The flip Markov chain and a randomising P2P protocol

Colin Cooper
Department of Computer Science
Kings College
London WC2R 2LS, UK
colin.cooper@kcl.ac.uk

Martin Dyer, Andrew Handley
School of Computing
University of Leeds
Leeds LS2 9JT, UK
{dyer,jobriath}@comp.leeds.ac.uk

Abstract

We define a network that relies on its protocol's emergent behaviour to maintain the useful properties of a random regular topology. It does this by spontaneously performing flips in an effort to randomise [16], allowing it to repair damage and to embed new peers without over-complicated joining schema.

The main theoretical result of this paper is informed by the need to show that flips randomise the network quickly enough. Formally, we show that performing random flip operations on a regular graph of size n will rapidly sample from all such graphs almost uniformly at random (i.e. with error ε). Here, "rapidly" means in time polynomial in n and $\log \varepsilon^{-1}$. This is done by extending a similar result for random switches, obtained in [3], using a two-stage direct canonical path construction. The directness of our approach allows for a much tighter bound than obtained in previous work.

1 Introduction

A pure peer-to-peer network consists of peers (computers) sharing a common protocol in which all peers have identical responsibilities and behaviour. The lack of a central authority permits excellent robustness and efficiency, as the network becomes difficult to disrupt and there is no central bottleneck on communication or performance.

In practice pure peer-to-peer networks are rare, and hybrid approaches with "superpeers", or where some peers can spontaneously assume different behaviour, are more common [1, 17, 10]. For instance, the modern Gnutella [10] protocol works by nominating a close-knit community of high-degree "ultrapeers". Poorly-connected leaves search by forwarding their queries to a small number of these ultrapeers, which forward it on in turn. The advent of ultrapeers meant queries experienced satisfactory network coverage after around only 4 hops, instead of 7.

[17] seeks to address the shortcomings of the Gnutella open protocol. Their approach results in constant-bounded degree, logarithmic diameter, and good connectivity under very harsh adversarial conditions. It does this by maintaining a central cache that samples a random subset of peers from the network for the use of joining peers, sacrificing immunity to a single point of failure. Current Bittorrent [1] incarnations have "trackers" which perform a similar duty, and there are many extensions to remove this bottleneck (see [9] for an overview and for an example relying on random walks).

Other protocols focus on searching using Distributed Hash Tables, at the expense of other peer-to-peer capabilities and free-text searching, Two prominent examples are Chord [22] and CAN (Content-Addressable Network [19]). In Chord, each peer address and each data key is hashed onto a circular (scalar with boundary conditions) keyspace. Responsibility for a given datum is given to the peer whose address hash follows the key hash; hence a "good" hash function can guarantee reasonably fair distribution of data across the peers. Each peer maintains knowledge of peers immediately before and after it in the keyspace; and also a "finger table" of the peers responsible for the data points $\{2^i\}$ after it, for suitable i. Both routeing and searching are efficient: by using these fingers as shortcuts, a query can half the keyspace-distance to its target each hop, resulting in logarithmic search times. CAN assigns peers partitions of a hypercubic keyspace, and hence accomplishes similar goals.

These and similar approaches typically require a lot of hard work in order to establish rigorous guarantees on their behaviour (such as logarithmic diameter, connectivity and so on). Another group of protocols centres on short-cutting this effort by producing a topology that corresponds to the well-characterised random regular graph. Once it has been shown that the topology becomes random regular, then a wealth of results apply. A graph is d-regular when every vertex has exactly d neighbours, and *random* regular when it is chosen randomly from the space of all such graphs. These graphs make excellent peer-to-peer topologies, for the following reasons [11]:

- Random regular graphs are expanders with small (logarithmic) diameter [24]. (A graph is an expander if every set of vertices $S, |S| \leq n/2$ is adjacent to more than $|S|$ vertices not in S.) On an expander, both a random walk's mixing time and a flood's coverage time are $\mathrm{O}(\log n)$.
- Deleting vertices at random will not disconnect the graph with high probability (w.h.p.). Even if a disconnection does occur, only small subgraphs splinter off from a still-functional giant component. (This was also observed in real-world Gnutella networks [20].)
- There are d disjoint paths between any two vertices, meaning bandwidth can be efficiently put to use.
- Due to their by-design lack of structure, it is not clear to a hostile adversary which peers to disable in order to cause a significant network split. Anything less than full knowledge of the topology risks leaving peers bridging the gap.

In order for the network to be scalable, no peer can have knowledge of a significant part of the topology; otherwise, keeping its knowledge current would become prohobitively expensive. (The protocol defined in section 3 requires knowledge of no more than a constant number of peers.) Despite this, peers occasionally require a sample of more global knowledge. Consider peers negotiating joins with a "gateway" peer already in the network. If they connect only to peers local to the gateway, their neighbourhood will strongly resemble the gateway's, rather than a random sample of the network.

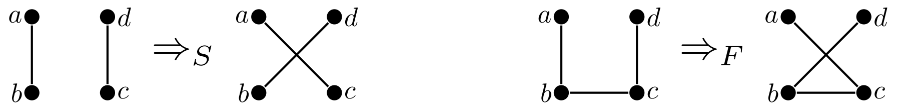

Figure 1: Left: a switch operation, denoted by $\Rightarrow_S$. Right: a flip, denoted by $\Rightarrow_F$.

The direct approach to overcoming this limitation of locality is to make information mobile. A newcomer in [2, 3] initiates $d/2$ random walks and waits until they mix in order to sample edges at random from the graph. In [4] each peer releases a number of address tokens that perform independent random walks, providing ready, mixed samples (and allowing disconnected networks to reconnect themselves).

The alternative is to continue the trend towards relying on emergent properties of the network protocol. If the network were to constantly strive to randomise itself then it could recover from the effects of any joining scheme or network damage, as long as it were given enough time, and remained connected. The device used in this paper (and in [16], and implicit in [8]) is a small, random topology-altering operation. Two are common: the switch and the flip, as pictured in Figure 1. The flip can be thought of as a switch where the switch edges are exactly one edge apart, and this locality implies that it is more suitable to a peer-to-peer protocol.

Kannan et al. [14] considered the switch Markov chain on bipartite graphs, and Cooper et al. [3] gave a bound on its mixing time over general regular simple graphs. In section 2 we extend their result from switches to flips using a two-stage direct canonical path construction. The flip was previously studied in this context in [16]. They show that a Markov chain (let us call it $\mathcal{M}_F$) over connected regular graphs, with transitions corresponding to a single flip, will eventually sample uniformly at random from all connected regular graphs. This implies that a connected regular network spontaneously performing random flips upon itself will indeed tend to acquire the desirable properties of a random regular graph. However, they do not provide a bound for the mixing time of $\mathcal{M}_F$. (They do show that a similar chain with the flip edges $\Theta(d^2n^2 \log \varepsilon^{-1})$ edges apart will mix in time $\mathrm{O}(dn)$).

A bound for $\mathcal{M}_F$ was subsequently given by [8] using a comparison argument [5, 18, 7] with the switch Markov chain, $\mathcal{M}_S$, and the bound obtained for it in [3]. They solve a fundamental problem (that flips cannot handle disconnected graphs, while switches can) by embedding $\mathcal{M}_S$ into a connected-only switch chain, showing how this can be done in such a way that a path in $\mathcal{M}_S$ is only polynomially lengthened. Their result relaxes the bound in [3], $\tau_{\mathrm{CDG}}(\varepsilon)$, by $\mathrm{O}(d^{41}n^{45})$. By employing a two-stage direct construction, we are able to give a much tighter result, relaxing $\tau_{\mathrm{CDG}}(\varepsilon)$ by just $\mathrm{O}(n^6d^5)$.

A final note: in practical networks, even-regular graphs are more convenient as they permit simple schema for joining and parting, and allow the graph size n to have either parity. For this reason we assume that d is even throughout this paper. Note that there is nothing to suggest that the result of section 2 relies upon this assumption, but it does clarify the argument slightly.

1.1 Useful definitions

Unless otherwise stated, G will be the simple graph $G = (V, E)$ where $V = \{1, 2, \ldots, n\}$ and $E \subseteq 2^V$. Vertex set S has an edge boundary $\partial_E S = \{uv \mid u \in S, v \notin S\}$, which contains the edges crossing from inside to outside of S. If a property is said to hold of a family of graphs "with high probability" (w.h.p.) or "asymptotically almost surely" (a.a.s) we mean that a graph of size n will satisfy that property with probability $1 - o(1)$. For a graph to be c-edge-connected there must exist c edge-independent paths between any two vertices; for it to be c-vertex-connected the paths must be vertex-independent. A graph's edge-connectivity can be higher than its vertex connectivity. A c-edge-cut of graph G is a set of edges of size c that when removed from G renders it disconnected. A c-vertex-cut is defined analogously. Throughout this paper, unless otherwise specified we indicate edge-connectedness or -cuts.

We make use of the following structural properties of d-regular graphs with even d. As already discussed, this simplifying assumption gives rise to little loss of generality in the context of peer-to-peer applications.

Lemma 1. *If $d = 2r$ for a positive integer r, then a d-regular graph G has the following properties:*

(i) For any $S \subseteq V(G)$, the edge boundary of S, $\partial_E S$, is even.

(ii) Every edge of G lies in some cycle.

Proof.

(i) We argue by contradiction. Assume otherwise, so that there is an odd number $k = 2k' + 1$ of edges incident upon some set S. Let $G[S]$ be the subgraph induced by S. Let $|S| = s$, $d = 2r$, and suppose $G[S]$ has m edges.

The sum of degrees of the vertices in S is ds. Exactly k edges have one endpoint in S, so the sum of degrees in $G[S]$ is $ds - k = 2rs - 2k' - 1 = 2(rk's) - 1 = 2m$. But no integer m satisfies this equation. Hence our assumption is false, and there is an even number of edges incident on S.

(ii) Again, we argue by contradiction. Assume we have an edge $e = uv$ that is not on any cycle in its connected component C. Removing e must cause C to become disconnected, since there is no other path from one endpoint of e to the other (or there would be a cycle containing e).

Consider the connected component C_v containing v that results from removing edge e. There is no edge except e incident upon C_v in G. This is an odd number of edges, and we showed in (i) that this cannot be the case. Hence e must lie on a cycle. □

1.2 Showing rapid mixing: the canonical paths argument

In [3] a multicommodity flow/canonical path argument [6, 13, 12, 21] is used to bound the mixing time of $\mathcal{M}_S$. The argument is based on showing that it is possible to route $\pi(x)\pi(y)$ units of "flow" along an x–y path in $\mathcal{G}_S$, for all pairs $x, y \in \Omega_S$, without any edge in Σ_S becoming too congested. The mixing time can then be bounded in terms of maximum congestion, path length, and ε, the permitted sampling error. (See [12] for details.) Here, we perform a two-stage direct canonical path construction. What this means is that canonical paths in $\mathcal{M}_F$ are defined in terms of (intuitively: "simulate") those of $\mathcal{M}_S$.

We will denote the respective Markov chains' transition matrices by P_S and P_F, their stationary distributions by π_S and π_F, and their underlying graphs by $\mathcal{G}_S = (\Omega_S, \Sigma_S)$ and $\mathcal{G}_F = (\Omega_F, \Sigma_F)$. The underlying graph of P_X has $xy \in \Sigma_X$ if and only if $P_X(x, y) > 0$ ($X \in \{S, F\}$).

The *mixing time* $\tau(\varepsilon)$ of a Markov chain is given by

$$\tau(\varepsilon) = \max_{x \in X} \min \{T \geq 0 \mid \mathrm{d_{TV}}(P_x^t, \pi) \leq \varepsilon \text{ for all } t \geq T\},$$

where P_x^t is the distribution of the random state X_t of the Markov chain after t steps with initial state x, and $\mathrm{d_{TV}}$ is the *total variation distance* between two probability distributions.

A *flow* in $\mathcal{G}$, is a function $f : \mathcal{P} \to [0, \infty)$ satisfying

$$\sum_{p \in \mathcal{P}_{xy}} f(p) = \pi(x)\pi(y), \text{ for all } x, y \in \Omega, x \neq y,$$

where $\mathcal{P}_{xy}$ is the set of all simple directed paths from x to y in $\mathcal{G}$ and $\mathcal{P} = \bigcup_{x \neq y} \mathcal{P}_{xy}$. A set of canonical paths clearly defines a flow. Extend f to a function on oriented edges by setting

$$f(e) = \sum_{p \ni e} f(p),$$

so that $f(e)$ is the total flow through e. Define the *capacity* $Q(e) = \pi(x)P(x, y)$ for the edge $e = xy$, and let

$$\rho(e) = f(e)/Q(e)$$

be the *load* of the edge e. The *maximum load*, a measure of the congestion of the flow, is

$$\rho(f) = \max_e \rho(e).$$

Then the mixing time of $\mathcal{M}$ can be bounded above in terms of these quantities by

$$\tau(\varepsilon) \;\leq\; \rho(f)l(f)\bigl(\log(1/\pi^*) + \log(1/\varepsilon)\bigr), \tag{1}$$

where $\pi^* = \min\{\pi(x) \mid x \in \Omega\}$ and $l(f)$ is the length of the longest path such that $f(p) > 0$.

The canonical path method [13] is often sufficient to obtain a suitable flow. Suppose we have an injective *encoding function* $\eta_e : cp(e) \to \Omega_S$ which maps, for every transition $e \in \mathcal{G}_S$, each canonical path that makes use of e onto a state in Ω. This bounds the number of paths running through e, and it follows that the load cannot be too great.

On the canonical path in $\mathcal{M}_F$, we use the encoding for $\mathcal{M}_S$ while we are simulating a switch by flips. This creates a minor technical problem, since $\Omega_F \subset \Omega_S$, and so the encodings can have different size. However, for large n, $|\Omega_F| \sim |\Omega_S|$ [24], so this issue can be ignored.

The main result of this paper is the following bound on the mixing time of the flip chain.

Theorem 1. *The mixing time of $\mathcal{M}_F$ is $\mathrm{O}\bigl(n^{16}d^{22}(dn\log dn + \log\varepsilon^{-1})\bigr)$, of which the two-stage direct construction is responsible for a factor of $\mathrm{O}(n^6d^5)$.*

2 Analysis of the flip chain

2.1 Simulating a switch: the long-flip

The core idea is to simulate an arbitrary switch by flips. The difficulty caused by the graph becoming disconnected is dealt with in the following section, where we define a mapping $\mathcal{H}$ from disconnected to connected graphs. For the time being, suppose G is connected, and hence $\mathcal{H}(G) = G$, and the switch to be simulated does not disconnect G. Mahlmann and Schindelhauer [16] introduced a switch-simulating operation which they called "k-flipper". Here k is the distance between the switch edges. We define a similar operation but, since we are not interested in k, we will simply call it a "long-flip". Our main alteration is to make it compatible with the canonical path argument.

Suppose that the canonical path in Ω_S from x to y is $Z_1, Z_2, \ldots, Z_\ell$, where $Z_R \in \Omega_S$ $(R = 1, 2, \ldots, \ell)$. Given a transition $t_S = (Z_R, Z_{R+1})$ on this path, its encoding L, and a "pairing" ϕ, [3] shows how to determine the next switch. These are specified as 4-cycles (a, b, c, d) (Figure 2), meaning a simulated switch must remove the "switch edges" $ab, cd \in Z_R$ and insert precisely $ad, bc \in Z_{R+1}$. We will denote the states encountered in simulating the switch (Z_R, Z_{R+1}) by $\mathcal{H}(Z_R) = X_0, \ldots, X_F, X_{F+1}, \ldots, X_\kappa = \mathcal{H}(Z_{R+1})$. In simulating a switch by flips, the information available to the long-flip operation is only the current flip transition $t_F = (X_F, X_{F+1})$. The information in the encoding is used only by the simulated switch chain, although with additional technical complexity it might be possible to reduce the amount of information that currently must be "guessed". Then, given the transition from X_{F+1} to X_{F+2}, the encoding, and a small amount of additional information, the canonical path method requires us to be able to reconstruct the entire canonical path from x to y. We refer to the additional information as the "guesses" which we must make in the relevant state.

The process for simulating a connected switch by flips is illustrated in Figure 2. Observe that the final configuration **D** is the same as the original **A** (though it is shown with the path p reversed), except that the edges ab and cd have been switched to ad, bc, as required. Our approach is to perform a series of flips which gradually move a's neighbour b along the path p until it is past d and adjacent to c. We refer to b in this case as the "bubble", as it percolates along the path. Having moved b, the new neighbour of a, p_1, becomes the next bubble until it too crosses d on path p. By repeating this with subsequent neighbours of a we can reverse the path as required, and complete the simulation in $\kappa = \frac{1}{2}\nu(\nu-1) \leq \frac{1}{2}n(n-1)$ flips.

The directed path p will be chosen to have following properties:

1. There are no edges between non-consecutive vertices on the path, except possibly from c (note that there cannot exist the edge bc because then t_S would not be a switch). This is needed because each vertex except c will at some point become adjacent to every other, so there cannot already exist edges between them. But c is only ever adjacent to d or b, and hence need not be so restricted.

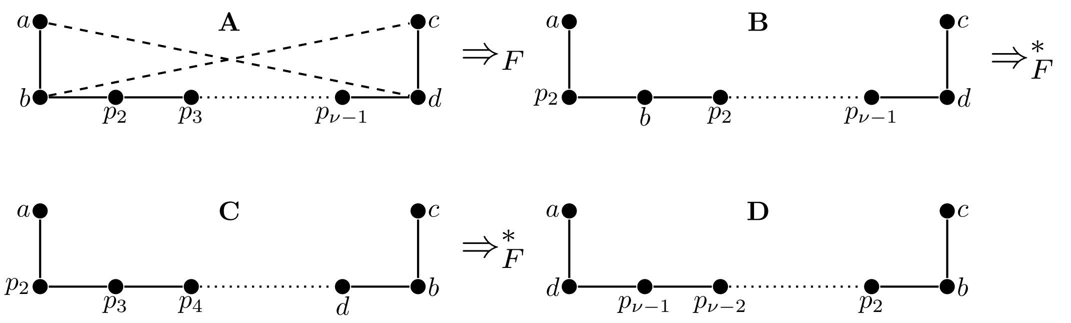

Figure 2: **A** shows switch path (a, b, c, d) in alternating solid ($\in Z_R$) and dashed ($\in Z_{R+1}$) edges, with some reconstructable path $(p_1 = b, p_2, \dots, p_\nu = d)$ overlaid. **B** shows a single flip, effectively swapping b closer to d. **C** shows b in its final position before the rest of p is reversed, and **D** shows the reversed p, where the switch has been fully simulated.

2. Let $u, v \in p$. The unique least (u, v)-path in Z_R for some total ordering on paths must be a subpath of p. We call this property "recognisability", and it enables us to recover sections of p between two of its vertices during the running of the algorithm below.

We now show how to create the path p. Note that p itself is only valid on $\mathcal{H}(Z_R) = X_0$, i.e. before we begin the simulation; we will use the notation p' for paths defined in subsequent X_F's. Identify each vertex in $V(G)$ with its numerical index $1, 2, \dots, n$. Under this ordering, let p be the lexicographically least shortest path from $(ab \dots dc)$ or $(ba \dots cd)$. Without loss of generality, we will refer to it as $(abp_2 \dots p_{\nu-1}dc)$. It follows that p is unique.

Note that p is a directed path. Since it is a shortest path, only consecutive vertices among $b, p_2, \dots, p_{\nu-1}, d$ can be adjacent, but we must examine the termini a and c. There are two possibilities:

- At most one of a or c is adjacent to another vertex in $(p_2 \dots p_{\nu-1})$. They cannot both be, except in the case below. Otherwise the symmetric definition of p would have found a shorter path between the two switch edges ab, cd, a contradiction. Without loss of generality, let c be either adjacent to another vertex in p, or have the higher index of a and c.

- There are triangles (a, b, p_2) and $(p_{\nu-1}, d, c)$. In this case we must perform a flip to break one triangle, allowing the above condition to obtain. This flip will have to be undone to restore the triangle when the long-flip is complete. Without loss of generality, we break the former triangle with the flip (b, p_2, p_3, v_x), where v_x is some vertex adjacent to p_3 that is not adjacent to p_2 (we know such a vertex exists by d-regularity with $d \geq 4$). Redefine path p as $(abp_3 \dots)$ on this newly acquired graph, then relabel the path vertices to be numbered contiguously $(p_2, p_3, \dots)$. Analysis of cases confirms that this new p remains lexicographically least. When undoing this flip we call the old p_2 "v_{del}" and we must remember v_x.

Looking closely at the path reversal process, we see that the part of the path being worked upon shrinks with each new choice of bubble. The unreversed section is between a and d, and d moves closer to a each time a bubble passes it. Hence an individual flip in the long-flip operation requires only the directed path $p' = (ap'_1 \dots p'_{j-1}dp'_{j+1})$, which *initially* is p as defined above.

We now show how to determine the next flip given the current one. In order to do this we must construct path p' on which the flip will be carried out. However, we are not given enough information to uniquely determine the path, and so we must "guess" some things. The precise mechanism will be examined later, but for now we will note only that each guess incurs a factor in the mixing time bound for $\mathcal{M}_F$ on the order of the number of possibilities from which the guess is made.

The structure of the problem permits us to build p' by guessing a and d, as well as their respective neighbours a' and d' on the path p'. Here a' is the next bubble, and d'—which is c if $p' = p$—is the previous bubble.

The current flip comprises the two edges $X_F \cup (X_F \triangle X_{F+1})$ and the single edge adjacent to both (where $\triangle$ is the symmetric difference). We know that the bubble p'_i is the closer to a of the middle vertices of the

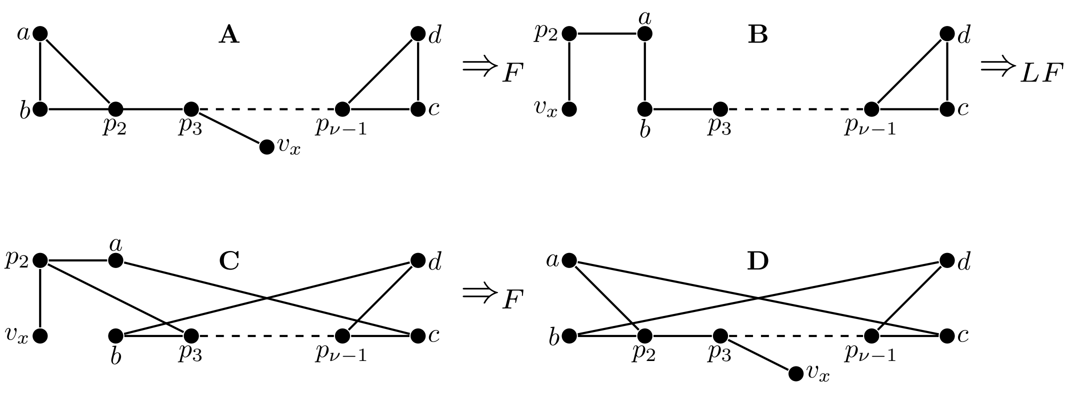

Figure 3: **A**: a long-flip with a triangle on both switch edges ab, cd. **B**: the triangle flip destroys the ab triangle using x. Let $p_2 = v_{\text{del}}$ for future reference. **C**: the long-flip goes ahead as normal. **D**: the pre-existing triangle is reinstated.

3-path $(p'_{i-1}, p'_i, p'_{i+1}, p'_{i+2})$, but unless $p'_{i-1} = a$ or $p'_{i+1} = d$ we must guess which is p'_i, for an additional mixing time factor of 2.

The rest of p' is formed by observing that the lexicographically least (a', p'_{i-1})- and $(p'_{i+1}p'_{i+2}, d)$-paths in X_F are, by the "recognisability" property, subpaths of the original p, except when one of the following causes a "shortcut":

- vertex c's potential to be adjacent to multiple vertices in p; or
- one of the "added" edges in $A = X_F \backslash X_0 \subseteq \{aa', p'_{i-1}p'_i, p'_ip'_{i+1}, dd', v_{\text{del}}v_x, cb\}$.

Hence we define two paths q_1 and q_2 to be the lexicographically least such paths that do not include c, a, d, p'_i and v_{del} (the latter four of which cover the edges in A). Unfortunately this means we must also guess the identity of c and v_{del}, the latter of which we can do by its adjacency to a. It is possible that a more careful construction of p could obviate this requirement, but we will not consider this here. Then q_1 and q_2 are subpaths of p, as we have precluded any chance of an accidental shortcut and the "removed" edges $R = X_0 \backslash X_F \subseteq \{ab, bv_{\text{del}}, d'p'_i, p'_ia', p'_{i-1}p'_{i+1}, cd\}$ fall outside of q_1 and q_2.

So let $p' = (aq_1p'_iq_2d')$. We now have enough information to specify the algorithm that determines the next flip. In each of the following cases, unless the algorithm halts, the next flip is given by $(p'_{i'-1}, p'_{i'}, p'_{i'+1}, p'_{i'+2})$.

- If $a' = d$, the path is now reversed and we are done, *unless* we had to break a triangle on a at the beginning of this simulation. Since we have no information about this we have to guess whether this was the case and, if it was, the vertices of the flip $(b, p_2, v_{\text{del}}, v_x)$ required to undo it. We have already guessed v_{del} by its adjacency to a. Further, v_{del} is adjacent to p_2 and v_x; and b is adjacent to p_2. Hence we can guess the flip from $(d-1)(d-2)(d-3)$ possibilities.
- If $p'_i = a'$, this is a new bubble. Let $i' = i + 1$ and we seek to guess $a' = p'_2$ next iteration.
- If $p'_{i+1} = d$, this flip swaps the bubble with d. The new bubble is a', i.e. $i' = 1$, and we seek to guess $d' = p'_i$ in the next iteration.
- If none of the above obtain, we move the bubble closer to d: $i' = i + 1$.

2.2 Disconnected graphs

2.2.1 Mapping between Ω_S and Ω_F

In order to allow the long-flip to operate over the same space as switches, i.e. potentially-disconnected graphs, we now define the mapping $\mathcal{H} : \Omega_S \to \Omega_F$ in a manner similar to [8]. Let the connected components of graph G_S be $H_1, \ldots, H_k$, ordered by the largest vertex number in each component. We refer to this largest

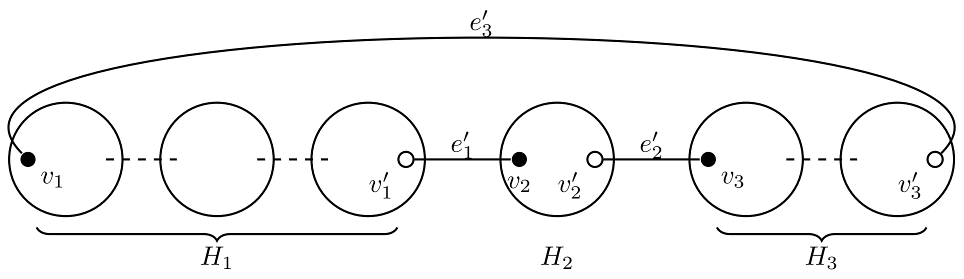

Figure 4: A connected graph representing a disconnected graph with three components. The black vertices are the "entry" vertices, i.e. the v_i's; and the white are the "exit" vertices, the v'_i's. Here H_1 (similarly, H_3) is split into three components because $e_1 = v_1v'_1$ lay on a cut cycle in H_1. When e_1 was switched away, the other cycle edges (shown dashed on the diagram) became cut edges. Note that these are not chain edges. In the above diagram, e'_3 is the loopback edge and $v_1 < v_2 < v_3$.

vertex of H_i as v_i, and nominate a "bridge edge" e_i adjacent to it. The other endpoint is arbitrary, and is denoted v'_i. A graph $G_S \in \Omega_S$ is then mapped onto graph $G_F \in \Omega_F$ by switching away the bridge edges and switching in the "chain edges" linking consecutive H_i's. More precisely, for $1 \leq i < k$, the bridge edge $e_i = v_iv'_i$ is replaced by the chain edge $e'_i = v'_iv_{i+1}$. To complete the chain a "loopback" edge $e'_k = v'_kv_1$ connects H_k and H_1. Note that this is the only chain edge whose direction might go from a greater to a smaller vertex. The resulting graph is like that shown in Figure 4.

Lemma 2. *G is connected iff $\mathcal{H}(G) = G$.*

Proof. If G is connected graph, $v_1v'_1$ is both a bridge and a chain edge, and hence $\mathcal{H}(G) = G$. Conversely, if G is not connected, $\mathcal{H}(G)$ switches at least two edges of G into different positions. Clearly in this case $\mathcal{H}(G) \neq G$. $\square$

Lemma 3. *If $G \in \Omega_S$, $\mathcal{H}(G)$ is connected and d-regular, and hence $\mathcal{H}(\Omega_S) = \Omega_F$.*

Proof. If G is connected then by Lemma 2 $\mathcal{H}(G)$ is connected. Otherwise, each $H_i \in G$ is biconnected, since it is at connected and, by Lemma 1(i), is even-connected. Hence the components cannot be disconnected by the removal of a bridge edge. The chain edges provide a path from any component to any other component, so $\mathcal{H}$(G) is connected. Further, each vertex affected by the removal of a bridge edge is the endpoint of a chain edge, so H does not alter vertex degrees, and hence is regularity-preserving. Together these imply biconnectivity of $\mathcal{H}(G)$. $\square$

Given the loopback edge e_k, the ordering of the chain allows us to recover the original G_S from a given G_F, i.e. to compute $\mathcal{H}^{-1}$. To identify e_1 given e_k, search the graph $G_F \backslash e_k$, starting from v_1, for a vertex numbered greater than v_1. This yields v_2 due to the fact that every other vertex in H_1 is less than v_1 by construction. The edge traversed to reach v_1 must be e'_1, as we removed the only other chain edge adjacent on H_1. Iterating this approach we can recover the identities of the remaining chain edges, and hence uniquely recover G_S, given only the loopback edge. That knowledge of the loopback edge is necessary follows from the fact that a graph G_F might contain edges indistinguishable from chain edges, as illustrated by the dashed edges in Figure 4. Here, the search for v_2 traverses two cut edges before arriving at the chain edge e'_1.

2.2.2 Recovering $G_S \in \Omega_S$ in the long-flip

Again, let $X_F \in \Omega_F$ denote the current graph at some point during a long-flip, and recall that we must specify the flip producing X_{F+1}. If Z_R was not connected (i.e. $\mathcal{H}(Z_R) \neq Z_R$) we must determine the components H_i in order to make progress, and hence we require the loopback edge. We could simply guess its identity from nd possibilities, but the following outlines a slightly tighter method.

The highest-numbered vertex in X_F must also be $v_k = n$, and so a chain edge is adjacent upon it. Delete this vertex and let C be the set of cut edges of the resulting graph X'_F. Note that C is computable in $\mathrm{O}(nd)$

time [15], and comprises the candidate loopback edges. Acquiring a bound for $|C|$ will give us a bound for the number of candidates for the loopback edge, and we concern ourselves with this now.

Let $T = (S, C)$, where S is a set of supervertices (henceforth, "nodes") formed by identifying all vertices in each (edge-)biconnected component. Graph T is a forest, as cycles would imply biconnected nodes and no edge in C is a loop in T (or that edge wouldn't be a cut edge in X'_F). The number of nodes with degree other than 2 can be bounded as below.

1. *T has at most $r_{\text{leaf}} \leq d$ leaves.* Let $T^* = (S \cup \{n\}, C \cup N_{X_F}(n))$, i.e. T with v_k and its neighbouring edges added. By Lemma 1(i), each $s \in S \cup \{n\}$ has even degree in T^*, since otherwise we could obtain an odd cut in X_F. Now v_k has neighbours in at most d distinct nodes of T^*, so the removal of v_k results in at most d nodes of odd degree. Leaves form a subset of such nodes, and hence there are no more than d leaves.

2. *There are $r_{\text{branch}} \leq d-2$ branch nodes of T (i.e. nodes having degree greater than 2).* Consider the set $\{S_i\}_{1\leq i\leq q}$ of connected components of T. Let $l(S_i)$ be the number of leaves in each S_i. The property follows by induction on $l = l(S_i)$. Clearly when l is 1 or 2, there can be no branches; and adding exactly one leaf increases the number of branches by at most one (specifically when the leaf is added to a node of degree 2).

Let $p \geq r = r_{\text{leaf}} + r_{\text{branch}}$ be the number of vertices of X'_F contained in the r leaves or branch nodes. Remove these from T, leaving $n - p - 1$ vertices in the remaining nodes. Each node must contain at least $d + 1$ vertices by d-regularity. (The worst case is $K_{d+1} \setminus e$.) Given that none of the edges adjacent to vertex v_k can be the loopback edge, we need not add it back to X'_F, and our bound for $|C|$ is:

$$\left\lfloor \frac{n-p}{d+1} \right\rfloor + p \;\leq\; \left\lfloor \frac{n-r}{d+1} \right\rfloor + r \;\leq\; \left\lfloor \frac{n-2d+2}{d+1} \right\rfloor + 2d - 2 \;\leq\; \frac{n}{d}, \quad \text{if } n \;\geq\; 2d^2(d-1).$$

Hence the choice of the loopback edge only incurs a penalty n/d in the mixing time. This bound is tight up to an $\mathrm{O}(d)$ term since, if $(d+1)\,|\,n$, we may chain together $n/(d+1)$ copies of K_{d+1}.

2.2.3 Simulating any switch

We now extend the connected long-flip to the disconnected case. It might be that $\mathcal{M}_S$ specifies a switch with bridge edge e_i as one of its switch edges (indeed both switch edges could also be bridge edges). Since $\mathcal{H}(Z_R)$ has had its bridge edges switched away for chain edges, we must perform a *bridge change* to replace the bridge before the algorithm can continue. To do so, choose $v^* \in N(v_i)\backslash v'_i$ in some deterministic way, e.g. the lowest valid index. The long-flip $(v^*_i, v_i, v'_i, v_{i+1})$ performs the appropriate edge changes, and because the graph is connected a long-flip can simulate it. The long-flip edges in the bridge change cannot be bridge edges themselves, so this operation is straightforward.

Depending on which components of Z_R contain the switch edges, there are two ways a switch can alter the number of components. If the switch edges lie in distinct components, those components are reconnected. If they lie in the same component, form a 2-cut and the switch doesn't reconnect them, a disconnection results and there is no path p to perform a long-flip. Otherwise there is no change to the number of components, and the switch is directly simulated by a single long-flip.

To handle disconnections, suppose H_i is the component containing the 2-cut that would cause the disconnection, and name the two resulting components B_1 and B_2. If the bridge edge is part of the 2-cut perform a bridge change, and without loss of generality let B_1 contain the bridge edge $e_i = v_i v'_i$.

Then our approach is to perform a "housekeeping" long-flip to add B_2 as a chained component, before we break the 2-cut. Let $e_+ = v_+ v'_+$ be the bridge edge of B_2, where $v_j < v_+ < v_{j+1}$, so B_2 is "between" H_j and H_{j+1} in the chain. Then performing the long-flip $(v_+, v'_+, v_j, v'_{j+1})$ leaves the chain edges in their correct configuration, and all that remains is to perform the original, "master" long-flip.

Reconnections are handled in a similar way. Let H_i and H_j be the components affected, where $v_i < v_j$. Performing the specified long-flip first joins together these two components, and then the "housekeeping" long-flip $(v_i, v'_{i-1}, v_{i+1}, v'_i)$ removes H_i from the chain, maintaining the correct ordering of the components.

2.3 Determining the mixing time

It is convenient to use the mixing time of the underlying switch chain to bound the mixing time of the flip chain, and we do so in terms of the load of the flow, $\rho(f)$, and the maximum path length, $l(f)$. The latter is increased because it takes many flips to simulate a single switch. To see the former, fix a flip $t_F \in \mathcal{G}_F$ and suppose that the algorithm is forced to guess from β choices. The load on the switch that t_F is simulating is at most $\rho(f)$, and at most β switches can make use of t_F in the flip chain, so β is a multiplicative penalty on $\rho(f)$.

Let us consider the worst case such penalty, which occurs when the current flip is restoring a triangle for a switch that specifies a reconnection or disconnection on an already disconnected graph, and when the switch edges are also bridge edges. In this scenario, we guess from $\beta = 2n^4d^5$ possibilities, broken down as follows:

- At any stage during the algorithm, the path p must be reconstructed, costing a factor of 2 for deciding the bubble identity from the unoriented flip, and a factor of $n(n-1)(n-2)d^3 < n^3d^3$ to guess a, d, a', d', v_{del} and c.

- If the underlying switch graph is disconnected, then the components H_i must be deduced after guessing the loopback edge, for a cost of n/d.

- To determine the reversing flip $(b, p_2, v_{\mathrm{del}}, v_x)$, we guess neighbours of known vertices for a factor of $(d-1)^2(d-2) < d^3$.

During a reconnecting or disconnecting switch, the current flip might be part of a "housekeeping" long-flip, where a disconnection (resp. reconnection) requires housekeeping first (resp. last). At the end of each long-flip we can guess if there remains another switch to perform. Doing so, we have only a constant factor to pay, as guessing the vertices of the next switch is already paid for in the construction of p. Bridge changes are similarly inexpensive, as once it becomes clear that a bridge change is necessary, the exact long-flip required to perform it is well-determined.

Simulation of a switch in the switch chain may mandate four distinct long-flips: at most two bridge changes, and a housekeeping/master pair. The length of path p in each long-flip is bounded by n, and hence the number of flips required is at most $\frac{1}{2}n(n-1) < \frac{1}{2}n^2$. Thus we must pay a multiplicative factor of at most $2n^2$ in the path length $l(f)$. Combined, this results in the bound stated in Theorum 1 of $\mathrm{O}\big(n^{16}d^{22}(dn\log dn + \log \varepsilon^{-1})\big)$.

The simulation is responsible for a factor of $\mathrm{O}(n^6d^5)$. The direct two-stage construction leads to a much tighter bound than the comparison approach of Feder et al.[8], which adds a factor of $\mathrm{O}(d^{41}n^{45})$ to the switch chain mixing time.

It is unlikely that the bound thus acquired is tight, and finding such a bound remains an open problem. Experimental evidence seems to suggest that $\mathrm{O}(n \log n)$ is a reasonable estimate.

3 A protocol based on randomising flips

The SWAN model of [2, 3] maintains its desirable properties though its joining scheme, in which newcoming peers are clothespinned onto edges sampled at random from the graph. In this section we define and analyse a protocol based around the idea of random flips occuring spontaneously throughout the network. As shown in the previous section these flips will tend to make the network topology random regular. If enough flips occur between joins then the network can countenance even the simplest joining scheme.

A problem that all networks must solve is making themselves accessible to joining peers. As with Gnutella, in the flip network all peers can serve as a "gateway", which will negotiate the incorporation of a newcomer into the network. Acquiring the addresses of peers inside the network is a problem for the implementation and the organisation: a solution could be as simple as an address on a website, or something similar to the Bittorrent trackers.

Join A newcomer contacts a peer already in the network. This gateway samples $d/2$ independent edges from its immediate neighbourhood using a greedy breadth-first search. It instructs the peers matched by these edges to disconnect from their matched neighbours and to connect to the newcomer—an operation referred to as the "clothespin". Note that this preserves regularity and that the gateway itself will be a neighbour of the joining peer.

Part The parting peer selects independent pairs of its neighbours and instructs paired peers to connect to one another, leaving it free to disconnect without disrupting regularity. This is the reverse clothespin operation, and in general there is more than one pairing to choose from. It is also possible that no such independent pairing exists, although we show below that this rarely occurs. However, if it does occur then one or more double-connections are formed, and the afflicted peers are given the responsibility of re-simplifying the network by performing flips.

Active randomisation In order to counteract the de-randomising effect of the joins, each peer will occasionally spontaneously initiate flips (and will take part in flips initiated by others). A flipping peer will choose one of its neighbours at random, which will choose one of its neighbours, and so on until a 3-path is found. A simple locking mechanism prevents two flips from sharing a vertex, which was found to be important in preventing disconnections in [16]. Note that if the mixing time of the flip chain is in fact $\mathrm{O}(n \log n)$ then an individual peer must flip only $\mathrm{O}(\log n)$ times per unit time.

3.1 Analysis of parting

Joining and spontaneous flips can always be performed as long as the network is regular. Further, when correctly carried out they bear no risk of disconnecting the network. Here we take a closer look at the parting operation under the simplifying assumption that it occurs on a random regular graph. As stated this will not generally be the case, but we argue that it is a decent approximation.

First we state two useful lemmas, the first of which is from the text of Janson et al. [11] § 9.2:

Lemma 4. *A small subgraph G with v_G vertices and e_G edges will occur in a random regular subgraph of size n with probability* $\mathrm{O}(n^{v_G} n^{-e_G})$.

Informally, this is because there are $(n)_d \sim n^{v_G}$ ways to select v_G vertices, and there is a $\mathrm{O}((d/n)^{e_G})$ probability of the given edges being present in the subgraph thus chosen. Note that this imples that "edge-dense" subgraphs are rare in random regular graphs.

Lemma 5. *Consider the graph G formed by removing the set of edges E from K_d, the complete graph. If $|E|$ is smaller than $d-1$, G has a perfect matching.*

(See appendix for proof.)

The reverse clothespin performed by a parting peer requires that its neighbourhood can be separated into independent pairs. Lemma 5 shows that at least $d-1$ edges are required to block a clean part. Here, the subgraph of interest is the d-star around the parting peer. By Lemma 4, with the additional blocking edges we would expect this configuration to arise with probability $\mathrm{O}\big(n^{d+1} n^{-2d+1}\big) = \mathrm{O}\big(n^{-d+2}\big)$, i.e. rarely for $d > 2$.

A concern is that parts without global information make it difficult to guarantee preservation of connectivity. Indeed, [17] argues that bounded-degree networks that treat connections identically cannot provide such a guarantee. While it is true that we cannot remove the possibility of disconnection on a part, we can make it arbitrarily small by exploring the immediate locale of a parting peer.

Figure 5 shows an example of a "v-precarious subgraph", i.e. a subgraph disconnected from the rest of the graph by the deletion of vertex v. If v were parting, we would need to be able to detect the precariousness of the situation, and be sure to pair vertices in the precarious subgraph with the remainder of the graph. We show that by exploring the graph to a constant depth we can detect an arbitrarily large proportion of these rare cases. A corollory of the lemma is that large clusters are extremely unlikely to splinter off from the giant component, so even in a disconnection the majority will see no degredation in service quality.

Fix some vertex v and a constant depth c. A v-precarious subgraph is "recognised" when vertex v can detect it by exploring its neighbourhood to maximum depth c. Let H be the subgraph of a random d-regular graph formed by such an exploration. If H contains a precarious subgraph then it is recognisable as an induced subgraph where every vertex except v is d-regular.

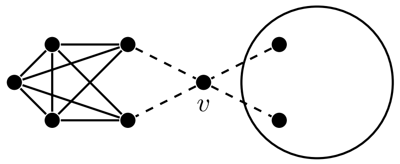

Figure 5: The smallest precarious subgraph for $d = 4$. Vertex v is parting, and the remainder of the graph is in the circle to the right.

Lemma 6. *The probability of there being a precarious subgraph that we cannot recognise is* $\mathrm{O}(n^{-c})$. *Hence we can avoid disconnection with reasonable values of* c *a.a.s.*

Proof. For simplicity let us first examine a precarious subgraph R in the case where $d = 4$. Due to Lemma 1(i) we can group v's edges into edge-pairs, here with one pair going into R and one pair into the rest of G.

A precarious subgraph must be of size at least $d + 1$. This follows because smaller non-null complete graphs require more than $d - 2$ edges incident upon them to satisfy regularity, v can only provide $d - 2$, and by definition no other vertex is adjacent to it. Recognising such a subgraph would occur on step $c = 2$. Were a precarious subgraph to be more difficult to recognise in the sense that it requires a greater c, clearly it would have to be at least 1 vertex larger. Iterating gives the result that size r of a precarious subgraph discovered on step $c \geq 2$ is at least $d + c - 1$.

Precarious subgraph R of size r requires $dr/2$ edges to satisfy regularity. For our simpler case of $d = 4$ we examine the probability of $R^* = G[R \cup \{v\}]$ being an unrecognised precarious subgraph of G. Here, $v_{R^*} = r + 1 \geq d + c$, and $e_{R^*} = \frac{dr}{2} + 1$ (where that final $+1$ is due to the edge-pair from v). By Lemma 4 we would expect this to occur with probability

$$\mathrm{O}(n^{r+1-\frac{dr}{2}-1}) = \mathrm{O}(n^{-\frac{1}{2}(d-2)r}).$$

Plugging in the fact that $r \geq d + c$ tells us that the probability of failing to recognise a given precarious subgraph because c was too small is at most $\mathrm{O}\big(n^{-\frac{1}{2}(d-2)c-\mathrm{O}(d^2)}\big) \subset \mathrm{O}(n^{-c})$ (since $d > 2$).

We finish by showing how the above result can be extended to arbitrary d. The only complications are that there could be multiple precarious subgraphs on v, or that several of v's edge-pairs go into one precarious subgraph. The former contains the above as a special case, so the bound is at most that of the largest precarious subgraph. The latter has the effect of adding an edge to an R^*, increasing its density and making it even less probable.

□

A quick note on peers maintaining knowledge of their neighbourhood to depth c: if $c = 2$ then the neighbours of a failed peer know enough to very quickly patch the network up to regularity. This still places no more than a constant demand upon each peer, but solves the problem of (non-compound) failures.

3.2 A worst-case network

The flip network protocol does not result in a pure random regular graph, as under normal operating conditions there will likely be recently-joined peers that are in some sense still randomising. For this reason we examine the functioning of the protocol using simulations, and measure the evolving topology against some of the metrics that first attracted us to random regular graphs.

In a real network we would expect newcomers to have access to a number of potential gateways. Some Gnutella applications solve this bootstrapping problem with a cache of well-known, reliable peers, and this list is kept fresh with new peers the application comes across in general usage. However, we find it convenient to study the "worst case" of a network with a single published gateway. Each joining peer clothespins with edges from the immediate neighbourhood of the gateway, so the only force preventing this from resulting in a very poor topology is the randomising action of the flips, allowing us to view its effects more clearly.

We use the model of joins and parts used in [17] (and extended in [3]). Let inter-join delays and residual service times be exponentially distributed with parameters ν and μ. Here we normalise $\nu = 1$ time unit for simplicity. This model results in the network size n tending towards $N = 1/\mu$ by encouraging joins when

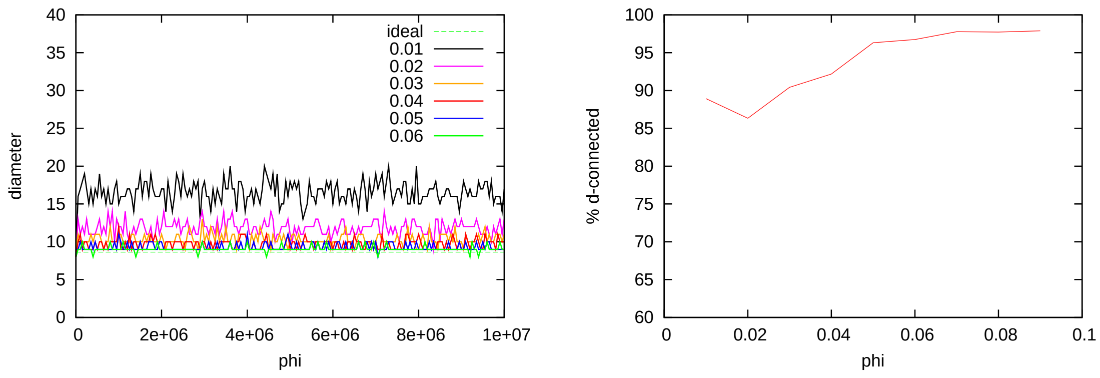

Figure 6: Traces of properties against graphs with varying ϕ. **Left:** diameters observed over the evolution of a network with expected size $N = 1000$, $d = 4$, and initial topology random regular. Note the convergance of characteristic diameter to ideal as ϕ increases. **Right:** the proportion of time a network has full d-connectivity (averaged over several runs).

$n < N$, and parts when $n > N$. Our model adds the flip rate parameter, ϕ, which denotes the probability that a given peer will initiate a flip in 1 time unit.

Clearly we cannot test a given graph for "randomness", but we can test for the properties that made random regular graphs interesting in the first place. The simulator tracks the following quantities:

- The diameter of the network, as this governs the worst-case quality-of-service (QoS) and has been well characterised in random regular graphs [23].
- Average shortest path, as this reflects the average QoS; and also because the still-randomising neighbourhood of the gateway peer will have less effect on this quantity.
- Minimum vertex-connectivity, calculated as the maximum flow between pairs of vertices, where both edges and vertices have capacity 1.

We have observed that, for a given flip-rate ϕ, the diameter of the network settles around some characteristic diameter greater than or equal to the ideal Bollobás (configuration model) diameter [11] (see Figure 6). As ϕ increases, this characteristic diameter tends closer to the ideal. It becomes indistinguishable at $\phi = 0.05$ when $N = 1000$, which implies around 50 flips between joins. Similarly, the chance of a network having full (d) vertex-connectivity climbs with ϕ, and is almost always ($> 95\%$) d-connected at $\phi = 0.05$.

A possible explanation for the characteristic diameter is in recently joined peers that have not yet migrated to a random place in the network. This would imply the network resembles a random regular majority, with a "spire" of insufficiently randomised newcomers. The average shortest path lengths are further evidence of this, as each network shows similar "characteristic" behaviour for a given ϕ, except the value observed is much closer to the ideal average shortest path length. This is to be expected if a minority of the peers are in the spire, as they contribute less to the average path length calculation than the diameter calculation.

References

[1] Bittorrent. `http://www.bittorrent.org`.

[2] V. Bourassa and F. Holt. Swan: Small-world wide area networks. In *Proceedings of International Conference on Advances in Infrastructure (SSGRR 2003w)*, L'Aquila, Italy, 2003. Paper #64.

[3] C. Cooper, M. Dyer, and C. Greenhill. Sampling regular graphs and a peer-to-peer network. *Comb. Probab. Comput.*, 16(4):557–593, 2007.

[4] C. Cooper, R. Klasing, and T. Radzik. A randomized algorithm for the joining protocol in dynamic distributed networks. *Theoretical Computer Science*, 406(3):248–262, October 2008.

[5] P. Diaconis and L. Saloff-Coste. Comparison theorems for reversible Markov chains. *Annals of Applied Probability*, 3:696–730, 1993.

[6] P. Diaconis and D. Stroock. Geometric bounds for eigenvalues of Markov chains. *Annals of Applied Probability*, 1:36–61, 1991.

[7] M. Dyer, L. Goldberg, M. Jerrum, and R. Martin. Markov chain comparison. *Probability Surveys*, 3:89–111, 2006.

[8] T. Feder, A. Guetz, M. Mihail, and A. Saberi. A local switch Markov chain on given degree graphs with application in connectivity of peer-to-peer networks. *FOCS*, 2006.

[9] C. P. Fry and M. K. Reiter. Really truly trackerless bittorrent. Technical report, Carnegie Mellon University, 2006.

[10] Gnutella. `http://en.wikipedia.org/wiki/Gnutella`.

[11] S. Janson, T. Łuczak, and A. Ruciński. *Random Graphs*. Wiley, 2000.

[12] M. Jerrum. *Counting, Sampling and Integrating: Algorithms and Complexity*. Birkhäuser, 2003.

[13] M. Jerrum and A. Sinclair. Approximating the permanent. *SIAM J. Comput.*, 18(6):1149–1178, December 1989.

[14] R. Kannan, P. Tetali, and S. Vempala. Simple markov-chain algorithms for generating bipartite graphs and tournaments. *Random Struct. Algorithms*, 14(4):293–308, 1999.

[15] K. Madhukar, Pavan D. Kumar, Pandu C. Rangan, and R. Sundar. Systematic design of an algorithm for biconnected components. *Science of Computer Programming*, 25(1):63–77, 1995.

[16] P. Mahlmann and C. Schindelhauer. Peer-to-peer networks based on random transformations of connected regular undirected graphs. In *SPAA '05: Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 155–164, New York, NY, USA, 2005. ACM.

[17] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter peer-to-peer networks. *Selected Areas in Communications, IEEE Journal on*, 21(6):995–1002, 2003.

[18] D. Randall and P. Tetali. Analyzing Glauber dynamics by comparison of Markov chains. *Journal of Mathematical Physics*, 41(3):1598–1615, 2000.

[19] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, volume 31, pages 161–172. ACM Press, October 2001.

[20] S. Saroiu, P. Krishna Gummadi, and S. D. Gribble. Measuring and analyzing the characteristics of napster and gnutella hosts. *Multimedia Systems Journal*, 8(5), November 2002.

[21] A. Sinclair. Improved bounds for mixing rates of Markov chains and multicommodity flow. *Combinatorics, Probability and Computing*, 1:351–370, 1992.

[22] I. Stoica, R. Morris, D. Karger, F. M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, volume 31, pages 149–160, New York, NY, USA, October 2001. ACM Press.

[23] N. Wormald and V. Sanwalani. The diameter of random regular graphs. Incomplete paper.

[24] N. C. Wormald. Models of random regular graphs. In J.D. Lamb and D.A. Preece, editors, *Survey in Combinatorics*, London Mathematical Society Lecture Notes **276**. Cambridge University Press, Cambridge, 1999.

Appendix A: the switch and flip Markov chains

```
From $G \in \Omega_S$ do
    with probability $\frac{1}{2}$ do nothing;
    otherwise
        choose two non-adjacent distinct edges $ij$, $kl$, u.a.r.,
        choose a perfect matching $M$ of $\{i, j, k, l\}$ u.a.r.,
        if $M \cap E = \emptyset$ then
            delete the edges $ij$, $kl$ and add the edges of $M$,
        otherwise
            do nothing
        end if;
    end if;
end;
```

Figure 7: The switch Markov chain $\mathcal{M}_S$.

```
From $G \in \Omega_F$ do
    with probability $\frac{1}{2}$ do nothing;
    otherwise
        choose any 3-path $(i, j, k, l)$ u.a.r.,
        if $E \cap \{ik, jl\} = \emptyset$ then
            delete the edges $ij$, $kl$ and add the edges $ik, jl$,
        otherwise
            do nothing
        end if;
    end if;
end;
```

Figure 8: The flip Markov chain $\mathcal{M}_F$.

Appendix B: proofs of lemmas in section 3.1

Proof of Lemma 5. Removing $d-1$ or more edges can block a perfect matching by isolating a single vertex, preventing its being matched. We use Tutte's theorem **cite** to show that fewer edges are insufficient. Tutte's theorem states that a graph has a perfect matching iff $\forall U \subseteq V, |U| \geq o(G[V \backslash U])$, where $o(G)$ counts the number of connected components of odd size in G. We show that no choice of $|U|$ can fail to satisfy this property if $|E| < d-1$.

Fix some arbitrary $|U|$ and let u be its size. If U is to form a contradiction to G's having a perfect matching then $G' = G[V \backslash U]$ must have at least $u+1$ odd components. Because d is even, the number of vertices not in U shares u's parity. For even u, in order to achieve $u+1$ odd components, G' must contain at least $u+2$ odd components by parity. A similar argument shows the same for odd u, so a counterexample requires that G' has $u+2$ components.

Lemma 7 shows that the most edge-efficient method of producing (odd) components is to isolate a single vertex at a time. This will have to be done $u-1$ times to yield $u-2$ components. However, as with $u=0$, doing this even once requires $d-1$ edges, and the cost increases with u. Hence there is no counterexample to the Tutte requirement, and so G must have a perfect matching.

□

Lemma 7. *In partitioning the complete graph K_d into k connected components, a strategy minimising the set of edges deleted, E, is to isolate vertices one at a time until $k-1$ have been isolated from the larger component.*

Proof. Consider a minimal vertex partition $P = [a_1, a_2, \ldots, a_k]$. Set E contains only inter-component edges because removing intra-component edges does not affect the partition. Let m be a largest component, and j any other such that a_j is not equal to 1 (and hence minimal). If no such j can be found, halt. Otherwise, we have m and j such that $a_m \geq a_j$.

To separate these two components we had to add $a_m a_j$ edges to E. However, consider a partition P^* with $a^*_m = a_m + 1$, $a^*_j = a_j - 1$, and $a^*_i = a_i$ for all other indeces. Partition P^* grows m at the expense of j. The number of edges required to separate m and j is now $(a_m+1)(a_j-1) = a_m a_j + a_j - a_m - 1 \leq a_m a_j + a_m - a_m - 1 < a_m a_j$. Further, the number of edges required to separate the other components has not changed. To see this, choose any other component i. The number of edges removed from K_d to make i independent of m and j is $(a_m+1)a_i + (a_j-1)a_i = a_m a_i + a_j a_i$, which is the same as it was under partition P. Hence either P^* is smaller than P (in the above sense), or $P = P^*$ and at most one component contains more than one vertex. The lemma follows.

□