} \in E : u \in \bar{S}, v \in \bar{S}\}|$ the sparsity of the cut $S, \bar{S}$, $sp(S, \bar{S})$ is

$$sp(S, \bar{S}) = \frac{|\{\{u, v\} \in E : u \in S, v \in \bar{S}\}|}{|\{\{u, v\} \in E : u, u \in S', v \in S'\}|} \ .$$

Let $T, \bar{T}$ be a cut of the graph with the largest sparsity, i.e., $sp(T, barT) = \min_{S,\bar{S}} sp(S, \bar{S})$

The heuristic is the following. For each link $\{u, v\}$, let e_{2u} and e_{2v} be the value assigned by the second eigenvector to nodes u and v, and let us consider the absolute difference $|e_{2u} - e_{2v}|$. Then, if $|e_{2u} - e_{2v}|$ is much larger than the average over all links of the graph:

$$|e_{2u} - e_{2v}| \ >> \ \frac{\sum_{\{i,j\} \in E} |e_{2i} - e_{2j}|}{|E|}$$

then the link $\{u, v\}$ crosses a cut whose sparsity is "close" to the sparsest cut of the graph. Therefore such links can be considered *critical links*. In this paper, we associate link criticality of a link $\{u, v\}$ precisely with the difference $|e_{2u} - e_{2v}|$.

Experiments

We have performed the following experiments. We have considered two disjoint random d-regular graphs, each graph of size 5K nodes, and for $d = 6$ and $d = 8$. We have then performed the following rewirings. We have considered a small number of 250 random pairs of links $\{u_1, v_1\}$ and $\{u_2, v_2\}$ belonging to distinct initial components, and have replaced them with $\{u_1, v_2\}$ and $\{u_2, v_1\}$. We therefore clearly have a sparse cut corresponding to the two initial disconnected components. This sparse cut consists of 500 links. We have then considered an initial vector $\vec{x}$ with $\sum_{i=1^n} x_i = 0$. Using then (3) and (4), and applying them, for example, equation (3) 2.5M times, or equation (4) 2.5*dM times, starting at $\vec{x}$ (i.e. $k = 250$ in (1)), we have computed final values z_i, $1 \leq i \leq n$. We consider $\vec{z}$ a good approximation to the relative weights of the second eigenvector $\vec{e_2}$. For each link $\{u, v\} \in E$, we consider the difference $|z_u - z_v|$.

We now need to check, experimentally, if the differences $|z_u - z_v|$ are indicative of link criticality. Ideally, we would like to check if links for which $|z_u - z_v|$ is large are indeed crossing sparse cuts. But this problem is computationally hard (finding the sparsest cut is NP-complete [5]), so we turn to a computationally easy comparison.

We consider shortest path routing of one unit of flow between all pairs of nodes of the graph $G(V, E)$. This is computationally very easy to implement (we resolve ties, i.e., if there are more than one shortest paths between two nodes, we choose one at random). We now consider the number of shortest paths going through a link as an indication of a link belonging to a sparse cut. Intuitively and theoretically [7], we know that routing congestion indeed occurs across sparse cuts.

As we see from these experiments, the pure spectral method gives excellent results.

IV. Heuristics for Asynchronous Averaging

In this section we present how to implement a network computation that is both distributed and asynchronous. The computation maintains a vector that is always perpendicular to the first eigenvector according to (1), and simulates the steps in (4). Thus, this computation aims to approach the direction of the second eigenvector.

In general, the distributed asynchronous computation is as follows. First, a vector $\vec{x}$ is picked that is vertical to $\vec{e_1}$, like in Section III. Each link, spontaneously, assigns $+\alpha$ to one of its endpoints and $-\alpha$ to its other endpoint. Each node considers as its current load the sum of the values that have been contributed to the node by the endpoints of the links that are incident to this node. Note that the initial charges do not need to be picked synchronously, and the vector $\vec{x}$ might be changing as new values are added from the endpoints of links. However, the vector $\vec{x}$ will always remain perpendicular to the first eigenvector $\vec{e_1}$. This is because the equation $\sum_{i=1}^{n} x_i = 0$, since the sum of the values of the nodes is the sum of the values of the links at their endpoints, and each link contributes $+\alpha - \alpha = 0$.

Once some vector $\vec{x}$ has been picked (and, again, this may be changing during the computation), links start, in a distributed asynchronous way to implement (4). In

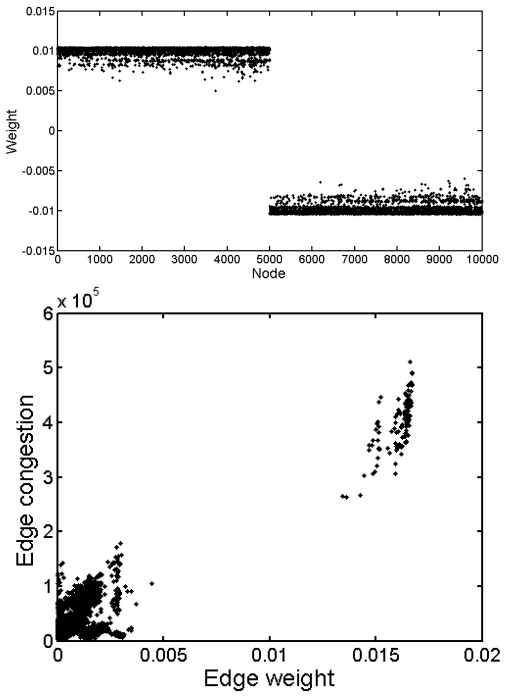

Fig. 3: **Top:** Weights assigned to the nodes by the second eigenvector. The topology is composed of two clusters of 5K nodes each; there are 250 inter-cluster edges; node degree is 6. **Bottom:** Correlation of the edge congestion to the absolute value of the difference of the weights assigned by the second eigenvector to the endpoints of the edge. Observe that inter-cluster links have large congestion and large difference in the weights of their endpoints.

particular, for some link $\{u, v\}$, let x_u be the values of node u and let x_v be the value of node v at a certain time. We think of x_u and x_v as partitioned in d parts, one part for each link incident to u and v. Spontaneously and independently from all other links, the link $\{u, v\}$ may perform the average

$$\frac{x_u/d + x_v/d}{2} = \frac{x_u + x_v}{2d} \ ,$$

thus implementing one step of (4) for one link. Note that this will also update the values of u and v. In particular, the new value of u will be

$$x_{u,\text{new}} = x_u - \frac{x_u}{d} + \frac{x_u + x_v}{2d} \tag{5}$$

and similarly we update the value of v:

$$x_{v,\text{new}} = x_v - \frac{x_v}{d} + \frac{x_u + x_v}{2d}$$

Realize that this operation maintains the sum $\sum_{i\in V} x_{i,\text{new}} = 0$:

$$\begin{aligned}
\sum_{i\in V} x_{i,\text{new}} &= \sum_{i\in V, i\notin\{u,v\}} x_i + x_{u,\text{new}} + x_{v,\text{new}} \\
&= \sum_{i\in V, i\notin\{u,v\}} x_i \\
&\quad + x_u - \frac{x_u}{d} + \frac{x_u + x_v}{2d} \\
&\quad + x_v - \frac{x_v}{d} + \frac{x_u + x_v}{2d} \\
&= \sum_{i\in V, i\notin\{u,v\}} x_i + x_u + x_v \\
&= \sum_{i\in V} x_i \\
&= 0.
\end{aligned}$$

Our computation simulates at each step the operation $\vec{x} \leftarrow P\vec{x} = \lambda_2\vec{x}$ (assuming that $\sum_i x_i = 0$). Hence, the values in the vector $\vec{x}$ are naturally decreasing. In the case of the synchronous computation of Section III this did not lead to numerical instabilities. However, when we experimented with asynchronous applications, as described above, we encountered numerical instabilities due to very small numbers. To solve this problem we needed to periodically normalize the vector; a good normalization is to divide the vector by its (Euclidean) norm (and this is compatible with the subgradient method of [1]. However, it is difficult to compute norms in decentralized systems. Instead, we used the following heuristic. After each step, the weights assigned to the endpoints of the link that was updated, are multiplied by a function of the number of updates, say t, that have taken place $f(t)$ with $f(t) \geq 1$ and $\lim f(t) = 1$; in our experiments we have used $f(t) = 1 + 1/t$. Observe that each node can maintain its local value of t, i.e. the number of updates it has performed, but during an update both endpoints need to use the same value of $f(t)$ to guarantee that the sum of weights remains zero. It is an interesting open question to establish formally functions $f(t)$ result in good numerical stability and fast convergence.

For simulation efficiency we performed multiple edge updates in parallel (the previous discussion assumed one update per step). In our simulations we update $100 - 2000$ edges in parallel (independently of the size of the network). We have observed that the number of parallel updates does not change significantly the results; the number of elementary steps required for convergence stays roughly the same as in Section III and the computed weights approximate well the second

eigenvector.

Experiments

We have performed the following experimental evaluation: We have considered initial topologies as in Section III, where two initial well connected d-regular topologies with 5000 nodes each have been connected by a sparse cut of 500 edges. We have started from some vector $\vec{x}$ over an entire network topology with sum of weights 0, as usual. In our experiments we have chosen the initial vector $\vec{x}$ as follows: (a) initially, $\vec{x} = 0$, (b) each edge $\{u, v\}$ gets a random number, α_{uv} between 0 and 1, (c) with probability $1/2$ we update $x_u \leftarrow x_u + \alpha_{uv}$ and $x_v \leftarrow x_v - \alpha_{uv}$; otherwise $x_u \leftarrow x_u - \alpha_{uv}$ and $x_v \leftarrow x_v + \alpha_{uv}$.

We use the distributed algorithm of Section IV for computing the second eigenvector and plot the computed values in Fig. IV. For Fig. IV we simulate 5K steps of the algorithm and, in each step, we update 500 distinct edges picked uniformly at random, for a total number of 2.5M elementary steps of (4), as in Section III. As expected the nodes of each cluster are assigned similar weights and the weights assigned to one cluster are positive and of the other cluster are negative. The relative error in the computation was on the average less than 5%; in some cases the error was up to 35%, but, still the sign of the weight was correct. Assuming that the weight of an edge equals the absolute value of the difference of the weights assigned to its endpoints, we observe that inter-cluster edges are assigned large weights (since their endpoints have weights of different sign).

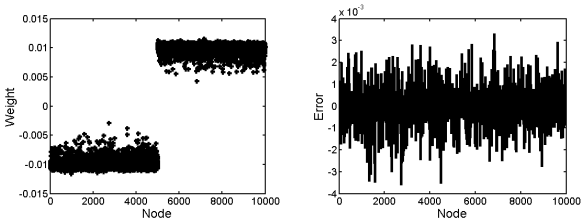

Fig. 4: The second eigenvector of a topology with two clusters with 5000 nodes each. Left: The weights assigned by the algorithm. Right: The error compared to the real eigenvector. (Average node degree is 8; 500 inter-cluster edges.)

As expected the distributed algorithm takes some time to converge to the estimates of the second eigenvector. In Figure IV we plot the node weights after 1K and 3K parallel steps (recall from Figure IV that the system converged after 5K parallel steps).

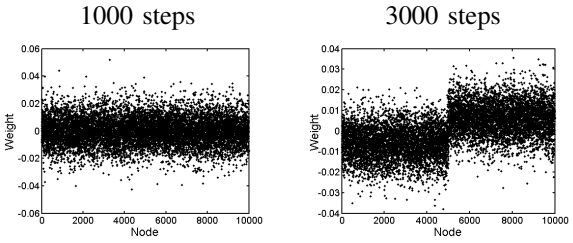

Fig. 5: The evolution of the eigenvector computation.

For presentation clarity, in most of this paper we examine topologies with two clusters. The methods described above and the distributed algorithm also work well in topologies with multiple clusters. In Figure IV, we plot the second eigenvector as computed by the distributed algorithm for a topology of 5 clusters of 20M nodes each (each node has degree 8; there are 1K connection between each pair of clusters). The weights can be used to identify at least one cluster and to infer the existence of 5 clusters in the topology. We have experimented with other topologies, with different sizes, number of clusters, and with clusters of different sizes, and observed that in all cases the distributed computation managed to converge fast and close to the real value.

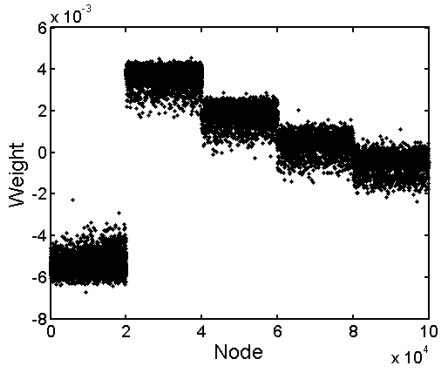

Fig. 6: The second eigenvector of a topology with multiple clusters. The nodes in each cluster are assigned similar weights. One cluster (the first from left) can be clearly separated from the rest of the graph; in addition, the difference in the weights in the nodes of the other clusters is still significant.

V. Application in Searching

The predominant searching method in unstructured P2P networks is search by flooding. This is essentially breadth first search of bounded depth. When a node initiates a query, the query is propagated to all nodes

that are at a constant distance from the initiating node (for existing P2P networks of a few million clients, this distance is 5-8). The reason that the depth of the flooding is bounded is in order to limit the load on the network. However, other ways of searching have been explored, e.g. see [8]–[11]

In [11], a generalized way of searching was suggested, which would preserve the load that a specific query imposes on the network, while improving on the number of hits of the queried nodes. This generalized search scheme assumed topology awareness by assuming weights on the links of the network -presumably links crossing or near to sparse cuts should have increased weight. The method of [11] is the following. Let L be the total load (i.e., number of nodes queried) that a certain node u initiating a query wishes to impose on the network. Let v be a neighbor of u, and let f_{uv} be the weight of the link connecting u to v. Then, for each neighboring node v of u, the query is passed, together with a remaining load proportional to f_{uv}, i.e., a budget of

$$\frac{(L-1)f_{uv}}{\sum_{w:\{w,v\}\in E} f_{vw}}$$

. The process is repeated, until a node receives a query with a budget of zero, in which case the query is no longer propagated.

In [11], and as a demonstration of the potential effectiveness of the above generalized search scheme, the generalized search scheme was examined when the weights were determined by the congestion of a link under shorted path routing of one unit of flow between all nodes. Of course, performing shortest path routing between all pairs of nodes is not practical in a P2P network. Therefore, here we shall examine the effectiveness of the generalized search scheme, when the link weights are determined by the second eigenvector method, when this eigenvector is computed either synchronously, as in (3) and (4), or asynchronously, as in Section IV.

In Figure V we plot the number of distinct nodes queried by four searching approaches; observe that the performance of searching, i.e. the number of copies of the file discovered, directly relates to the number of distinct nodes discovered. We have performed searching starting from 1000 random nodes in a topology of 100M nodes, which is composed of 5 clusters of 20M nodes each. The average node degree is 8 and there are 1K links between each cluster of nodes. All algorithms were configured to use the same number of messages (24K messages). Observe that flooding using the edge weights performed significantly better than both regular flooding (by almost 25% on the average) and random walks (by 8% on the average). Also, observe that the performance of searching using the weights computed by the distributed eigenvector computation is very similar to the performance of searching using the real eigenvector to compute the edge weights. In summary, the use of the weights computed by the distributed algorithm resulted in better searching performance compared to both flooding and random walk, and the searching performance was very close to that expected if we could accurately estimate the second eigenvector.

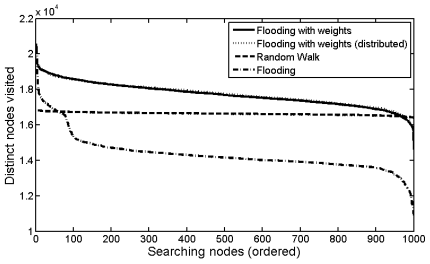

Fig. 7: Performance of searching.

VI. Application in Topology Maintenance

In this Section we examine the usefulness of link criticality in the maintenance of a well connected network topology.

The following method of maintaining a well connected network topology based on the Markov chain randomizing method has been suggested and formalized in [12]–[14]. Let $G(V,E)$, $|V|=n$, $|E|=m$ denote the topology of a network. Links $\{u_1,v_1\}$ and $\{u_2,v_2\}$ are chosen uniformly at random and are replaced by links $\{u_1,v_2\}$ and $\{u_2,v_1\}$. In [12] it was shown that the corresponding Markov chain on the state space of all graphs with the same degree sequence as G is "rapidly mixing." That means that a random such graph is reached by performing a polynomial in n number of such exchanges. Considering the fact that most of these exchanges can be also been performed in parallel, the result of [12], complemented by the simulations of [13] signify that random link exchanges are effective in bringing the network topology in a "random" configuration, hence, by standard graph theory [3], [4] a configuration with good expansion, conductance and consequently nearly best possible connectivity.

But how can links $\{u_1,v_1\}$ and $\{u_2,v_2\}$ be picked, in practice, uniformly at random? In a P2P network, each client has only limited knowledge of the rest of the network, typically a constant number of nodes in its neighborhood. This implementation issue was addressed

in [14] who considered only exchanges of the following form: If $\{u_1, u_2\}$ is a link of G, only then the pair $\{u_1, v_1\}$ and $\{u_2, v_1\}$ can be chosen and exchanged, the significance being that all nodes u_1, u_2, v_1 and v_2 are within constant distance and are hence aware of each other. [14] showed that even this Markov chain, with very realistic restricted exchanges is also rapidly mixing. It was thus established, in a very practical sense, that the random link exchange Markov chain method (which is also part of several real protocols [15]) is an effective protocol for maintaining well connected network topologies.

Another way of implementing the random link switching of the Markov chain of [14] is the following: A link $\{u_1, u_2\}$ is chosen uniformly at random among all links, and links $\{u_1, v_1\}$ and $\{u_2, v_2\}$ are exchanged with links $\{u_1, v_2\}$ and $\{v_2, u_1\}$. Again, the significance of doing the switches locally is that, in practice, since nodes u_1, u_2, v_1 and v_1 are in constant distance with each other, it is realistic to expect that they "know" of each other and can perform the local exchanges. Again, it follows from [14] that this Markov chain is rapidly mixing. We shall call this Markov chain the *flip Markov chain*.

We have performed the following experiments. We have started with two random d-regular topologies, connected with a few links as in the previous sections. We have then performed the flip Markov chain under two scenarios. We measure the connectedness of the resulting topology, after application of a few steps of the Markov chain by computing the second eigenvalue of the stochastic normalization of the underlying graph. It has been well established [3], [4], [10], [16] that the second eigenvalue is an excellent metric of the good connectivity properties of the underlying graph.

In the first scenario we have performed the flip Markov chain on the graph underlying the topology, by picking initial links $\{u, v\}$ uniformly at random, without taking into account any weights on the links of the underlying graph.

In the second scenario we have performed the flip Markov chain on the graph underlying the topology, however, we pick initial links $\{u, v\}$ with probabilities proportional to the weights given on the links by the approximate distributed asynchronous computation of the second eigenvector, as suggested in Section IV. and indicated in (3) and (4).

We expect that, since links crossing the sparse cut have larger weights according to the second eigenvector, convergence of the flip Markov chain will be faster in the second experiment.

Steps	Naive swaps	Weight-based swaps
0	0.9952	0.9952
100	0.9952	0.9757
200	0.9952	0.9706
1000	0.9951	0.9457
2000	0.9951	0.9110
5000	0.9949	0.8102
15000	0.9907	0.6630

TABLE I: Evolution of λ_2 as a function of the number of edge switches. Without taking into account the weights in the edges (naive swaps) the topology improves, i.e. the number of inter-cluster edges increases, very slowly. By using the weights, the nodes quickly discover the cut in the network and construct edges so as to increase the number of inter-cluster links.

VII. Application in Routing Congestion

The last application that we give in maintaining topologically aware networks is in routing. Typically, networks perform shortest path routing, where each link counts for one hop. Instead, we propose shortest path routing, where each link $\{u, v\}$ has a length proportional to $|e_{2u} - e_{2v}|$, i.e. the difference of the values assigned to its endpoints by the second eigenvalue of the underlying topology. Intuitively, since links that are crossing sparse cuts have large weights, such links will be considered "long" and hence avoided by several paths, thus resulting in better load balancing.

We have performed the following experiment. We have considered three distinct sparse graphs A,B and C, each of 6K nodes. We have added very few links between A and B , and many more links between A and C and B and C (5 links between A and B, and 2000 links between A and C, and B and C). We then considered one path between each pair of nodes of the entire graph. In the first case we did shortest path routing. In the second case, we assigned lengths to links, as suggested by the second eigenvector, and performed Dijkstra's shortest path routing from each node to every other node.

We call the number of shortest paths that pass through a link the congestion of the link. In Figure VII we plot the congestion of the most congested links. As expected the links between the clusters A and B are the most congested. However, by using the edge weights and routing traffic around edges with high weights, we manage to reduce the traffic in the most congested links; in this case the traffic flows from A to C and then to B. Indeed, the traffic in the most congested link dropped by a factor of almost 2.

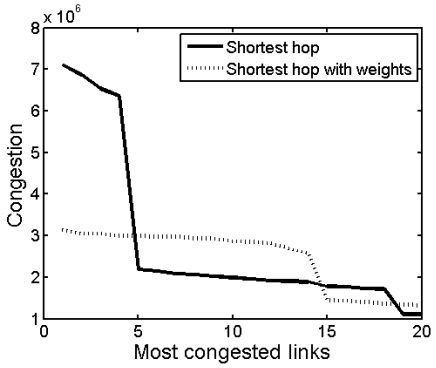

Fig. 8: The congestion of the 20 most congested links with shortest-hop routing and with using the edge weights. Observe that with the edge weights the congestion of the most congested links drops to half.

VIII. Summary, Conclusions and Further Directions

We have considered methods to enhance networks with topology awareness. We discussed and applied exact methods based on semidefinite programming which put the correspoding problems in P, but for which we do not know decentralized algorithms. Any progress towards decentralizing such algorithms is significant in many networking applications.

We considered a distributed asynchronous implementation of the spectral method for identifying critical links. We showed that the method is efficient, and that awareness of such link criticality is usefull is searching and topology maintainance in P2P networks, as well as routing in general networks.

In most of the experiments that we discuss here we have considered a graph with a single sparse cut. We have repeated similar experiments in graphs with several sparse cuts; the results remain the same.

All the cases that we have presented in this paper concern regular graphs. Exact synchronous computation of the second eigenvector of non-regular graphs (as in Section III) is known, however, the eigenvectors of non-regular graphs are not necessarily orthonormal, and asynchronous distributed simulation of the computation of the second eigenvector of such graphs is a very interesting open problem. Of course, if the graphs are "nearly regular" our methods still give good results, if we consider the maximum degree of the graph $d_{\max}$, replace all transitions of the matrix P by $1/2d_{\max}$ and increase the weights of the self loops accordingly.

Finally, we should talk about node failures. The method discussed here is particularly sensitive to failures of elements, due to its numerical sensitivity to the invariant that the sum of the values over all nodes should always add up to 0. If the protocol can guarantee that a failing node distributes its weight to one, or some of its neighbors before leaving the network, then the problem of node failing can is solved. However, it is not easy to assume that this can be easily implemented in a protocol. In addition, we could have the case of several nodes failing at the same time. This issue of numerical sensitivity is, indeed, more general in all spectral methods. We therefore state the issue of designing a protocol that can sustain certain node failures as a very important open problem in the current networking context.

References

[1] Stephen Boyd, Persi Diaconis, and Lin Xiao, "The fastest mixing markov chain on a graph," *SIAM Review, vol 46, no 4*, 2004.

[2] Arpita Ghosh, Stephen Boyd, and Amin Saberi, "Convexity of effective resistance," *manuscript*, 2006.

[3] Fan Chung, "Spectral graph theory," *CBMS Series in Mathematics*, 1997.

[4] Alistair Sincalir, "Algorithms for random generation and counting: A markov chain approach," *Birkhauser*, 1993.

[5] Sanjeev Arora, Satish Rao, and Umesh Vazirani, "Expander flows, geometric embeddings, and graphs partitionings," *ACM-STOC*, 2004.

[6] Daniel Spielman and Shang-Hua Teng, "Spectral partitioning works: Planar graphs and finite element meshes," *IEEE-FOCS*, 1996.

[7] Thomson Leighton and Satish Rao, "An approximate max-flow min-cut theorem for uniform multicommodity flow problem with application in approximation algorithms," *IEEE-FOCS 88, and JACM*, 1999.

[8] Y. Chawathe, S. Ratnasamy, L. Breslau, and S. Shenker, "Making gnutella-like peer-to-peer systems scalable," *ACM-SIGCOMM*, 2003.

[9] Qin LV, Pei Cao, Kai Li, and Scott Shenker, "Search and replication in unstructured peer-to-peer networks," *Sigmetrics*, 2002.

[10] Christos Gkantsidis, Milena Mihail, and Amin Saberi, "On the random walk method in peer-to-peer networks," *INFOCOM*, 2004.

[11] Christos Gkantsidis, Milena Mihail, and Amin Saberi, "Hybrid search schemes in unstructured peer-to-peer networks," *INFOCOM*, 2005.

[12] Colin Coopen, Martin Dyer, and Catherine Greenhill, "Sampling regular graphs and a peer-to-peer network," *SIAM-SODA*, 2005.

[13] P. Mahlmann and C. Schiendelhauer, "Peer-to-peer networks based on random transformations of connected regular undirected graphs," *SPAA, Symposium on Parallel Algorithms*, 2005.

[14] Tomas Feder, Adam Guetz, Milena Mihail, and Amin Saberi, "A local switch markov chain on given degree graphs with application in connectivity of peer-to-peer networks," *IEEE-FOCS*, 2006.

[15] Hector Garcia-Molina, "Building primitives of peer-to-peer systems," *Stanford University Project*.

[16] Christos Gkantsidis, Milena Mihail, and Ellen Zegura, "Spectral analysis of internet topologies," *INFOCOM*, 2003.