Routing Complexity of Faulty Networks

Omer Angel *
Department of Mathematics
University of British Columbia
Mathematics Road Vancouver, B.C. V6T 1Z2
Canada
angel@math.ubc.ca

Itai Benjamini
Department of Mathematics
The Weizmann Institute of Science
Rehovot 76100 Israel
itai.benjamini@weizmann.ac.il

Eran Ofek
Department of CS and applied Math.
The Weizmann Institute of Science
Rehovot 76100 Israel
eran.ofek@weizmann.ac.il

Udi Wieder
Department of CS and applied Math.
The Weizmann Institute of Science
Rehovot 76100 Israel
udi.wieder@weizmann.ac.il

ABSTRACT

One of the fundamental problems in distributed computing is how to efficiently perform routing in a faulty network in which each link fails with some probability. This paper investigates how big the failure probability can be, before the capability to efficiently find a path in the network is lost. Our main results show tight upper and lower bounds for the failure probability which permits routing, both for the hypercube and for the $d-$dimensional mesh. We use tools from percolation theory to show that in the $d-$dimensional mesh, once a giant component appears — efficient routing is possible. A different behavior is observed when the hypercube is considered. In the hypercube there is a range of failure probabilities in which short paths exist with high probability, yet finding them must involve querying essentially the entire network. Thus the routing complexity of the hypercube shows an asymptotic phase transition. The critical probability with respect to routing complexity lies in a different location then that of the critical probability with respect to connectivity. Finally we show that an oracle access to links (as opposed to local routing) may reduce significantly the complexity of the routing problem. We demonstrate this fact by providing tight upper and lower bounds for the complexity of routing in the random graph $G_{n,p}$.

Categories and Subject Descriptors: G.3 [Probability and Statistics]: Stochastic Processes; C.2.4 [Computer Communication Networks]: Distributed Systems

General Terms: Algorithms, Reliability, Theory

*Research done while at the Weizmann Institute.

PODC'05, July 17–20, 2005, Las Vegas, Nevada, USA.

Keywords: Routing, Fault Tolerance, Random Graphs, Percolation.

1. INTRODUCTION

The goal of this paper is to investigate the effectiveness of routing in faulty networks. Suppose that a network is represented by a graph G. Two kinds of fault models are common in the theoretical literature: Worst case faults and random faults which are our concern. In the random fault model it is assumed that each component of the network fails with some probability and independently of all other components. In this paper we consider edge failures so we assume each edge in G fails independently with some probability $q = 1-p$. An edge that fails is called *closed* and a surviving edge is called *open*. The graph induced by the open edges is denoted by G_p. One can ask what is the probability that two nodes u and v remain connected in G_p. This had been the focus of much research concerning the existence of *giant components* in such graphs, and the critical values of p for the existence of those, cf. [1, 30, 20, 2, 23]. But in many applications the fact that a path between u and v exists is not sufficient, one wants to be able to *find* the path in a distributed manner.

It is known that if the topology of a graph has some randomness, then the existence of short paths in a graph does not guarantee the ability of efficiently finding them. For instance a cycle with a random matching has a logarithmic diameter [6], yet paths connecting a given pair of nodes can not be found in less than $\sqrt{n}$ time [21]. This phenomenon is especially acute when considering 'natural' networks such as the world wide web, social networks, P2P networks etc, in which typically the network size is huge, the diameter of the network is small and the challenge is to find short paths within a time complexity that is comparable to the diameter. Indeed, Kleinberg's model of the small world phenomenon [21, 22] is aimed at explaining the ability to *find* short paths in social networks (and not merely their existence). In the context of P2P, several randomized topologies were proposed along with routing algorithms that find short paths in the random graph cf. [5, 15, 26]. Variations of these routing algorithms are able to find paths between nodes even when nodes or links fail cf. [18, 29, 32]. While our findings do not

apply directly to these networks, we expect that our main result regarding the hypercube would hold for them as well. See Section 1.3 for more details.

In this paper we analyze the algorithmic complexity of finding a path between nodes u, v in G_p as a function of the failure probability. In particular, we seek to find the exact values of p for which it is possible to perform routing in G_p within time complexity that is comparable to the diameter. One difficulty is that with positive probability u, v are in distinct components of G_p. We therefore restrict our attention to the case where a giant component exists, and condition on the event that u, v are connected.

Our findings present a complex picture. We show that for some graphs, as the $d-$dimensional mesh, efficient routing is always possible, i.e. it is easy to find with high probability short paths between nodes within the giant component (whenever it exists). However, for other graphs, such as the hypercube, efficient routing is possible only for some failure probabilities. In other words, there is a range of failure probabilities for which with high probability a giant component exists, the diameter of the giant component is small, yet in order to find a path between nodes it is necessary to probe a large portion of the graph. We provide tight upper and lower bounds on the routing complexity, indicating the exact location of the transition.

1.1 The Model

DEFINITION 1. *Given a graph G_p and two vertices u, v, a* routing algorithm *is an algorithm that is allowed to* probe *whether an edge exists in G_p, and outputs a path between u, v if such exists. A routing algorithm is said to be* local, *if the first edge it probes is adjacent to u and subsequently it probes only edges to (an end point of) which it has already established a path from u.*

Local algorithms aim to capture the realistic constraints of routing in a network. If each node is a server in a network and u wishes to send a message to v then u must find a path to v while probing edges it has already reached. In Section 5 we show that a local router may require *exponentially* more probes than a non local one, thus the distinction between the two kinds of algorithms is necessary. A non local routing algorithm may be referred to as an *oracle* routing algorithm. Denote by $\{u \sim v\}$ the event that u is indeed connected to v.

DEFINITION 2. *Given a graph G, probability p and a routing algorithm A, the* routing complexity *of A denoted by $comp(A)$, with respect to the nodes u, v, is the random variable that counts the queries A makes (i.e. edges probed) to find a path between u, v in G_p, conditioned on $\{u \sim v\}$.*

The routing complexity measures how many *probes* are needed to route a message from u to v in G_p, assuming this routing is possible. We do not consider here any computations that the algorithm performs. As indicated above, the question is most interesting when $\Pr[u \sim v]$ is bounded away from zero, and indeed we limit our discussion to this case. A simple upperbound on the routing complexity could be achieved by performing a BFS search on G_p. In terms of the routing complexity this is tantamount to probing the entire graph. However there may exist algorithms which achieve a much smaller routing complexity. In particular, if the diameter of G is small, we are interested in finding a routing algorithm with a complexity that is comparable to the *actual distance* between the nodes, or show that none exists.

We stress that the routing complexity measures the complexity of finding a path between two specified vertices, and not the complexity of finding a full blown routing scheme between all nodes. In this sense small routing complexity may be seen as the minimal requirement of fault tolerance in networks. We also assume that nodes have no prior knowledge on the faults. In reality, nodes may have some information on the status of links in the network, making the task of finding a path easier. In some sense we model the problem of finding a path 'from scratch'.

In this paper we focus on analyzing the hypercube and the $d-$dimensional mesh, which are probably the most widely investigated topologies in this context.

1.2 Related Work

Denote by $H_{n,p}$ the $n-$dimensional hypercube, when each edge is deleted with probability $1-p$ and survives with probability p. Random subgraphs of the hypercube had been the focus of much research. It is known (see eg. [11]) that if $p < \frac{1}{2}$ then with high probability $H_{n,p}$ is not connected and if $p > \frac{1}{2}$ then with high probability $H_{n,p}$ is connected. A classic result by Ajtai, Komlos and Szemeredi [1] states that if $p \geq n^{-1}(1+\varepsilon)$ for any fixed $\varepsilon > 0$ then with high probability[1] $H_{n,p}$ contains a giant component (i.e. a component with $\Theta(2^n)$ nodes), while if $\varepsilon < 0$ then w.h.p a giant component will not exist. This result was sharpened by Bollobás *et al* in [7] and then by Borgs *et al* in [8].

A related notion to routing complexity is that of *emulation*. Roughly speaking, network A emulates network B if A can perform *any* computation B performs with a constant slowdown. When the emulating network is a random subgraph the notion of emulation implies not only that short paths could be found but also that they do not create bottlenecks in the computation. Hastad *et al* [16, 17] considered *node* failures, and showed that if p is a constant close enough to 1, then $H_{n,p}$ could *emulate* H_n with a small slowdown. Cole *et al* [10] proved that a faulty butterfly network can perform efficient permutation routing even if each node or edge fails with some constant probability. Emulation under worst case faults were considered by Leighton *et al* [24]. In particular these results imply that if the failure probability is small enough then it is possible to find paths efficiently between nodes in the giant component. On the other hand Angel and Benjamini [3] showed that if $p < \frac{1}{\sqrt{n}}$ then the hypercube could not be embedded in its giant component with constant distortion. This result suggests that for $\frac{1}{n} < p < \frac{1}{\sqrt{n}}$ even though a giant component exists, w.h.p it defines a metric that is (significantly) different from that of the hypercube.

Let M_p^d be a d-dimensional mesh with M^d nodes, in which each edge is deleted with probability $1-p$. It is known that for each d there exists a critical probability p_c^d such that if $p < p_c^d$ then w.h.p. there will not be a giant component in M_p^d, and if $p > p_c^d$ then w.h.p. M_p^d will contain a giant component. The exact values of the critical probabilities are not always known. It is known that $p_c^2 = \frac{1}{2}$ and that

[1]Throughout this paper, the term 'with high probability' means with probability that tends to 1 as $n \to \infty$.

$p_c^d = (1 + o(1))/2d$ and is decreasing in d. See the book by Grimmett [12] and the references therein. Kaklamanis *et al* [19] showed that if p is large enough then M_p^2 can emulate M^d with $O(\log n)$ slowdown. Mathies [27] extended this result for any $p > p_c^2 = \frac{1}{2}$. These results do not imply an efficient routing algorithm. Naor and Wieder [28] used planar duality to prove that efficient routing is possible in M_p^2 whenever $p > \frac{1}{2}$. Cole *et al* [9] proved that a two dimensional array can tolerate a constant fraction of worst case faults and still emulate the non faulty array with a constant slowdown.

1.3 Summary of Results

Recall that $H_{n,p}$ denotes a random subgraph of the n-dimensional hypercube obtained by selecting each edge independently with probability p. As mentioned, it is known that when $p > (1+\varepsilon)n^{-1}$ with high probability a giant component exists ([1]). Furthermore, it is implicit in the proof that the diameter of the component is polynomial in n. If however $p \leq n^{-1}(1-\varepsilon)$ then the size of the largest connected component is $o(2^n)$ w.h.p. This suggests that efficient routing in the giant component of $H_{n,p}$ might be possible for any $p \geq n^{-1}(1+\varepsilon)$. Our work shows that this is *not* the case: there is a threshold for efficient routing that lies in a *different* location. The following theorem provides a complete characterization of the routing complexity as a function of the failure probability:

THEOREM 3. *For a fixed α, let $p = n^{-\alpha}$.*

(*i*) *Let $\alpha > 1/2 + \beta$ for $\beta > 0$. For every pair of vertices u and v any local routing algorithm in the hypercube $H_{n,p}$ makes at least $2^{\Omega(n^{\beta})}$ queries w.h.p..*

(*ii*) *There is a local routing algorithm A on the hypercube such that the following holds: For any $\alpha < 1/2$ there exists $k = k(\alpha)$ so that for any two vertices, $comp(A) < n^k$ with probability $1 - \exp(-cn^{1-\alpha})$ for some constant $c > 0$.*

Thus for $p = n^{-\alpha}$ for $\frac{1}{2} < \alpha < 1$, an intriguing phenomenon occurs: the giant component of $H_{n,p}$ shares some structural properties of H_n, in particular it has diameter $poly(n)$ (w.h.p.), and has roughly the same expansion of H_n, yet the ability to find short paths is lost. Angel and Benjamini proved in [3] that for these failure probabilities H_n could not be embedded in $H_{n,p}$ with constant distortion, so the result of Part (*i*) is not entirely surprising, yet the techniques we use are different then that of [3]. Part (*i*) is proven by showing that if p is small enough then a ball centered at v is likely to look more or less like a tree rooted at v, which contains closed edges. Now, in order to reach v from u it is necessary to find a leaf which is connected to v via an open path, an event which is proven to be rare. Our technique is general enough to be used on other families of graphs.

It is proven in [3] that if $\alpha < 1/2$ then there is an embedding of H_n in $H_{n,p}$ with constant distortion. This embedding is used to derive the matching upperbound of part (*ii*). Note that the algorithm of (*ii*) does not depend on α, and only its efficiency changes. Therefore if $\alpha = 0$, i.e. there are no faults, then the algorithm reduces to a greedy algorithm which routes along the hypercube's shortest paths.

Many popular P2P topologies share some structural similarities with the hypercube cf. [32, 5, 31]. We did not prove that Theorem 3 holds for these topologies, yet it is reasonable to assume that that this is the case. If so, then Theorem 3 implies that if the network suffers many faults, flooding and gossiping techniques would remain efficient means to locate data (in terms of latency) while the routing algorithms based on exact search fail.

The phenomenon described in Theorem 3 does not occur in all graphs. Recall that M_p^d denotes a d-dimensional mesh with M^d nodes, in which each edge is deleted with probability $1 - p$.

THEOREM 4. *Let u, v be two vertices at distance n in M^d. There exists a local routing algorithm in M_p^d so that if $p > p_c^d$ then the expected routing complexity is $O(n)$.*

Thus in the mesh when p is large enough so that a giant component appears it is possible to find paths between two vertices in time comparable to the distance between the nodes. It is important to note that if we allow p to be close enough to 1 then Theorem 4 is fairly easy prove. The main difficulty is proving that the Theorem's statement is correct for *any* $p > p_c^d$, this involves some deep results from Percolation Theory.

The previous two theorems assumed the routing algorithms are local, i.e. they are only allowed to probe edges for which they have already established a path. What if we remove the locality assumption and allow the routing algorithm to probe *any* edge in the graph? we call this model *oracle* routing. On first glance it might seem as if oracle routing may not change considerably the routing complexity. Yet, in Section 5 we show a graph in which there is an exponential gap between the routing complexity with respect to oracle routing and that of local routing. We also provide tight upper and lower bounds for routing in $G_{n,p}$ and show that in this natural model, oracle routing outperforms local routing.

In the next section a lemma which provides a lower bound for routing complexity in a general scenario is proved. In Sections 3 and 4 we prove our results for the hypercube and the mesh respectively. Section 5 concerns the oracle routing model. Finally Section 6 discusses some related open problems.

2. THE LOWER BOUND LEMMA

In this Section we prove a lemma which is instrumental in proving hardness of local routing on various graphs. The basic intuition could be seen through the following example: Consider a graph in which there are exactly d edge disjoint paths of length 2 between nodes u, v. Now assume each edge remains open with probability $\frac{1}{\sqrt{d}}$. We expect that both u and v would be connected to about $\sqrt{d}$ open edges, thus by the birthday paradox w.h.p there would be an open path of length 2 between the nodes. Assuming such a path exists, it is easy to see that $\Omega(d)$ edges should be probed w.h.p before one of these paths is found.

This intuition could be generalized as follows: If S is a subset of the nodes, and $v \in S$ while $u \notin S$, then a path from u to v must at some point find an edge in the cut $(S, \bar{S})$ which is connected to v. If the probability that an edge in the cut is connected to v via the set S is low enough, then many such edges should be probed before a path is found.

More formally: for a set S and vertices $u \in V, v \in S$ we write $\{(u \sim v) \in S\}$ for the event that u is connected to v by an open path in the set S. Similarly $\{(u \sim e) \in S\}$ denotes the event that u is connected to an end point of the edge e via a path in the set S.

LEMMA 5. *Let $V = S \cup \bar{S}$ be a partition of the vertex set of a graph and $v \in S$ be a vertex. Assume for any edge e crossing the cut $(S, \bar{S})$ we have $\Pr[(v \sim e) \in S] \leq \eta$, and let X be the number of queries made by a local routing algorithm from u to v (given that $u \sim v$), then*

$$\Pr[X < t] \leq \frac{t\eta + \Pr[(u \sim v) \in S]}{\Pr[(u \sim v)]}. \quad (1)$$

If $u \in \bar{S}$ then the numerator becomes $t\eta$.

We stress that u need not be in $\bar{S}$. The lemma applies also for the case that $u \in S$ and this will be used (e.g. the proof of Theorem 3).

PROOF. It is more convenient to let X denote the total number of queries made by the algorithm without it being conditioned on $(u \sim v)$ (as opposed to its definition in the lemma). The conditioning could be added (later) by using the simple inequality $\Pr[X < t \mid (u \sim v)] \leq \Pr[X < t] / \Pr[u \sim v]$. So it is sufficient to show that $\Pr[X < t] \leq t\eta + \Pr[u \sim v \in S]$. First we use the following inequality:

$$\Pr[X < t] \leq \Pr[(X < t) \cap \overline{\{(u \sim v) \in S\}}] + \Pr[(u \sim v) \in S]$$

It remains to show that $\Pr[(X < t) \cap \overline{((u \sim v) \in S)}] \leq \eta t$.

If indeed $\overline{\{(u \sim v) \in S\}}$, which is always the case if $u \in \bar{S}$, then all paths from u to v must cross the cut $(S, \overline{S})$. Thus an algorithm which outputs a path from u to v must find a path in S from some edge of the cut to v. A cut edge is called *successful* if indeed it is the origin of an open path to v. It is sufficient to bound the number of cut edges which should be queried by the algorithm, until a successful edge is found. By assumption, a priori, the probability of each edge to be successful is at most η. If the event of an edge being successful had been independent of other edges, then the process of finding a successful edge would have been geometric with parameter η, and this suffices to prove the lemma. Unfortunately, the event of an edge being successful depends upon the status of the edges the algorithm had already probed. Our goal is to show that these dependencies do not increase the probability of finding a successful edge.

To simplify the argument, we will change the model slightly. Denote by $G_p(S)$ the induced sub-graph formed by S. Assume that when the algorithm queries a cut edge e, it is also given for free all the edges and vertices in S which are connected to e via a path in S, i.e. the connected component of $G_p(S)$ which contains the end point of e. If e is connected to v in S (i.e. the edge e is successful), then the algorithm succeeded in finding a path and stops. In the case where $u \in S$, the algorithm is also given the connected component of u in $G_p(S)$, thus if $\overline{\{(u \sim v) \in S\}}$ then the path must first leave S and then return to it in similarity to the case in which $u \in \overline{S}$. It is easy to see that the new model may only *reduce* the query complexity of a local algorithm (the algorithm can simply ignore the extra edges given to it). Furthermore, without loss of generality we may assume now that a local algorithm queries cut edges only. Note that edges outside S are independent of the event of a cut edge being successful.

Let $e_1, e_2, ..., e_t$ be the sequence of the $(S, \bar{S})$ cut edges queried by the algorithm. The sequence $e_1, ..., e_{i-1}$ encapsulates exactly all the information that the algorithm has until the i-th query (given G_p). We will show that:

$$\Pr[e_i \text{ is connected to v via } S \mid e_1, ..., e_{i-1}] \leq \eta. \quad (2)$$

This implies that the query complexity of the algorithm is stochastically bounded by a geometric random variable with parameter η. In particular, $\Pr[X < t] \leq \sum_{i=1}^{t} \eta(1-\eta)^i \leq \eta t$ as needed.

Denote by C_j the vertices of the subgraph revealed when querying e_j. For every $j < i$ it holds that C_j is a connected component of $G_p(S)$, and that $v \notin C_j$.

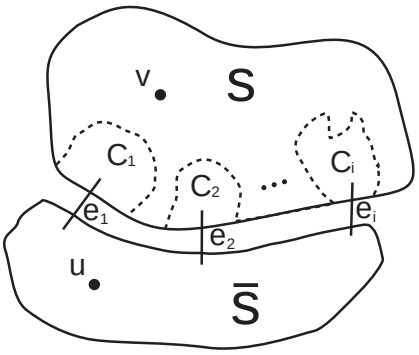

Figure 1: probing a cut edge e_i reveals its connected component in S (denoted by C_i).

Thus when the algorithm queries e_i there are two possibilities:

1. The end point of e_i belongs to a previously revealed C_j. Now the probability e_i is successful is zero.
2. The end point of e_i does not belong to any of the previously revealed C_j. Now

$$\Pr[(e_i \sim v) \in S | e_1, \ldots, e_{i-1}] =$$

$$\Pr[(e_i \sim v) \in S \setminus \cup_{j=1}^{i-1} C_j] \leq \Pr[(e_i \sim v) \in S] \leq \eta$$

□

Figure 1 demonstrates the argument in the case where $u \in \bar{S}$.

2.1 An Illustrative Example

In the following we use Lemma 5 to lower bound the routing complexity of the double binary tree. The double binary tree of depth n denoted TT_n is constructed by taking two binary trees of uniform depth n and identifying their corresponding leaves. Let u, v be the two roots of the trees, as depicted in Figure 2. First we identify the values of p (which is the probability that an edge is open) for which u, v are connected with probability which is bounded away from 0.

LEMMA 6. *If $\frac{1}{\sqrt{2}} < p \leq 1$ then there exists a path between u and v in $TT_{n,p}$ with probability bounded away from 0. If $p \leq \frac{1}{\sqrt{2}}$ then w.h.p. no such path exists.*

PROOF. In order for a path to exist between u and v, it must be the case that there exists an open branch from a leaf w to the root of the first tree u, and that the mirroring branch from w to v is also open. This is equivalent to the case of a single tree where each edge is open with probability p^2. It is well known that the critical probability of a Galton Watson tree (or the binary branching process) is $\frac{1}{2}$. See for instance [14] for details. □

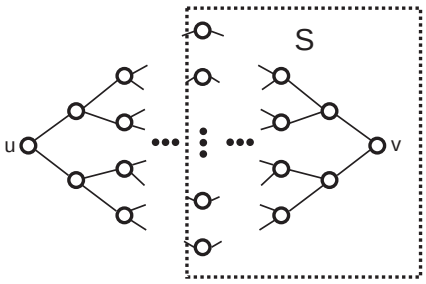

Figure 2: the double binary tree.

Now we show that for *any* $p < 1$, the local routing complexity is exponential in the graph diameter.

THEOREM 7. *Let* $\frac{1}{\sqrt{2}} < p < 1$. *For some* $c > 0$ *and any* a, n, *any local router between the two roots of* TT_n *makes at least* ap^{-n} *queries with probability at least* $1 - ca$.

PROOF. Apply Lemma 5 with S being the second tree to get the desired bound: Clearly we may have $\eta = p^n$. The nodes u and v can be connected only via the cut $(S, \bar{S})$, this happens with probability at least $c(p)$. Lemma 5 now implies $\Pr[A < ap^{-n}] < \frac{ap^{-n}p^n}{c(p)} = \frac{a}{c(p)}$ □

If we set a to be a decaying function (say $\frac{1}{n}$) then the probability a local router would probe less than $\frac{p^{-n}}{n}$ is $O(\frac{1}{n})$. The double binary tree has the interesting property that an *oracle* routing algorithm may find a path between u and v with a *polynomial* number of probes. See Section 5.

3. HYPERCUBE – TIGHT UPPER AND LOWER BOUNDS

In this section we show the exact location of the probability p, in which the routing complexity shows a phase transition between being exponential and being polynomial (in n). The idea is to show that when $p < \frac{1}{\sqrt{n}}$ then balls around nodes look more-or-less like trees, and therefore when trying to reach node v, a routing algorithm would need to 'penetrate' a tree through its leaves, as was demonstrated in the double binary tree graph. When $p > \frac{1}{\sqrt{n}}$ then there are enough edges so that some variant of greedy routing will find a path within polynomial time.

3.1 The Lower Bound

Here we apply Lemma 5 to get a lower bound on the local routing complexity of the hypercube when $p < n^{-1/2}$. The given bound translates to a fractional exponential (in n) bound on the routing complexity.

PROOF OF THEOREM 3(I). Let v and u be two vertices. We apply Lemma 5 to the hypercube with S being a ball of radius $l = n^\beta$ around v, for some $0 < \beta < \alpha - 1/2$.

The first stage is to bound η of the lemma, i.e. bound the probability v is connected to an edge on the boundary of S via a path within S . We show that for large n, for any e connecting S and $\bar{S}$ we have $\Pr[(v \sim e) \in S] \leq \eta$ holds with $\eta = 2n^{(\beta-\alpha)n^\beta}$. Let x be the endpoint of e in S with $d(x, v) = l$. Consider a path from v to x in S as a sequence of coordinates in which consecutive steps are taken. Let A_k be the set of such paths of length $l + 2k$ (by parity this catches all paths).

For $k = 0$ we have $|A_0| = l!$ since a path of A_0 uses each of the l coordinates exactly once. To bound $|A_k|$ we show a map from A_k to A_{k-1} that maps at most $n \cdot l^2$ paths to each path. Existence of such a map implies $|A_k| \leq nl^2|A_{k-1}|$ and therefore by induction $|A_k| \leq n^k l^{2k} l!$. To define the map, consider the first $l+1$ steps of a path. Since the path remains in the ball S, at least one of the coordinates is repeated. Take such a repeated coordinate and eliminate its first two occurrences. It is easy to see that this maps a path $p \in A_k$ to a path $p' \in A_{k-1}$. To reconstruct p from p' one needs to know which coordinate was removed (n possibilities) and the indices at which it appeared ($\binom{l+1}{2} \leq l^2$ possibilities). Thus the pre-image of p' contains at most nl^2 paths from A_k.

This bound clearly counts many paths more than once, as well as many non-simple paths, but it is good enough. Each simple path in A_k is open with probability p^{l+2k}, and so

$$\Pr[(v \sim x) \in S] \leq \sum_{k=0}^{\infty} p^{l+2k} n^k l^{2k} l! \leq$$

$$(lp)^l \sum_{k=0}^{\infty} (nl^2 p^2)^k = \frac{n^{(\beta-\alpha)n^\beta}}{1 - n^{2\beta+1-2\alpha}}.$$

For large n the denominator is close to 1, hence $\eta = 2n^{(\beta-\alpha)n^\beta}$ is a valid choice.

Next, we estimate the other terms in (1). Since each of u and v is in the giant component with probability tending to 1, $\Pr[(u \sim v)] \to 1$. If $u \notin S$ then $\Pr[(u \sim v) \in S] = 0$. Otherwise, suppose $d(u, v) = m \leq l$. The same argument as above shows that the number of paths in S of length $m + 2k$ from u to v is at most $m!(nl^2)^k$ and hence the probability that any of them are open satisfies

$$\Pr[(u \sim v) \in S] \leq \sum_{k=0}^{\infty} p^{m+2k} n^k l^{2k} m! = \frac{m! p^m}{1 - n^{2\beta+1-2\alpha}}.$$

The denominator tends to 1 and for $m \leq l$, the numerator is $o(1)$ because $mp \leq lp = n^{\beta-\alpha}$.

Using Lemma 5, we now see that if the complexity of a local router in the hypercube is A, then

$$\Pr[A < n^{(\alpha-\beta)n^\beta}/n] \leq \frac{2/n + \Pr[(u \sim v) \in S]}{\Pr[(u \sim v)]} \to 0.$$

□

3.2 The Upper Bound

Next we show that when p is large, local routing on the hypercube may be performed using n^k probes with high probability. This is a variation on the result of [3] showing that in this regime the metric distortion of the percolation is

bounded. This shows that there is indeed an asymptotic phase transition in the complexity of routing on the hypercube. The proof below shows that $k = O((1-2\alpha)^{-1})$, though it would be interesting to know the exact dependence of k on α (the optimal k need not be integral).

PROOF OF THEOREM 3(II). Here, the terms neighbor and distance relate to the metric of the hypercube before percolation. Percolation neighbor and percolation distance are used for the percolated hypercube $H_{n,p}$.

We refer to the definition of a good vertex from [3], which roughly means having a high degree in $H_{n,p}$. The condition that a vertex is good is determined by the neighborhood of percolation radius 2 around it. In [3], Section (2) the following is proved:

(1) Any given vertex is good with probability of at least $1-\exp(-cn^{1-\alpha})$.

(2) With probability $1-\exp(-cn)$, all pairs of good vertices at distance up to 3 have percolation distance at most l for some $l = l(\alpha) = O((1-2\alpha)^{-1})$.

Now the algorithm is straight forward. Pick arbitrarily a path from u to v, of minimal length: $u = u_0, u_1, \ldots, u_m = v$, and use BFS iteratively to find a path from u_i to u_{i+1}. With probability tending to 1 all the vertices of the path, including u and v are good (each one is not good with probability at most $\exp(-cn^{1-\alpha})$, there are at most n vertices in the path). On this event, the percolation distance between u_i and u_{i+1} is at most l and a path from u_i to u_{i+1} can be found by, say, BFS of complexity n^l. The total complexity is at most n^{l+1}. □

Remark. A natural approach would be to use greedy routing, i.e. at each routing step, probe edges that reduce the Hamming distance to the target. While this strategy may work most of the way, in the final steps a more extensive search is required. It may be the case though that a greedy approach at the early stages of the routing would reduce the exponent in the complexity of the algorithm.

4. THE MESH — UPPER BOUND

In this section we show that the phenomenon observed for hypercubes does not apply when the mesh is considered, i.e. whenever a giant component exists, it is possible to efficiently route between nodes. Consider a cube of the d-dimensional mesh, i.e. a submesh with M^d nodes, and let each edge remain open with probability with some fixed p, and be closed with probability $q = 1-p$. Let $d(\cdot,\cdot)$ denote distance in the mesh, and $D(\cdot,\cdot)$ denote the distance in the giant component (which may be referred as percolation distance). We seek a path between two vertices u, v in the cube with $d(u,v) = n$ (the cube size is M^d which may be much larger than n). We are interested in the routing complexity in terms of n when p is fixed. As mentioned, there exists a number p_c^d such that if $p \le p_c^d$ then $\Pr[u \sim v] = o(1)$ as $n \to \infty$, so hereinafter we assume $p > p_c^d$. For such p there is a giant cluster in the cube, and with probability bounded from 0, both u and v are in the giant component and therefore connected.

We give an algorithm that efficiently finds a short path from u to v. The case of $d = 2$ was solved by Naor and Wieder in [28], where planar duality is used to show that in a two dimensional grid with n^2 vertices, the routing complexity is $O(n)$ w.h.p. It is important to note that it is fairly easy to find a path between u, v if we assume that p is sufficiently close to 1. The main difficulty is pushing the probability p all the way down to p_c^d. In order to do that we need some fairly recent and strong results from Percolation Theory.

4.1 The Routing Algorithm

Consider n vertices which belong to some shortest path between u, v. With high probability many of them are in the giant component and the percolation distance between them is not too large. The algorithm searches around each of them, until the next one is found. More formally:

1. Fix $u = u_0, u_1, u_2, \ldots, u_n = v$ to be a shortest path between u and v. Start from $u_0 = u$.

2. Assume u_i has been reached. Exhaustively probe edges around u_i (using say BFS) until some vertex u_j with $j > i$ is reached.

3. Repeat at most n times until reaching $u_n = v$.

Note that a BFS up do distance k from a vertex takes only $O(k^d)$ queries since only edges of the mesh at distance k from the starting point may be reached. The key point is that it is very unlikely at any iteration that a large depth is needed. Correctness of the algorithm is clear since the search at each stage stops once a closer approximation to v is found. If an open path from u to v exists, then some path will be found.

PROOF OF THEOREM 4. Let u_i be some vertex along the chosen path to v that is in the giant component. Let u_j be the next vertex along the path in the giant component. It follows that the $j-i-1$ vertices along the path between them are outside the giant component, an event that is exponentially unlikely (see [12]):

$$\Pr[|j-i| > k] < e^{-c_1 k} \qquad \text{for some } c_1 = c_1(p) > 0.$$

Note that j always exists since v is assumed to be in the giant component. In practice, u_j might be skipped over by the algorithm if some further vertex u_k is reached first. If the algorithm explores a neighborhood of u_i, it finds a further vertex of the path at distance at most $D(u_i, u_j)$. Thus to bound the number of queries the algorithm makes to reach some u_k we use the following Lemma, which is a proper restatement of result by Antal and Pisztora [4, 13].

LEMMA 8. *For any $p > p_c^d$ and any x, y in a cube M^d of the infinite mesh, let $D(x,y)$ be the percolation distance (in M^d) between them. For some $\rho, c_2 > 0$ depending only on the dimension and p, and for any $a > \rho \cdot d(x,y)$)*

$$\Pr[(D(x,y) > a) \wedge (x \sim y) \in M^d] < e^{-c_2 a}.$$

Either $d(u_u, u_j)$ is large or it is small. In the latter case, $D(u_i, u_j)$ is unlikely to be large, and the former case is itself unlikely:

$$\Pr[D(u_i,u_j) > k] <$$

$$\Pr[d(u_i,u_j) > k/\rho] + \Pr[(d(u_i,u_j) \le k/\rho) \wedge (D(u_i,u_j) > k)]$$

$$< e^{-c_1 k/\rho} + e^{-c_2 k} < e^{-c_3 k}.$$

Consequently, if A_i is the number of queries made from u_i,

$$\Pr[A_i > k] < \Pr[D(u_i, u_j) > ck^{1/d}] < e^{-c_4 k^{1/d}}.$$

Since this is summable, for each vertex u_i of the path that is in the giant component the expected work to get from u_i to a further vertex is $O(1)$.

The number of queries made by the algorithm is at most the sum over all vertices of the path in the giant component of the work to progress from them (actually it is less since some may be skipped over, and some queries may be duplicated). By additivity of expectation, $\mathbb{E}[\sum A_i] = O(n)$. □

5. ORACLE ROUTING VS. LOCAL ROUTING

In this section we consider routing algorithms that are allowed to query *any* edge, and not just edges to which it has established a path. This is called *oracle routing*. Surprisingly, it might be the case that a huge gap exists between the complexity of local and oracle routing. A simple (yet somewhat artificial) example for this is the double binary tree TT_n with fixed $\sqrt{1/2} < p < 1$. In section 2 we showed that any local routing algorithm which finds a path between the two roots of TT_n w.h.p. makes exponentially many queries. The following theorem shows that oracle routing algorithms can do significantly better.

THEOREM 9. *There is an oracle router between the two roots of TT_n with average complexity cn for some $c = c(p) < \infty$ and any $p > \sqrt{1/2}$.*

PROOF. A simple path between the two roots is just a branch up to level n in the first tree joined to the corresponding branch in the second tree. The oracle router is very simple: To find a path from the root to level n that is open in both trees, query edges together with their corresponding edges in the second tree. Each such pair of edges is open with probability $p^2 > 1/2$. The problem is equivalent to finding a path from the root to level n in a super-critical Galton-Watson tree, and a depth first search accomplishes this in expected complexity linear in n.

To see this, observe that any branch of the infinite binary branching process (the Galton-Watson tree) that fails to reach level n has expected size $c(p)$ (and is in fact exponentially unlikely to be large), see [25]. Since at most n bad branches are encountered before reaching level n, the routing complexity is bounded by a sum of n random variables with finite expectation and second moments, and hence is linear in n. □

A more natural example is the graph $G_{n,p}$: For each pair of nodes u, v the edge (u, v) is open with probability p. In our setting it could be thought of as a faulty complete graph. It turns out that local routers can not do much better than querying all the edges:

THEOREM 10. *Any local routing algorithm for the $G_{n,p}$ model where $p = c/n$ (for $c > 1$) has an expected local routing complexity of at least $\Omega(n^2)$.*

PROOF. Assume we wish to route from u to v, and let X be the number of queries required. Let U_t be the set of vertices of the graph which are connected to u by paths known to be open after t queries. Thus $U_0 = \{u\}$. Each vertex in U_t has probability p of being connected to v, thus the probability of finding a route while $|U_t| \le k$ is at most pk.

To reach an additional vertex given that a set of vertices has been reached, the only option is to probe an edge connecting U_t to its complement. By symmetry all such edges are equivalent, and each has probability c/n of connecting to a new vertex. Thus $|U_t| - 1$ is just a sum of $0 - 1$ random variables with expectation $p = c/n$.

Since u, v are both in the giant component with probability at least some $a > 0$, it follows that

$$\Pr[X < k] < \frac{\Pr[U_k > n\sqrt{k}p] + \Pr[X < k | U_k \le n\sqrt{k}p]}{\Pr[u \sim v]}$$
$$< \frac{\Pr[U_k > n\sqrt{k}p] + p \cdot n\sqrt{k}p}{\Pr[u \sim v]} < \frac{\sqrt{k}/n + \frac{c^2\sqrt{k}}{n}}{a}$$
$$= O(\sqrt{k}/n).$$

where the last inequality follows from Markov's inequality. This is close to 0 for $k = o(n^2)$ and shows that the average complexity is $\Omega(n^2)$. □

Next we give tight upper and lower bounds on the oracle routing complexity. The next theorem implies that oracle routing in this case is better than local routing by a factor of exactly $\sqrt{n}$.

THEOREM 11. *There exists a routing algorithm with average complexity $O(n\sqrt{n})$; Any algorithm succeeds with $an\sqrt{n}$ queries with probability at most $O(ca^{2/3} + cn^{-1})$.*

PROOF. Let U_t and V_t be the sets of vertices reachable from u and v after t queries. For the upper bound, consider the following algorithm

(1) Whenever there are unqueried edges between V_t and U_t, probe one of them,

(2) Otherwise, pick the smaller of U_t, V_t and probe an unprobed edge connecting it to a previously unreached vertex.

(3) If no such edge exists, return that $u \not\sim v$.

The algorithm is trivially correct. Since U_t and V_t grow by one vertex at a time, they are roughly of equal size. Since each edge is open with probability c/n, on average a connection between U_t and V_t will be found when $|U_t| = |V_t| = \theta(\sqrt{n})$. Since adding a vertex to either of the sets requires a number of queries with geometric distribution and mean n/c, it takes $O(n^{3/2})$ queries to find a path from u to v.

For the lower bound, note that $|U_t \cup V_t| \le 2 + s_t$ where s_t is the number of open edges found by time t, and $\mathbb{E}[s_t] = ct/n$. Before a connection from u to v is found there must be an open (unprobed) edge between U_t and V_t, and the probability of that is at most $\frac{c|U_v|\cdot|V_t|}{n} \le \frac{c(s_t+2)^2}{4n}$. Thus for any algorithm A and any λ

$$\Pr[comp(A) < t] \le \Pr[s_t > \lambda] + \frac{c(\lambda+2)^2}{4n} \le \frac{ct}{n\lambda} + \frac{2c(\lambda^2+4)}{4n}.$$

If $t = an^{3/2}$, then setting $\lambda = t^{1/3}$ results in:

$$\Pr[comp(A) < an^{3/2}] \le \frac{3ca^{2/3}}{2} + \frac{2}{n}.$$

□

6. OPEN QUESTIONS

So far we observed that sometimes efficient routing is possible whenever the giant component exists, and sometimes the routing complexity has a phase transition at a different value of the percolation parameter. It is natural to assume that this phenomenon relates to the growth rate of the graph. In particular:

- Prove or refute: there exists a family of constant degree graphs in which: the diameter is logarithmic in the number of nodes, and the locations of the phase transition of percolation and routing coincide (at a location bounded away from 1).

In particular it would be interesting to analyze De-Bruijn graphs, Shuffle-Exchange graphs, Butterflies and other families often used in the context of parallel computing.

It would be interesting to see hardness results for oracle routers. Above we see that in the complete graph the best oracle router has complexity $\theta(n^{3/2})$ where the giant component has diameter $\theta(\log n)$. If p were a small power of n, then the diameter would be $O(1)$ and the complexity would still be some power of n, and thus there is no bound for the complexity in terms of the diameter. The results of [3] suggest that oracle routing would *not* help in the hypercube.

- Prove that for $\frac{1}{n} < p < \frac{1}{\sqrt{n}}$ the *oracle* routing complexity of the hypercube is exponential in n.

Finally, it is important to check how well do these results fit into a realistic model. For instance, it would be interesting to see whether our routing algorithm indeed works well under a large number of faults in hypercube-like topologies such as Chord and Skip-Graphs.

7. REFERENCES

[1] M. Ajtai, J. Komlós, and E. Szemerédi. Largest random component of a k-cube. *Combinatorica*, 2(1):1–7, 1982.

[2] N. Alon, I. Benjamini, and A. Stacey. Percolation on finite graphs and isoperimetric inequalities. In *Annals of Probability*, 33(3A):1727-1745, 2004.

[3] O. Angel and I. Benjamini. A phase transition for the metric distortion of percolation on the hypercube. *arXiv:math.PR/0306355*.

[4] P. Antal and A. Pisztora. On the chemical distance in supercritical bernoulli percolation. *The Annals of Probability*, (24):1036–1048, 1996.

[5] J. Aspnes and G. Shah. Skip graphs. In *Proc. 14th ACM-SIAM Symp. on Discrete Algorithms (SODA 2003)*, pages 384–393, Jan. 2003.

[6] B. Bollobas and F. Chung. The diameter of a cycle plus a random matching. *SIAM Journal on Discrete Mathematics*, 1:328–333, 1988.

[7] B. Bollobás, Y. Kohayakawa, and T. Luczak. The evaluation of random subgraphs of the cube. *Random Structures and Algorithms*, 1(3):55–90., 1992.

[8] C. Borgs, J. T. Chayes, R. van der Hofstad, G. Slade, and J. Spencer. Random subgraphs of finite graphs: Iii. the phase transition for the n-cube. In *ArXive Article math.PR/0401071*.

[9] R. Cole, B. M. Maggs, and R. K. Sitaraman. Multi-scale self-simulation: a technique for reconfiguring arrays with faults. In *ACM Symposium on Theory of Computer Science (STOC)*, pages 561–572, 1993.

[10] R. Cole, B. M. Maggs, and R. K. Sitaraman. Routing on butterfly networks with random faults. In *IEEE Symposium on Foundations of Computer Science*, pages 558–570, 1995.

[11] P. Erdos and J. Spencer. Evolution of the n-cube. *Comput. Math. Appl.*, (5):33–39., 1979.

[12] G. Grimmett. *Percolation*. Springer-Verlag, second edition, 1999.

[13] G. Grimmett and J. M. Marstrand. The supercritical phase of percolation is well behaved. *Proceedings of the Royal Society (London)*, A(430):439–457, 1990.

[14] G. R. Grimmett and D. R. Stirzaker. *Probability and Random Processes*. Oxford Science Publications, second edition, 1993.

[15] K. P. Gummadi, R. Gummadi, S. D. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of DHT routing geometry on resilience and proximity. In *Proc. ACM SIGCOMM 2003*, pages 381–394, 2003.

[16] J. Hastad, T. Leighton, and M. Newman. Reconfiguring a hypercube in the presence of faults. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 274–284, May 1987.

[17] J. Hastad, T. Leighton, and M. Newman. Fast computation using faulty hypercubes. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 251–263, 1989.

[18] K. Hildrum and J. Kubiatowicz. Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks. In *17th International Symposium on Distributed Computing (DISC)*, pages 321–336, 2003.

[19] C. Kaklamanis, A. R. Karlin, F. T. Leighton, V. Milenkovic, P. Raghavan, S. Rao, C. D. Thomborson, and A. Tsantilas. Asymptotically tight bounds for computing with faulty arrays of processors. In *The 31st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 285–296, 1990.

[20] A. R. Karlin, G. Nelson, and H. Tamaki. On the fault tolerance of the butterfly. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 125–133, 1994.

[21] J. Kleinberg. The Small-World phenomenon: An algorithmic perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 163–170, 2000.

[22] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. *Advances in Neural Information Processing Systems (NIPS)*, (14), 2001.

[23] M. Krivelevich, B. Sudakov, and V. Vu. Sharp threshold for network reliability. *Combinatorics, Probability and Computing*, (11):465–474, 2002.

[24] F. T. Leighton, B. M. Maggs, and R. K. Sitaraman. On the fault tolerance of some popular bounded-degree networks. *SIAM Journal on Computing*, 27(5):1303–1333., 1998.

[25] R. Lyons and Y. Peres. *Probability on Trees and Networks*.

[26] G. S. Manku, M. Naor, and U. Wieder. Know thy

neighbor's neighbor: The power of lookahead in randomized p2p networks. In *Proceedings of the 36th ACM Symposium on Theory of Computing (STOC)*, pages 54–63 , 2004.

[27] T. R. Mathies. Percolation theory and computing with faulty arrays of processors. In *Proceedings of the Third Annual Symposium on Discrete Algorithms (SODA)*, pages 100–103, 1992.

[28] M. Naor and U. Wieder. Scalable and dynamic quorum systems. In *ACM Conf. on Principles of Distributed Computing (PODC)*, pages 114–122, 2003.

[29] M. Naor and U. Wieder. A simple fault tolerant distributed hash table. In *Second International Workshop on P2P Systems (IPTPS)*, pages 88–97, 2003.

[30] C. M. Newman and L. S. Schulman. One dimensional $1/|j-i|^s$ percolation models: The existence of a transition for $s \leq 2$. *Communications in Mathematical Physics*, 180:483–504, 1986.

[31] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[32] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.