

End-to-End Packet-Scheduling in Wireless Ad-hoc Networks

V.S. ANIL KUMAR¹ MADHAV V. MARATHE¹
SRINIVASAN PARTHASARATHY² ARAVIND SRINIVASAN³

Abstract Packet-scheduling is a particular challenge in wireless networks due to interference from nearby transmissions. A *distance-2 interference model* serves as a useful abstraction here, and we study packet routing and scheduling under this model of interference. The main focus of our work is the development of fully-distributed (decentralized) protocols. We present polylogarithmic/constant factor approximation algorithms for various families of disk graphs (which capture the geometric nature of wireless-signal propagation), as well as near-optimal approximation algorithms for general graphs. A basic distributed coloring procedure, originally due to Luby (*Journal of Computer and System Sciences*, 47:250–286, 1993), underlies many of our algorithms. Experimental work of Finocchi, Panconesi, and Silvestri (SODA 2002) showed that a natural modification of this algorithm leads to improved performance, and a rigorous explanation of this was left as an open question; we prove that the modified algorithm is provably (much) better in the worst-case. Finally, using simulations, we study the impact of the routing strategy and the choice of parameters on the performance of our distributed algorithm for unit disk graphs.

1 Introduction

Packet routing and scheduling are two key problems that arise in the control and design of packet-switched networks. To send a packet from a node u to node v in a network, one needs to choose a path from u to v ; once the paths for all the packets have been determined, we are left with the issue of scheduling the packets along the paths. If multiple packets reach some node simultaneously, they must be

¹ Basic and Applied Simulation Science (CCS-5) and National Infrastructure and Analysis Center (NISAC), Los Alamos National Laboratory, MS M997, P.O. Box 1663, Los Alamos, NM 87545. Email: {anil,marathe}@lanl.gov. The work is supported by the Department of Energy under Contract W-7405-ENG-36.

²Department of Computer Science, University of Maryland, College Park, MD 20742. Most of the work was done while visiting Los Alamos National laboratory. Email: sri@cs.umd.edu.

³Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. Research supported in part by NSF Award CCR-0208005. Email: srin@cs.umd.edu.

queued or dropped. At most a given number of packets can be sent at a time on an edge (and sometimes nearby edges cannot have traffic simultaneously), and the scheduling problem is to decide which of the packets queued at a node should be sent first. The exact algorithms used for these problems have a profound impact on network performance. In this paper, we study the scheduling problem in the context of wireless ad-hoc networks. An *ad-hoc network* consists of a group of *transceivers* (also known as *stations*) sharing a common wireless channel and communicating with each other using this channel. Our two foci here are dealing with *interference*, and the development of *distributed* algorithms that have provably good approximation guarantees. Concretely, we are given a graph $G = (V, E)$, with packets that need to be sent from some sources to some destinations. We will also assume that paths are given; for the scheduling algorithms that we use, the algorithm of [27] can be modified to obtain good paths. If we find paths using a variant of the algorithm of [27] and then run our scheduling algorithms, we only lose an additional constant factor in the approximation ratio. We also discuss, via empirical analysis, the influence of the routing strategy on the overall performance of the protocols.

The key issue in our case is *interference*. Almost all the theoretical, algorithmic work on packet routing and scheduling so far (e.g., [13, 15]) has primarily considered the following constraint: at most one packet can be sent on an edge at a time. While useful in wired networks, this assumption is not sufficient for wireless networks, where transmission by one wireless (or transceiver) precludes transmission by all nearby transceivers. An example of this is shown in Figure 1: if the transmissions on (a, b) and (c, d) occur simultaneously, node b receives garbled signals from both a and d simultaneously; note that nodes a and c are not aware of this problem at b (unlike, for instance, in Ethernet, where a collision is immediately detected by all nodes). This is called the *hidden node problem* in wireless networks. On the other hand, the transmissions on (a, b) and (d, e) in Figure 1 can happen simultaneously. To model this notion combinatorially, first we construct the *interference graph* for a set of transceivers by having a node for each radio, and an edge (u, v) if v lies in the transmission range of u (note that we are assuming equal transmission ranges here; see Section 2 for the general versions that we work with). We now require that the set of edges on which simultaneous transmission happens forms a *distance-2 matching* (these definitions are given in section 2); edges (a, b) and (d, e) in Figure 1 satisfy this property. this model is called the *distance-2 interference model* [7, 8]. Earlier wireless network models (e.g. [22, 24, 23, 12]) only placed vertex constraints: nodes that transmit simultaneously must form an independent set; this is clearly inadequate, from the above description.

The above requirement might seem too restrictive sometimes: for instance, in Figure 1, b could transmit to a and c could transmit to d simultaneously; this is called the *exposed node problem* in radio networks. While this is true, reliable transmission requires transmission both ways (to acknowledge receipt of packets, for instance), and then the constraint described above seems reasonable. The 802.11 protocol addresses

these issues by the MACA (Multiple Access with Collision Avoidance) algorithms (see [21] for details): a sender first transmits a *Request to Send (RTS)* signal and the receiver then sends a *Clear to Send (CTS)* signal. Once the transmission starts, the receiver sends acks to the packets he receives correctly.

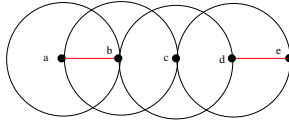


Figure 1: Distance-2 Interference

The discussion so far only addresses the restrictions that the MAC layer places in radio networks. The MAC layer only ensures delivery of packets from a node to its immediate neighbor. The *Routing Layer* of the protocol stack is responsible for choosing the routes along which packets must travel (see [21] for details), and the MAC layer moves packets one step at a time along these paths. By incorporating the above interference constraint, we have the following generalization of the problem studied in [13], which we call **END-TO-END PACKET SCHEDULING WITH INTERFERENCE (EPSI)**: given a wireless network with packets that have pre-assigned paths, develop a protocol to schedule the movement of the packets from their sources to their destinations, while ensuring that at each step packets transmitted successfully were on edges forming a distance-2 matching. Our goal is to minimize the maximum time taken by any packet to reach its destination, or the *makespan*. Also, these algorithms need to be distributed in order to be useful in practice. We remark that the problem of routing and scheduling using “standard” (i.e., distance-1 as opposed to our distance-2) matchings, has been studied by [1]. Our main contributions are as follows.

(a) Packet-Scheduling Algorithms. We give the first provably-efficient distributed algorithms for packet-scheduling in networks with the distance-2 interference constraint. While the original motivation for the model is radio networks, we also study this problem on different non-geometric graph classes. Let n denote the number of nodes in the given network. For arbitrary graphs with maximum degree Δ , we present a distributed $O(\Delta \log^2 n)$ -approximation algorithm; we also show that it is hard to approximate the minimum time within a factor of $\Delta^{1-\epsilon}$ for any constant $\epsilon > 0$, even in the centralized, polynomial-time setting. As is well-known, disk graphs, defined in Section 2, model radio communication well, and we obtain the following improved approximation algorithms for them. For general disk graphs, we provide a distributed $O(\log^2 n)$ approximation algorithm and a centralized $O(\log n)$ -approximation algorithm. For unit-disk graphs, we give a distributed $O(\log n)$ -approximation algorithm and a centralized $O(1)$ -approximation algorithm. The above distributed algorithms are in the synchronous model; for unit-disk graphs, we also obtain a distributed $O(\log^3 n)$ approximation in an asynchronous model. Our results for unit disk graphs can also be extended to a more general class of graphs

known as (r, s) -civilized graphs¹ As discussed in [12, 7] (r, s) -civilized graphs are a more realistic model for ad-hoc networks due to their ability to model occlusions.

A key driver of our algorithms is the *random delays* technique [13]; we combine this with additional new ideas to develop our algorithms. In particular, the standard lower bound of “congestion plus dilation” is shown to be no longer good in our model, and we use a more refined lower bound. For general disk graphs, we introduce a new packing result, which, along with the graph-theoretic notion of *inductive coloring*, helps us claim our approximation bounds. Many of our algorithms also employ a basic (list-)coloring algorithm due to [17]; we prove the improved performance of a variant of this algorithm, as discussed next.

(b) Distributed Coloring and Network Decomposition. Variants of the above-mentioned algorithm due to [17] have been useful in many distributed coloring algorithms (see, e.g., [11]) and also in our algorithms for packet scheduling. Due to its simplicity and generality, an extensive experimental evaluation of this and related coloring algorithms has been undertaken in [10]. A natural “non-lazy” variant of the original algorithm of [17] was empirically found to be much better. To quote [10], “In particular, when compared with Luby’s algorithm it consistently turned out to be 2-3 times faster. The available asymptotic analyses do not explain this behavior ... and we leave it as an interesting open question”. We present such a rigorous explanation regarding the worst-case behavior of the two algorithms. We exhibit a constant c such that for any n -vertex graph, the modified algorithm would terminate within $c \log n$ steps with high probability; we also show that the original algorithm of [17] would still need to process $n^{\Theta(1)}$ vertices after $c \log n$ steps with high probability, when run on the complete graph K_n . In passing, we also improve the running time of a powerful distributed primitive known as *network decomposition* [4, 5, 6, 16, 20]; see, e.g., [9] for an application to distributed network protocols. We observe that the protocol of [16] which runs in $O(\log^2 n)$ time, can be modified to run in $O(\log n)$ time. In addition to being of independent interest, this is likely to be useful in an efficient, practical implementation of our algorithms.

(c) Empirical Analysis. We perform a detailed analysis of some of our algorithms, and the routing algorithm based on [3] on random geometric graphs, with a large number of packets injected at random sources. Several popular routing protocols, such as DRS and AODV (see [21] for details) are based on shortest path routing. Another popular algorithm is Valiant’s paradigm [26] We observe that the algorithm based on [3] yields significantly smaller congestion, compared to the other two approaches. Also, the congestion in Valiant’s paradigm is always almost twice that in the Shortest path case, which is interesting in itself. For expanders such as hypercubes, Valiant’s paradigm is known to perform very well; our networks here may not be good ex-

¹For each fixed pair of real values $r > 0$ and $s > 0$, a graph G can be drawn in R^2 in an (r, s) -civilized manner if its vertices can be mapped to points in R^d so that the length of each edge is at most r and the distance between any two points is at least s and no two edges intersect (except at their endpoints).

panthers. We also study the impact of various parameters on the performance of our distributed algorithms. Specifically, the two parameters of interest are the number of colors (equivalently, the frame length) used in the distributed coloring algorithm and the maximum initial random delay which each packet could be subjected to. Increasing either of the two would lead to better performance (in terms of packet losses) at the cost of increased makespan. Our studies indicate that for a wide range of values, increasing the frame length has a much bigger impact on performance rather than maximum initial random delay.

Related Work. For the earlier model of one packet per edge at a time, one of the most significant results is the work of Leighton et. al. [13], where they show the existence of a constant factor approximation, using the Local Lemma. Their result assumes that the packets already come with pre-specified paths. This work was followed by a series of papers improving either the performance or the complexity of the algorithm (see [14, 26, 18]). Rabani and Tardos [25] give a distributed algorithm for this problem, that takes time $O(C + D(\log^* n)^{O(\log^* n)} + \log^6 n)$, which was improved by Ostrovsky and Rabani [19] to $O(C + D + \log^{1+\epsilon} n)$. The distance-2 interference model MAC layer packet-scheduling in ad-hoc networks has been considered in [7, 22, 24]. The problem can be cast as either vertex or edge coloring depending on the particular setting [8]. The problem of finding an end-to-end schedule of packets under radio interference model has not been studied prior to this paper.

2 Preliminaries

Formally, an instance of the packet scheduling problem is specified as $EPSI(G(V, E), \{p_1, \dots, p_k\})$. $G(V, E)$ is the underlying interference graph (described below), which would be undirected for the most part, except in disk graphs. p_1, \dots, p_k are the packets to be transmitted, with packet p_i starting at s_i and destined for t_i , along the path P_i . The path P_i is encoded in packet p_i . We will assume that any packet takes one unit of time to cross a link and at any time at most one packet can cross a link. In addition, if packets are being sent simultaneously on edges $e = (u, v), e' = (u', v')$, then $d(e, e') \geq 2$ ($d()$ is defined below). If a packet-transmission violates either of these requirements (i.e., if some other transmission is made simultaneously on an edge that is within distance less than two), then the transmission *fails* and has to be retried later. Each node/edge has a buffer in which a packet can wait till it successfully moves to the next node in its path. The objective is to construct a *schedule*, \mathcal{S} , that decides which packet should be sent out at a node at any time. A schedule is *valid* iff it sends all packets along their paths subject to the above re-trial requirement (in the case of failures). Let $C_{\mathcal{S}}(p_i)$ denote the (random) time at which packet p_i is delivered in schedule \mathcal{S} . The *Makespan* of \mathcal{S} , denoted by $C(\mathcal{S}) = \max_i C_{\mathcal{S}}(p_i)$ is the time taken by \mathcal{S} to route all the packets, and our objective is to construct a schedule with low makespan. For the most part, we will assume a synchronous, distributed communication model; this is reasonable in our context

of a primarily local-area network, since nodes can keep synchronized, e.g., by using GPS receivers. Let Δ denote the maximum degree in G . Given subsets $A, B \subset V$ in graph $G(V, E)$, the distance $d_G(A, B)$ is defined to be the minimum length of the (directed or undirected) shortest path over all pairs of vertices $a \in A, b \in B$. For edges $e = (u, v), e' = (u', v') \in E(G)$, $d_G(e, e')$ is defined as $d_G(\{u, v\}, \{u', v'\})$. For vertex $w \in V$, we will also sometimes use $d_G(w, e)$ to denote $d_G(\{w\}, \{u, v\})$. We will drop the subscript whenever it is clear. A subset $M \subset E$ is said to be a *distance-2 matching* if $d(e, e') \geq 2$ for any distinct pair $e, e' \in M$.

Disk Graphs. A disk graph is specified by a set of points V , with a disk $D(v)$ centered at each $v \in V$, with radius $r(v)$. The directed graph $G(V, E)$ induced by these disks is the following: the set of nodes is V and a (directed) edge (u, v) is present if $v \in D(u)$. The special case where all radii are equal is called a unit disk graph, and in this case, if edge $(u, v) \in E$, then $(v, u) \in E$; as a result we can think of unit disk graphs as undirected. See section 3 for more details on the model.

3 Disk graphs

A popular model for radio networks is a disk graph. The disk around a point naturally corresponds to the effective transmission range of the radio. As described in Section 2, we will think of disk graphs as being directed. Because of our communication model, which requires two way transmission, only bidirected edges can be used for transmission; so we assume that the paths P_1, \dots, P_k only use bidirected edges. The undirected edges only contribute to the interference: therefore, if edge $e = (u, v)$ is being used at time t , no other edge $e' = (u', v')$ with $\min\{d(u', e), d(v', e), d(u, e'), d(v, e')\}$ can be used simultaneously. Our algorithm involves choosing random delays at the first step, as in [13] and then scheduling packets at each time step by solving a coloring problem. A sequential coloring algorithm yields an $O(\log n)$ approximation to the makespan, while a distributed coloring yields an $O(\log^2 n)$ approximation.

We need some more notation for disk graphs. For edge $e = (u, v)$, define $r(e) = r(u) + r(v)$. Define $N_{\geq}(v) = \{e' \mid d(v, e') \leq 1, r(e') \geq r(v)\}$ and $N_{\geq}(e) = \{e' \mid d(e, e') \leq 1, r(e') \geq r(e)\}$. Define $C(e)$ to be the number of packets whose path uses some edge of $N_{\geq}(e)$, and $C = \max_e C(e)$. D is still defined to be the dilation, as before.

Lemma 3.1 *For any vertex v , the size of the largest distance-2 matching in the subgraph induced by $N_{\geq}(v)$ is $O(1)$.*

Lemma 3.2 $OPT = \Omega(C + D)$

Proof We just need to verify that $OPT = \Omega(C)$. This follows from the previous lemma: at any time instant, the set of edges on which packets can be transmitted simultaneously forms a distance-2 matching. From the previous lemma, for any edge $e = (u, v)$, at most $O(1)$ edges can be simultaneously used within $N_{\geq}(u)$ and within

$N_{\geq}(v)$. Therefore, $\Omega(C(e))$ timesteps are needed to transmit all the packets in $N_{\geq}(e)$.

■

Algorithm DISKEPS

1. Each packet p_i chooses a delay Y_i uniformly at random from $\{1, \dots, cX_0\}$ ($c > 0$ is a specific constant and $X_0 = C + D$). and waits for $Y_i c \log^2 n$ steps at s_i .
2. From time $Y_i + 1$ onwards, the packet moves along path P_i . At each node v on P_i , the packet spends $c \log^2 n$ steps.
3. p_i reaches the j th node v on P_i by time $Y_i + j \cdot c \log^2 n$. If it reaches before $Y_i + j \cdot c \log^2 n$, it waits in the queue till then.
4. Let T be a multiple of $c \log^2 n$. All packets are moved to their next hop, from the current location at time T , during the time interval $[T, \dots, T + c \log^2 n]$ by the following steps.
 - (a) Let E_T be the edges on which packets need to be moved at step T . Run Subroutines DISKCOL below in $c' \log n$ steps to choose a color $\alpha_i \in \{1, \dots, c'' \log^2 n\}$ for each packet p_i (this can be done, by Lemma 3.3)
 - (b) At step $T + c' \log n + \alpha_i$, packet p_i is moved to the next node on its path. Once it reaches t_i , the packet is removed.

Subroutines DISKCOL

1. Repeat the following steps in rounds $i = 1, 2, \dots$ till all edges are colored.
2. **Round i :** Each uncolored edge chooses to do nothing with probability $1/2$. With the remaining probability, it performs the following steps.
 - (a) Each uncolored edge e chooses a color uniformly at random from $\{(i - 1)d \log n + 1, \dots, id \log n\}$.
 - (b) Each edge e checks whether some edge e' in $N_{\geq}(e)$ has chosen the same color as e .
 - (c) If there is no such edge e' , e is colored with the color it chose; otherwise, e remains uncolored in this round.

Figure 2: Distributed algorithm for solving $EPSI$ problem on disk graphs.

Figure 2 contains a description of our algorithm for solving the $EPSI$ problem on disk graphs. The algorithm is called **Algorithm DISKEPS**. The description of the algorithm is complicated because of the distributed nature. The underlying sequential algorithm, based on [13], is simple: first we construct an invalid schedule \mathcal{S}' which does not respect the matching constraints by first giving a random delay at the origin of each packet, and then letting it zip through, one step at a time. We then show that

at each step, for each packet p_i , $O(\log n)$ packets are transmitted simultaneously on edges not within distance-2 of it. We now construct a valid schedule \mathcal{S} by considering all packets at each time step T , and moving them to their next hop, by a distance-2 coloring algorithm, DISKCOL, described below. The algorithm below is a distributed algorithm, based on [17]. Lemma 3.3 proves that this algorithm runs for $O(\log n)$ rounds and uses $O(a \log n)$ colors for coloring the set E_T of edges in the algorithm DISKEPS. The lemma also proves that the additional $\log n$ factor is not needed for the sequential greedy algorithm.

The algorithm's performance can be shown in the following three steps.

- Lemma 3.3**
1. *At any T (which is a multiple of $c \log^2 n$), for any edge e , define $C_T(e) = N_{\geq}(e) \cap E_T$. Then, $C(e) = O(\log n)$, for each e, T , with high probability.*
 2. *Let T be any multiple of $c \log^2 n$. The algorithm DISKCOL colors the edges in E_T (defined in algorithm DISKEPS) in $c' \log n$ steps, using $c'' \log^2 n$ colors, with high probability. The set E_T can be colored sequentially using $O(\log n)$ colors.*
 3. *Algorithm DISKEPS schedules the packets in time $O(\log^2 n)$ times the optimal schedule, with high probability. The sequential version of the algorithm has an approximation guarantee of $O(\log n)$.*

Proof Sketch: (1) Let each packet p_i be at the end point of edge e_i at step T . During the time interval $[T, T + c \log^2 n)$, packet $p_i, \forall i$ only moves along e_i , in an interference free manner. Now, just consider the time steps $j c \log^2 n, j = 0, 1, \dots$ with respect to these, packets wait for a random delay, and then move to their destination, one step at a time. As in [13], the expected value of $C_T(e) \leq 1$, for any such $T = j c \log^2 n$ and for any edge e . The statement now follows by a Chernoff bound.

(2) Order the edges in E_T in nonincreasing order of their $r(e)$ value. Since $C_T(e) = O(\log n)$, list coloring uses $O(\log n)$ colors for this order. For the distributed algorithm DISKCOL, in each round, each uncolored edge gets a color that does not conflict with any edges with higher $r()$ value, with constant probability. As a result, each edge gets a color in $O(\log n)$ steps, with high probability. Since $O(\log n)$ colors are used in each round, this gives a total of $O(\log^2 n)$ colors in all.

(3) The previous statements imply that each packet p_i moves one edge forward on P_i after the $Y_i c \log^2 n$ steps, with high probability for the distributed version. For the sequential coloring, this movement happens every $O(\log n)$ steps. ■

4 Unit disk graphs

When all disks have fixed radius, we obtain significant improvements in the approximation guarantee. By a repeated planar decomposition, we can actually get an $O(1)$ approximation. This sort of decomposition only requires a sparsity condition, rather than geometry, and can be applied to bounded genus graphs also. We then give an

$O(\log n)$ distributed algorithm by refining the analysis of the algorithm DISKEPS in Section 3. Finally, we give a distributed algorithm in the asynchronous model with a $O(\log^3 n)$ approximation guarantee.

4.1 An $O(1)$ approximation algorithm

The notation used here is defined in Section 2. We assume the common radius to be 1 here. Let B be a bounding box in the plane for the points in V . If we assume that G is a connected graph, B must have sides of length $O(n)$. Let B_k be a partition of B into smaller grid cells, each cell having dimensions $k \times k$. Let B'_k be obtained by translating the grid B_k by $k/2$ along the x and y axes. A cell in B_k will refer to one of the $k \times k$ sized pieces in it, and a point in B_k is any lattice point with integer coordinates. We will denote lattice points within B_k by small letters and the $k \times k$ cells within B_k by capital letters.

For a disk S of radius 1 in the plane, let $C(S)$ be the number of paths P_i that visit some vertex $v \in V$, located within S . Define $C = \max_S \{C(S)\}$. As before, D is the length of the longest path. $\max\{C, D\}$ is still a lower bound on the optimal size, and as the following observation shows, even if C is defined as the maximum of $C(D(x))$ for points $x \in B$, $C + D$ is still at least a constant factor of the optimal.

The main intuition for the partitioning algorithm is the following. After the first step (giving random delays) of [13], both C and D become $O(\log n)$ within each time frame. This means that the smaller scheduling subproblem in any frame is localized to a $O(\log n) \times O(\log n)$ region of the plane. Thus, in addition to the temporal decomposition, we are able to do a *spatial decomposition* too. If we carry this process once more, we end up with scheduling problems on regions of size $O(\log \log n) \times O(\log \log n)$, and at this point, we can solve by brute force in $poly(n)$ time. The algorithm is described in Figure 3 and is called **Algorithm UNITDISKEPS**

Subroutine PARTITION(k), described below, forms a partition of the problem into smaller subproblems, each on a grid of dimensions $2 \log k \times 2 \log k$.

Lemma 4.1 *There exists a choice of random delays for all the packets in step 1 of subroutine PARTITION(k) which satisfies the following property: for any time frame T of length $\log k$, and for any lattice point p in the input to the subroutine, the number of paths visiting some vertex $u \in V$ located in $S(p)$ is $O(\log k)$.*

Proof The input points lie in a $k \times k$ grid. By our assumption that each path visits only a constant number of vertices within $S(v)$ for any v , it follows that the largest path, D , is $O(k)$. Let $X_0 = C + D$. Consider any grid point v , and any time t . $Pr[\text{packet } p_i \text{ passes through } D(v) \text{ at } t] \leq \frac{1}{cX_0}$ and $E[\# \text{ packets through } D(v) \text{ at } t] \leq 1$. By the Chernoff bound, the number of paths visiting vertices in $D(v)$ during a time frame T of length is $O(\log n)$ with probability at least $1 - \frac{1}{k^c}$, for some constant $c > 0$. Since there are $O(k^2)$ grid points, and $O(k)$ time frames to consider, the lemma follows by the union bound. ■

Algorithm UNITDISKEPS

1. Run subroutine PARTITION(n) to create smaller problems on $2 \log n \times 2 \log n$ sized grids.
2. For each of the subproblems on a $2 \log n \times 2 \log n$ sized grid, run subroutine PARTITION($2 \log n$) to create smaller subproblems on $O(\log \log n) \times O(\log \log n)$ sized grids.
3. Solve the scheduling problem within a $O(\log \log n) \times O(\log \log n)$ sized grid by exhaustive search (details in Lemma 4.3).
4. Combine the schedules for all the subproblems together to form the whole schedule.

Subroutines PARTITION(k)

INPUT A scheduling instance on a $k \times k$ region.

OUTPUT Partition this instance into smaller scheduling problems, defined on grid cells of size $2 \log k \times 2 \log k$.

1. Construct an invalid schedule $\mathcal{S}_1(\Pi)$ in the following manner:
 - (a) For each packet, choose a random delay from $\{1, \dots, c(C + D)\}$, where $c > 0$ is a specific constant, such that Lemma 4.1 is satisfied (the property in Lemma 4.1 can be checked in polynomial time; so this step involves choosing the random delays, checking the property and repeating if necessary).
 - (b) Allow each packet to zip through along its path, after waiting for the random delay at the source.
2. Partition B into grids $B_{2 \log k}$ and $B'_{2 \log k}$.
3. Consider successive time frames of length $\log k$.
4. For each time frame T of size $\log k$, assign each packet p_i to a unique cell Z in $B_{2 \log k}, B'_{2 \log k}$ such that the path traversed by p_i during T lies completely within Z ; break ties arbitrarily.
5. For each time frame T , for each cell Z in $B_{2 \log k}, B'_{2 \log k}$, the problem restricted to Z involves scheduling the packets assigned to it during T , along the segments of the paths within Z .

Figure 3: Algorithm for solving *EPSI* problem on unit disk graphs.

Lemma 4.2 *In step 4 of subroutine PARTITION(k), the path traversed by any packet p_i during a time frame T (of length $\log k$) can be uniquely assigned to some cell in $B_{2\log k}$ or $B'_{2\log k}$.*

Lemma 4.3 *A schedule of length $O(\log \log n)$ can be constructed for the scheduling problem on a grid of size $O(\log \log n) \times O(\log \log n)$.*

Proof By our assumption, each packet can only traverse $O(\log \log n)$ steps within such a grid cell; so $D = O(\log \log n)$. Also, by the guarantees of the subroutine PARTITION, $C = O(\log \log n)$ within such a grid. Therefore, the total number of packets is $O((\log \log n)^3)$. The maximum number of possible schedules is this number raised to the power of C , which is at most a polynomial in n . Therefore, we can try out all schedules in polynomial time. ■

Corollary 4.4 *A schedule for the distance-2 interference problem on unit disk graphs of length $O(1)$ times the optimal can be found in polynomial time.*

4.2 Distributed algorithms

Synchronous model Algorithm DISKEPS detailed in Figure 2 for disk graphs can be modified to yield a better bound of $O(\log n)$ for the case of unit disk graphs. Since all disks have the same radius, the notation and ordering of Section 3 is not needed. We will use the lower bounds C, D defined in the previous subsection.

The first three steps of the algorithm DISKEPS are unchanged, except for the delays and time frames being multiples of $\log n$, instead of $\log^2 n$. In step 4 of the algorithm, instead of running algorithm DISKCOL, we actually run Luby's algorithm [17]. This takes $O(\log n)$ steps, and uses $O(\Delta)$ colors, where Δ is the max degree of the subgraph induced by E_T , which we is $O(\log n)$.

Lemma 4.5 *There is a distributed, $O(\log n)$ approximation algorithm for the packet-scheduling problem on unit disk graphs.*

Asynchronous model The algorithm described in the previous section needs centralized, synchronous control, which is difficult in practice. We now describe a completely distributed, asynchronous, randomized algorithm that gives a schedule of length at most $O(\log^2 n)$ times the optimal, with high probability.

The basic idea is to combine contention resolution methods along with the random delays plus coloring techniques that have been used so far. Note that if there are C packets in the vicinity of some packet p , that are contending for a transmission slot at a time, all of these can be scheduled in $O(C \log n)$ steps with high probability. The random delays step allows us to reduce the effective congestion at every step, and after that one can perform coloring via the contention resolution. Note that we

need to simulate some sort of synchronization, to ensure that the right set of packets is contending at any time, and this can easily be achieved by suitable waiting for polylogarithmic steps at regular intervals. We also need to keep track of the path encoded in the packet headers. The algorithm is described in Figure 4 and is referred to as **Algorithm** ASYNCHRONOUSUNITDISKEPS.

Algorithm ASYNCHRONOUSUNITDISKEPS

1. Each packet p_i chooses a delay uniformly at random from $\{1, \dots, \alpha_1 X_0\}$, where $\alpha_1 > 0$ is a constant and X_0 is as defined before.
2. For each packet p_i , define P'_i to be a path obtained by prepending X_i virtual edges to P_i , the original path (this is implemented by following self loop edges at the origin of p_i).
3. Partition each P'_i into segments of length $\log n$ each.
4. Let $W = \alpha_2 \log^3 n$. During time $(j-1)W, \dots, jW-1$, each packet p_i attempts to proceed on its j th segment in the following manner:
 - (a) Let $W' = \alpha_2 \log^2 n$.
 - (b) During time $(j-1)W + (\ell-1)W' \log n, \dots, (j-1)W + \ell W' \log n - 1$, each packet will attempt to move on the ℓ th edge of the current segment:
 - i. t denotes the current time; $t \in \{(j-1)W + (\ell-1)W' \log n, \dots, (j-1)W + \ell W' \log n - 1\}$
 - ii. If packet p_i has already sent its ℓ th edge of the j th segment, keep waiting till time $(j-1)W + \ell W' \log n - 1$.
 - iii. If p_i has not already succeeded in sending its ℓ th edge of the j th segment, it chooses to send with probability $\frac{1}{\alpha_2 \log n}$.
 - iv. If p_i sent at time t , and detected a collision, retry as in the above step.
 - (c) After packet p_i succeeds in moving on the ℓ th edge of current segment j , it just waits till time $(j-1)W + \ell W' \log n - 1$.
5. If packet p_i has already finished moving on the j th segment of its path P'_i , it just waits at current node till time $jW-1$.

Figure 4: Asynchronous distributed algorithm for solving $EPSI$ problem on unit disk graphs.

Lemma 4.6 *Each packet p_i moves on its ℓ th edge of its j th segment during time $(j-1)W + (\ell-1)W' \log n, \dots, (j-1)W + \ell W' \log n - 1$ with high probability. Each packet p_i moves on its j th segment during time $(j-1)W, \dots, jW-1$ with high probability.*

Proof Sketch: This statement is essentially equivalent to Lemma 4.5. ■

Lemma 4.7 *Each packet p_i moves on its j th segment during time $(j-1)W, \dots, jW - 1$ with high probability.*

Corollary 4.8 *All packets are delivered within time $O(OPT \log^3 n)$ with high probability, where OPT is the length of the optimal schedule.*

5 Arbitrary graphs

We now handle the case of general graphs; the maximum degree Δ of the given graph will play a key role now. We start by claiming the hardness of approximating EPSI:

Lemma 5.1 *Let ϵ be an arbitrary positive constant. It is not possible to approximate the optimum makespan of every instance of EPSI problem in polynomial time within a factor of $\Delta^{1-\epsilon}$, unless $P = NP$.*

Proof The reduction is from Vertex Coloring. Given a graph $G(V, E)$ whose chromatic number we wish to approximate, we construct graph $G'(V', E')$ in the following manner. For each $v \in V$, we add vertices $v, m(v)$ in G' . Each edge $(u, v) \in E$ is part of E' ; in addition, we have edges $(v, m(v)), \forall v \in G$ in E' . We have packet p_v destined from $m(v)$ to v , for each $v \in V$, and the path that p_v has to take is exactly the edge $(v, m(v))$. $(G', \{p_v \mid v \in V\})$ is our instance of PSI. We will argue that the minimum makespan of this instance of PSI is exactly the chromatic number of G , and this completes the proof. Two packets p_u, p_v can be simultaneously delivered if and only if $(u, v) \notin E$ (and also $\notin E'$), by construction of G' . Therefore, the set of packets that can be simultaneously sent in a time unit corresponds to an independent set in G , and the time required to transmit all the packets equals the chromatic number.

Given the $n^{1-\epsilon}$ -hardness of chromatic number, the $n^{1-\epsilon}$ -hardness of our problem follows immediately. Furthermore, as pointed out by Subhash Khot to us, the approach of [28] on the hardness of the independent set problem, in fact show that chromatic number is hard to approximate to within $\Delta^{1-\epsilon}$. The lemma follows. ■

5.1 A Distributed Algorithm

We now present a distributed approximation algorithm using the the random delays approach from [13]. First, we need to define a better lower bound on the optimal makespan, since edge congestion is too weak: consider a complete graph K_n with one packet to be sent on every edge; the edge congestion and dilation are each 1, but the optimal makespan is $m = n(n-1)/2$. Define C as the maximum, over all edges $e = (u, v)$, of the number of paths that pass through u or v (or through both). Define D as usual to be the dilation, and let $X_0 = \max\{C, D\}$. path. Let OPT denote the number of steps in the optimum schedule. It is easily seen that $OPT \geq X_0$. Our

algorithm GENERALEPS has two steps. (1). Construct an invalid schedule \mathcal{S}' in the following manner: (a) For each packet, choose a random delay from $\{1, \dots, cX_0\}$ ($c > 0$ is a specific constant). (b) Allow each packet to zip through along its path, after waiting for the random delay at the source. (2). Now that step (1) is done, Convert \mathcal{S}' into a valid schedule \mathcal{S} as follows. Let E'_t be the set of edges on which packets are being transmitted at time t in \mathcal{S}' . Greedily distance-2 edge color edges in E'_t ; let $E'_t(1), \dots, E'_t(k)$ be the color classes (k is the number of colors used): each $E'_t(i)$ is a distance-2 matching. This can be done distributively. Schedule the packets using E'_t in k steps: at the i th step schedule packets in set $E'_t(i), i = 1, \dots, k$. The algorithm can be made distributed in the same way as algorithm DISKPS in § 3.

Our proof follows in the lines of [13]: we first show that schedule \mathcal{S}' has length $O(OPT)$, and at any time t , edges in E'_t can be colored using $O(\log n)$ colors. The following sequence of lemmas formalize this.

Lemma 5.2 *Schedule \mathcal{S}' has length $O(C + D)$, and at any time t and for any edge $e = (u, v)$, the number of packets that use any edge e' incident on u or v at time t is $O(\log n)$ with high probability.*

Proof The random delay for each packet is $O(C + D)$. After that, each packet reaches its destination in $O(D)$ time in schedule \mathcal{S}' . Therefore, \mathcal{S}' has length $O(C + D)$.

Next, we bound the congestion at any time. Consider any edge $e = (u, v)$ and time t . Let P be a packet whose path passes u or v . Then, $Pr[P \text{ passes through } u \text{ or } v \text{ at } t] \leq \frac{1}{cX_0}$, and $E[\#\text{packets through } u, v \text{ at } t] \leq 1/c$, since the maximum number of paths passing through u or v is at most $C \leq X_0$. Since the delays for packets are independent variables, by the Chernoff bound, $Pr[\#\text{packets through } u, v \text{ at } t \geq d \log n] \leq 1/n^5$, for a suitable constant $d > 0$. Since the number of edges and the number of time steps are both bounded by $O(n^2)$, the lemma follows. ■

The following lemma is used in constructing \mathcal{S} from \mathcal{S}' .

Lemma 5.3 *For any t , the edges in E'_t can be distance-2 edge colored distributively using $k = O(\Delta \log n)$ colors in $O(\Delta \log n)$ time, with high probability.*

Proof From the previous lemma, it follows that for any edge $e = (u, v)$, the number of edges e' incident on u, v on which a packet is transmitted at time t is $a = O(\log n)$ with high probability. It is possible that multiple packets get transmitted through an edge simultaneously, but we ignore this effect for now; the argument can be easily extended to handle this case. Consider an edge $e = (u, v)$ in E'_t . In the square of the line graph of G , the degree of the vertex corresponding to e is $O(\Delta a)$, which can be colored using $O(\Delta a)$ colors in a completely distributed manner. ■

Corollary 5.4 *Algorithm GENERALEPS produces a schedule of length $O((C + D) \cdot \Delta \cdot \log n)$ with high probability.*

In the case where the paths are not specified, we can use the algorithm of [27] to get paths whose $C + D$ is within $O(1)$ of the optimal, and schedule packets using the above algorithm. We note that the algorithm in [27] is based on LP-rounding and designed for wired networks. However, the LP in [27] can be modified to accommodate D2-edge congestion for wireless networks rather than the plain edge congestion in wired networks.

6 Distributed Vertex Coloring

Our packet scheduling algorithms can be viewed as implementing a distributed coloring algorithm within each frame. This motivates the question of efficient distributed algorithms for various coloring problems. Luby's algorithm [17] is one of the most often used distributed vertex coloring algorithms. One of the parameters in Luby's algorithm is the *sleep probability* at each step, which needs to be at least $1/2$ in Luby's analysis. The empirical results of Finocchi [10] showed that Luby's algorithm improves by a constant factor on setting this probability to 0. Though this seems intuitive, it is non trivial to prove, and we give a proof in this section.

Let G denote the input graph. Luby's algorithm works in the following manner. Each vertex $u \in V(G)$ is initially given a list of colors $L(u)$; initially, $|L(u)| \geq \Delta + 1$, where Δ is the maximum degree in G . Vertices get colored using a distributed list-coloring algorithm in a synchronous round-by-round fashion (in a given round, any vertex communicates only with its neighbors). A generic round proceeds as follows.

(a) Each yet-uncolored vertex wakes up with probability w or goes to sleep with probability $1 - w$.

(b) Each vertex u which is awake, chooses a *tentative color* uniformly at random from its current list $L(u)$.

(c) Each vertex u that has some neighbor that chose the same tentative color as u , is called *unsuccessful*; all other (yet-uncolored) vertices are called successful.

(d) Each successful vertex v is permanently given its chosen tentative color c , and this color c is removed from $L(w)$ for all neighbors w of v such that $c \in L(w)$. The unsuccessful vertices proceed to the next round.

Note that once a vertex gets a permanent color, it is never considered again. Let d_u be the degree of u in the current round (in the graph induced by the yet-uncolored vertices). Let $L(u)$ be its current color-list. Then it can be easily verified that $|L(u)| \geq d_u + 1$. This also implies that step (b) is well defined; i.e., if u is yet-uncolored, $|L(u)| \geq 1$. It is also easy to verify that (if and) when the algorithm terminates, G has a valid vertex coloring. Let ϵ be an arbitrarily small constant and let n be sufficiently large. We now prove that for all graphs with n vertices, if $w = 1$, the above algorithm yields a valid vertex coloring within $f(n, \epsilon)$ rounds (where $f(n, \epsilon)$ is defined later). In addition, we prove that, if $w = 1/2$, there exists graphs with n vertices such that after $f(n, \epsilon)$ rounds, a large number of vertices remain uncolored with high probability.

Lemma 6.1 *Let $w = 1$. Let G be a graph with n nodes. Let $\epsilon > 0$ be an arbitrarily small constant. With high probability, G has a valid vertex coloring after $f(n, \epsilon)$ rounds where $f(n, \epsilon) = (1 + \epsilon) \frac{\log n}{\log 1/(1-(3/4)^4)}$.*

Proof Consider one round of the algorithm being applied to some instance. Let c be the expected fraction of the yet-uncolored vertices which are successful after the round. The following claim holds.

Claim 6.2 $c \geq (3/4)^4$.

The proof of this claim can be found in the appendix.

Let R_i be the number of yet-uncolored nodes after i rounds. The above claim and induction on i implies that $E[R_i] \leq (1 - (3/4)^4)^i n$. Letting $i = f(n, \epsilon)$, we have,

$$E[R_{f(n, \epsilon)}] \leq (1 - (3/4)^4)^{(1+\epsilon) \frac{\log n}{\log 1/(1-(3/4)^4)}} \cdot n = n^{-\epsilon}$$

Since $R_{f(n, \epsilon)}$ is a non-negative integer valued random variable, we have

$$Pr[R_{f(n, \epsilon)} = 0] \geq 1 - E[R_{f(n, \epsilon)}] \geq 1 - 1/n^\epsilon$$

This completes the proof of lemma 6.1. ■

Lemma 6.3 *Let $w = 1/2$; Let K_n be the complete graph with n nodes. Let $\epsilon > 0$ be an arbitrary constant. There exists a constant n_0 such that, for all $n \geq n_0$, with high probability, after $f(n, \epsilon)$ rounds, the number of yet-uncolored vertices is at least $n^{k(\epsilon)}$, where $k(\epsilon)$ is a constant dependent only upon ϵ .*

Proof Let R_i be the number of yet-uncolored vertices after i rounds of the algorithm. Let δ be an arbitrarily small constant. Then, the following claim holds.

Claim 6.4 *There exists a constant n_0 such that, for all $n \geq n_0$,*

$$Pr[R_i \leq n(1 - \delta - 1/2\sqrt{\epsilon})^i] \leq 8\sqrt{\epsilon}/(\delta^2 n(1 - \delta - 1/2\sqrt{\epsilon})^i)$$

The proof of this claim can be found in the appendix.

Letting $i = f(n, \epsilon)$ in the above equation we have

$$Pr[R_{f(n, \epsilon)} \leq n^{k(\epsilon)}] \leq 8\sqrt{\epsilon}/\delta^2 n^{k(\epsilon)}$$

where, $k(\epsilon) = 1 - (1 + \epsilon) \log(1/(1 - \delta - 1/2\sqrt{\epsilon}))/\log(1/1 - c)$. ■

7 Fast Distributed Graph Decompositions

A λ -decomposition of a graph $G = (V, E)$ is a partition of the vertex set into λ subsets (called *blocks*). The *diameter* of a decomposition is the least d such that any two vertices belonging to the same connected component of a block are at distance $\leq d$. In the distance computation, if we allow the paths between two vertices in a block to pass through other vertices which are in the same block, then the diameter is said to be *weak*; otherwise, the diameter is said to be the *strong*. In [16], Linial and Saks show that for any graph G and any constant $p \in (0, 1)$, there exists a λ -decomposition of G such that the weak diameter of each block is at most d , where $\lambda, d = O(\log n)$. In addition, they provide a distributed algorithm which constructs such a decomposition and terminates in $O(\lambda d)$ time with high probability. We first describe their graph decomposition algorithm and then present our modification to this algorithm. Our modification yields the same guarantees in terms of the number of blocks and the block diameter, with improved time and message complexity. In the discussion below, we assume that each node u has a unique ID, ID_u .

7.1 Linial-Saks Graph Decomposition algorithm

The Linial-Saks graph decomposition algorithm comprises of several rounds. At the end of each round, one block gets constructed. A generic round proceeds as follows. Each remaining vertex y selects an integer radius r_y at random (according to a distribution given below, which is approximately geometric). It then broadcasts (ID_y, r_y) to all vertices which are within distance r_y from it. After collecting all such messages from other vertices, each vertex y selects the vertex $C(y)$ of highest ID from among the vertices whose broadcast it received in the first round (including itself), and joins the current block if $d(y, C(y)) < r_{C(y)}$ (note that it is necessarily the case that $d(y, C(y)) \leq r_{C(y)}$). All vertices which joined the current block are removed from the current set of vertices.

The distribution by which each vertex x selects its radius r_x is a truncated geometric distribution, which is defined in terms of two parameters, p and B :

$$Pr[r_x = j] = \begin{cases} p^j(1-p) & j \leq 0, 1, \dots, B-1 \\ p^B & j = B \end{cases}$$

The following lemmas are proved in [16]. Let G be any graph with at most n vertices and let S be the set of vertices selected at the end of one round of the algorithm applied to G .

Lemma 7.1 *The weak diameter of S is at most $2B$.*

Lemma 7.2 *For each vertex y of G , the probability that it belongs to S is at least $p(1-p^B)^n$.*

The above lemmas yield the following lemma.

Lemma 7.3 *Let $w(n)$ be any function which tends to infinity with n . Letting $B = \log n + w(n)/\log(1/p)$, with high probability, the above algorithm results in a λ -decomposition with diameter D where $\lambda = (1 + o(1)) \log n / \log 1/(1 - p)$ and $D \leq 2B$. In addition, the running time of the algorithm is $O(D\lambda)$.*

7.2 Our modification to Linial-Saks

We now describe our modification to the Linial-Saks algorithm. The main idea here is to parallelize the construction of all the blocks, by “simulating” the λ rounds of the previous algorithm within a single round of our algorithm. At the beginning of this round, every vertex y , chooses λ radii independently at the random from the same distribution as that of Linial-Saks. Let these radii be $r_{y,1}, r_{y,2}, \dots, r_{y,\lambda}$ respectively. It then broadcasts $(ID_y, r_{y,1}, r_{y,2}, \dots, r_{y,\lambda})$ to all vertices which are within distance B from it. After collecting all such messages from other vertices, each vertex y attempts to join one of the blocks in $\{1, 2, \dots, \lambda\}$, in that order. While trying to join block i , y considers all radii $r_{x,i}$, such that $r_{x,i} \geq d(x, y)$. The requirements for choosing a leader and for joining this block are the same as before. Among all nodes x which sent y a message such that $r_{x,i} \geq d(x, y)$, the current leader of y is the node with highest ID. Let the leader be $C(y)$. y joins block i if $d(y, C(y)) < r_{C(y),i}$; otherwise y tries to join block $i + 1$.

Recall that r_x denote the radius chosen by x in the first round of the Linial-Saks algorithm. Let $p(y_i)$ be the probability of y joining block i in our algorithm, given that y did not join blocks $1, \dots, i - 1$ in our algorithm. Let $p'(y_i)$ be the probability of y joining block i in the Linial-Saks algorithm, given that y did not join blocks $1, \dots, i - 1$ in the Linial-Saks algorithm. Our first observation is that, for all i , $p(y_i)$ has the same value. This follows from the fact that any node x chooses all its radii from the same distribution. Our next observation is that, $p(y_1) = p'(y_1)$. This follows from the fact that, for the first round in both the algorithms, the joint distribution of the radii chosen by all the nodes is identical. These two observations lead to the following lemmas.

Lemma 7.4 *The weak diameter of any block constructed in our algorithm is at most $2B$.*

Lemma 7.5 *For any vertex y , $p(y_i) = p(1 - p^B)^n$.*

The above lemmas yield the following lemma.

Lemma 7.6 *Let $w(n)$ be any function which tends to infinity with n . Letting $B = \log n + w(n)/\log(1/p)$, our algorithm results in a λ -decomposition with diameter D where $\lambda = (1 + o(1)) \log n / \log 1/(1 - p)$ and $D \leq 2B$ (same as the previous algorithm). However, the running time of our algorithm is always $O(D)$.*

Corollary 7.7 *Any graph G can be decomposed by our algorithm into λ blocks with weak diameter D in $O(D)$ time where, $\lambda = O(\log n)$ and $D = (\log n)$.*

8 Empirical Analysis

We study two aspects of the packet scheduling problem empirically. The first set of experiments deal with the effect of the routing strategy on the congestion. The routing strategies we tried are: (i) Shortest path routing, (ii) Valiant’s paradigm [26], and (iii) Modified Source Routing (SR). The last strategy is an adaptation of the Source Routing algorithm proposed in [3]. Routes are chosen sequentially for each packet using a weighted shortest path algorithm. After each path is chosen, the weights for the edges along this path (and for some of edges which are two hops away from the path) are increased by a multiplicative factor (1.25 in our experiments). This ensures that no edges (or regions in the network) gets overloaded by too many packets.

The second set of experiments deal with the sensitivity of our algorithms to various parameters: (i) the maximum initial random delay for any packet (mrd), (ii) the maximum number of colors available for the distributed edge coloring algorithm (mc), and (iii) the maximum number of rounds in the distributed coloring algorithm (nt). During a particular stage of the distributed algorithm, if a packet needs to be transmitted on some edge, and if this edge cannot be colored after nt rounds of the coloring algorithm, then the packet gets dropped. We study the sensitivity of packet loss to mrd and mc . The routing is done by SR, with $nt = 15$. The number of packets injected into the network was varied from 156 to 10000. All our experiments were performed on random connected unit disk graphs obtained by a random placement of 10000 nodes in a 50×50 square. The source and destination for each packet was chosen randomly from all nodes.

Impact of Routing Figures 5 (a) and (b) show the congestion and dilation in the network with respect to the number of packets in the network for the three routing algorithms. Source Routing, which is specifically tailored for reducing congestion, performs far better than either of the two strategies. Interestingly, Valiant’s paradigm which is a provably good routing algorithm for hypercube networks, incurs about twice the congestion as shortest paths. We believe, this due the fact sources and destinations are chosen random from all nodes in our experiments. Hence, the advantage of choosing a random intermediate node (which may reduce congestion for fixed source-destination pairs), is not applicable any longer. Notice how the dilation varies for each strategy. After a certain number of packets, the dilation stabilizes at a constant value for all three routing algorithms. Naturally, shortest path has the least dilation of the three. Valiant’s algorithm has about twice the dilation of shortest paths. This is along expected lines, since the since packets go through a random intermediate node. Source Routing has the maximum dilation of the three. While trying to route around the heavily congested regions in the network, Source

Routing incurs additional costs in terms of the dilation. However, this cost is amply compensated by the much bigger gains incurred by Source Routing with respect to congestion.

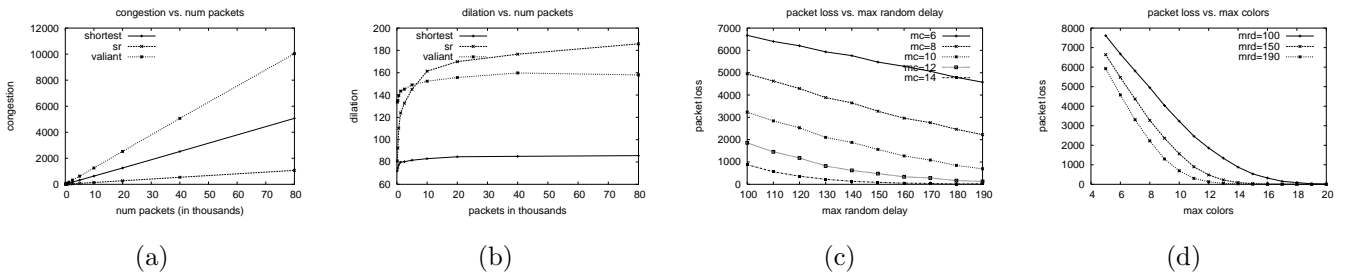


Figure 5: (a) and (b): Congestion and Dilation for different routing strategies. (c) and (d): Packet loss vs. mrd and mc

Sensitivity to Parameters Recall that mrd and mc are the maximum initial random delay and the maximum number of colors used during distributed coloring respectively. Figures 5 (c) and (d) show the effect of mrd and mc on the total number of packets lost. For a fixed value of mc , packet loss decreases linearly with increasing values of mrd . On the other hand, for a fixed value of mrd , packet loss seems to decrease exponentially with increasing values of mc . In particular, for the range of values plotted here, doubling the value of mc yields a substantial reduction in packet loss than doubling the value of mrd . Clearly, this has important implications in practice. Increasing either of the two parameters increases the latency incurred by packets. However, in order to reduce packet loss, for a large range of values of mc and mrd , it is better to increase mc than mrd .

Acknowledgments. We thank Hari Balakrishnan, Subhash Khot and Alessandro Panconesi for helpful discussions.

References

- [1] N. Alon, F. Chung and R. Graham. Routing permutations on graphs via Matchings. *SIAM Journal of Discrete Mathematics*, 7, pp. 513-530, 1994.
- [2] M. Adler and C. Sheideler. Efficient Communication Strategies for Ad-hoc Networks. *Proc. ACM Symposium on Parallel Algorithms and Architectures*, pp. 259-268, 1998.
- [3] M. Andrews, A. Fernandez, A. Goel and L. Zhang. Source Routing and Scheduling in Packet Networks. *Proc. IEEE Symposium on Foundations of Computer Science*, 2001.

- [4] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Low-diameter graph decomposition is in NC . *Random Structures & Algorithms*, 5:441–452, 1994.
- [5] B. Awerbuch, A. V. Goldberg, M. Luby, and S. A. Plotkin. Network decomposition and locality in distributed computation. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 364–369, 1989.
- [6] B. Awerbuch and D. Peleg. Routing with polynomial communication-space trade-off. *SIAM J. on Discrete Mathematics*, 5:151–162, 1992.
- [7] H. Balakrishnan, C. Barrett, V.S. Anil Kumar, M. Marathe, S. Thite. Induced Matchings and its Relationship to Maximum Instantaneous Capacity of Media Access Layer, manuscript.
- [8] C. Barrett, G. Istrate, V. S. Anil Kumar, M. Marathe and S. Thite. Algorithms for link scheduling in wireless radio networks. Manuscript.
- [9] D. Dubhashi, A. Mei, A. Panconesi, J. Radhakrishnan, and A. Srinivasan. Fast Distributed Algorithms for (Weakly) Connected Dominating Sets and Linear-Size Skeletons. *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 717–724, 2003.
- [10] I. Finocchi, A. Panconesi and R. Silvestri. Experimental analysis of simple, distributed vertex coloring algorithms. *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pp. 606–615, 2002.
- [11] D. A. Grable and A. Panconesi. Fast distributed algorithms for Brooks-Vizing colorings. *J. Algorithms*, 37:85–120, 2000.
- [12] S. O. Krumke, M. V. Marathe, and S. S. Ravi. Models and approximation algorithms for channel assignment in radio networks. *Wireless Networks*, 7:575–584, 2001.
- [13] T. Leighton, B. Maggs and S. Rao. Packet Routing and Job Shop Scheduling in $O(\text{Congestion} + \text{Dilation})$ Steps. *Combinatorica*, v14, 2, pp. 167-180, 1994.
- [14] T. Leighton, B. Maggs and A. Richa. Fast algorithms for finding $O(\text{Congestion} + \text{Dilation})$ packet routing schedules. *Combinatorica*, pp. 1-27, 1999.
- [15] T. Leighton, B. Maggs, A. Ranade and S. Rao. Randomized rounding and sorting on fixed-connection networks. *Journal of Algorithms*.
- [16] N. Linial and M. Saks. Low Diameter Graph Decompositions. *Combinatorica*, 13:441–454, 1993.

- [17] M. Luby. Removing randomness in parallel computation without a processor penalty. *JCSS*, 47(2) 1993.
- [18] F. Meyer auf der Heide and B. Vöcking. A packet routing protocols for arbitrary networks. *LNCS*, Vol. 439, 291-302, 1995.
- [19] R. Ostrovsky and Y. Rabani. Universal $O(\text{congestion} + \text{dilation} + \log^{1+\epsilon} N)$ local control packet switching algorithms. *Proc. 29th Symposium on the Theory of Computation*, pp. 644-653, 1997.
- [20] A. Panconesi and A. Srinivasan. On the Complexity of Distributed Network Decomposition. *J. Algorithms*, 20:356-374, 1996.
- [21] L. Peterson and B. Davis. *Computer Networks: a systems approach*. Morgan Kaufman publishers.
- [22] S. Ramanathan. Scheduling Algorithms for Multi-Hop Radio Networks. Ph.D. thesis, Department of Computer Science, University of Delaware, Newark, DE, 1993.
- [23] S. Ramanathan. A Unified Framework and Algorithm for (T/F/C) DMA Channel Assignment in Wireless Networks, in *Proc. IEEE INFOCOM'97*, Kobe, Japan, April 1997.
- [24] S. Ramanathan and E. Lloyd. Scheduling algorithms for multi-hop radio networks. *IEEE/ACM Transactions on Networking*, 1:166-172, 1993.
- [25] Y. Rabani and E. Tardos. Distributed packet switching in arbitrary networks. *28th ACM Symposium on the Theory of Computing*, 366-375, 1996.
- [26] C. Scheideler. Universal routing strategies for interconnection networks, *Lecture Notes in Computer Science*, v 1390, Springer Verlag.
- [27] A. Srinivasan and C-P. Teo. A constant factor approximation algorithm for packet routing, and balancing local vs. global criteria. *Proceedings of the ACM Symposium on the Theory of Computing*, pp. 636-643, 1997.
- [28] L. Trevisan. Non-approximability Results for Optimization Problems on Bounded Degree Instances. *Proceedings of the ACM Symposium on Theory of Computing*, pp. 453-461, 2001.

Appendix

A Proof of claim 6.2

Let p_u and p'_u indicate the probability of vertex u being successful and unsuccessful respectively, in the current round. Vertex u is said to be a *high degree vertex* if $d_u \geq 3$. Suppose u has currently a list of size d . We refer to these colors as $1, 2, \dots, d$. u has currently at most $d - 1$ yet-uncolored neighbors v_1, v_2, \dots, v_{d-1} . Let $p_{i,c}$ be the probability that v_i picks c as its tentative color. Then, it is easy to see that

$$\begin{aligned} p'_u &= \sum_{c=1}^d \frac{1}{d} \cdot [1 - \prod_{i=1}^{d-1} (1 - p_{i,c})] \\ &= 1 - \frac{1}{d} \cdot \sum_{c=1}^d \prod_{i=1}^{d-1} (1 - p_{i,c}) \end{aligned} \tag{1}$$

$$p'_u \leq \frac{1}{d} \sum_{c=1}^d \sum_{i=1}^{d_u} p_{i,c} \leq \sum_{i=1}^{d_u} \max_c p_{i,c} \tag{2}$$

$$p'_u \leq \frac{1}{d} \sum_{i=1}^{d_u} \sum_{c=1}^d p_{i,c} \leq \frac{d_u}{d} \tag{3}$$

We will now use the above expressions to lower bound p_u . Vertex u may belong to one of the following cases.

Case 1: $d_u = 1$. In this case, (3) implies that $p'_u \leq 1/2$. Hence, $p_u \geq 1/2$.

Case 2: $d_u = 2$. In this case, (3) implies that $p'_u \leq 2/3$. Hence, $p_u \geq 1/3$.

Case 3: $d_u = 2$. Also, vertex u has two high degree neighbors. (2) implies that $p'_u \leq 2 \cdot \frac{1}{4} = \frac{1}{2}$. Hence, $p_u \geq 1/2$.

Case 4: $d_u = 2$. u has one high degree neighbor and another of degree ≥ 2 . (2) implies that $p'_u \leq \frac{1}{4} + \frac{1}{3} = \frac{7}{12}$. Hence, $p_u \geq \frac{5}{12}$.

Case 5: Vertex u is a high degree node. Recall equation (1). We imagine an adversary who wishes to minimize $\sum_{c=1}^d \prod_{i=1}^{d-1} (1 - p_{i,c})$ and aim to show that this minimum cannot be “too small”. What constraints does the adversary have? They are:

$$(C1) \quad \forall(i, c), p_{i,c} \geq 0.$$

$$(C2) \quad \forall i, \sum_c p_{i,c} \leq 1.$$

$$(C3) \quad \forall(i, c), p_{i,c} \leq 1/2, \text{ since each } v_i \text{ has a current list of size at least 2.}$$

Let us study some properties of a vector of values $p_{i,c}$ that minimizes the adversary’s objective function, subject to the above constraints. (A basic theorem from analysis asserts that a minimum indeed exists, which we assume.) First of all, the adversary might as well replace (C2) by: “(C2’): $\forall i, \sum_c p_{i,c} = 1$ ”: given any feasible solution to the above system of constraints, we can easily add some values to some of the $p_{i,c}$ to ensure (C2’), without increasing the optimal objective value OPT . We will employ the following useful claim:

Claim A.1 Suppose we are given some solution with objective value OPT , where some $p_{i,c}$ is neither 0 nor $1/2$; then we can transform this to another solution, also with objective value OPT , which has a lesser number of $p_{i,c}$ that are neither 0 nor $1/2$.

Proof Suppose we have an optimal solution in which, say, $p_{i,c} \in (0, 1/2)$. Then, constraints (C1), (C2') and (C3) show that there must exist some $c' \neq c$ such that $p_{i,c'} \in (0, 1/2)$. Now, consider simultaneously adding a value α to $p_{i,c}$ and subtracting α from $p_{i,c'}$, and let $g(\alpha)$ be the net increase in the objective function from this change. It is easy to check that $g(\alpha)$ is a linear function of α with $g(0) = 0$. Thus, one of the following two conditions should hold: (A1) $g(\alpha) \leq 0$ for all $\alpha \geq 0$, or (A2) $g(\alpha) \leq 0$ for all $\alpha \leq 0$. If (A1) holds, we will get an objective function no greater than OPT by increasing $p_{i,c}$ and decreasing $p_{i,c'}$ (the increase and decrease being by the same value), until one of them hits 0 or $1/2$. Similarly, if (A2) holds, we will get an objective function no greater than OPT by increasing $p_{i,c'}$ and decreasing $p_{i,c}$ (the increase and decrease being by the same value), until one of them hits 0 or $1/2$. This concludes the proof of claim A.1. ■

We can repeat the argument in claim A.1 a finite number of times to ensure that all the $p_{i,c}$ lie in $\{0, 1/2\}$. Now, in such a solution, let $f(c)$ be the number of i for which $p_{i,c} = 1/2$; thus, $OPT = \sum_c (1/2)^{f(c)}$. We can check using our above system of constraints that $\sum_c f(c) = 2(d-1)$. Now, for any two integers c_1 and c_2 such that $c_1 \leq c_2 - 2$,

$$(1/2)^{c_1} + (1/2)^{c_2} \geq (1/2)^{c_1+1} + (1/2)^{c_2-1}.$$

So, since $\sum_c f(c) = 2(d-1)$, the value $OPT = \sum_c (1/2)^{f(c)}$ is minimized when all the $f(c)$ are "as equal as possible". This happens when $d-2$ of the colors c have $f(c) = 2$, and the remaining two colors c have $f(c) = 1$. Thus, $\sum_c (1/2)^{f(c)} \geq (d-2)/4 + 1 > d/4$. Plugging back into (1), we see that $p'_u \leq 1 - 1/4 = 3/4$. Hence, $p_u \geq 1/4$.

Case 6: Vertex u is a high degree node with only high degree neighbors. In this case, using arguments similar to the previous case, we can prove that $p_u \geq (3/4)^4$.

To complete the proof, we note that there always exists a partition of the set of vertices into subsets such that the expected fraction f of successful nodes in any subset is at least c . In fact, it is easy to see that there exists a partition such that each subset belongs to one of the following types of subsets. We note that the success probabilities of vertices occurring in these subsets have been computed earlier (see cases 1 to 6 above).

Type A: Two nodes u and v such that $d_u \geq 3$ and $d_v = 1$. In this case, $f = (p_u + p_v)/2 \geq (\frac{1}{4} + \frac{1}{2})/2 \geq c$.

Type B: Three nodes u, v and w such that v is connected to u and w ; $d_v = 2$; and $d_u, d_w \geq 3$. In this case, $f \geq (2 \cdot \frac{1}{4} + \frac{1}{2})/3 \geq c$.

Type C: Three nodes u, v and w such that v is connected to u and w ; $d_v = 2$; $d_u \geq 3$; and $d_w = 1$. In this case, $f \geq (\frac{1}{4} + \frac{1}{3} + \frac{1}{2})/3 \geq c$.

Type D: Two nodes u and v such that $d_u \geq 3$ and $d_v = 2$; v is connected to u and another neighbor whose degree is 2. In this case, $f \geq (\frac{1}{4} + \frac{5}{12})/2 \geq c$.

Type E: Node u such that u is a high degree node and connected only to other high degree nodes. In this case, $f \geq c$.

Type F: Three nodes u , v and w such that v is connected to u and w ; $d_v = 2$; $d_u, d_w = 1$. In this case, $f \geq (\frac{1}{2} + \frac{1}{2})/3 \geq c$.

Type G: Two nodes u and v such that $d_u = 1$; $d_v = 2$; v is connected to u and another neighbor whose degree is atleast 2. We have, $f \geq (\frac{1}{2} + \frac{1}{3})/2 \geq c$.

Type H: Node u such that $d_u = 2$ and u is connected only to nodes of degree atleast 2. We have, $f \geq 1/3 \geq c$.

Type I: Node u such that its degree is 1 or 0. In this case, $f \geq \frac{1}{2} \geq c$.

This completes the proof of claim 6.2.

B Proof of claim 6.4

Consider the complete graph K_d where all nodes have the same color-list of size $d+1$. Let s be the random variable denoting the number of successful nodes after one round of the algorithm applied on K_d . Let μ and σ be the mean and the standard deviation of s . It is easy to see that,

$$\mu = d \cdot \frac{1}{2} \cdot (1 - 1/2(d+1))^{d-1}$$

We note that $(1 - 1/2(d+1))^{d-1}$ is a decreasing function of d . In addition, $\lim_{d \rightarrow \infty} (1 - 1/2(d+1))^{d-1}$ exists and is equal to $1/\sqrt{e}$. Hence, for every constant δ , there exists a constant d_0 such that, if $d \geq d_0$,

$$\mu \leq d \cdot (\delta/2 + 1/2\sqrt{e})$$

Let X_u be the indicator random variable for node u being successful in this round. Symmetry implies that for all u , $E[X_u] = E[X_1]$. In addition, for all distinct pairs u and v , $E[X_u X_v] = E[X_1 X_2]$. Hence,

$$\begin{aligned}
\sigma^2 &= E[s^2] - (E[s])^2 \\
&= \sum_{\substack{u,v=d \\ u,v=1}} E[X_u X_v] - E[X_u]E[X_v] \\
&\leq d(E[X_1^2] - E[X_1]^2) + 2 \cdot \binom{d}{2} (E[X_1 X_2] - E[X_1]E[X_2]) \\
&\leq d(Pr[X_1 = 1] - Pr[X_1 = 1]^2) + 2 \cdot \binom{d}{2} Pr[X_1 = 1](Pr[X_2 = 1|X_1 = 1] - Pr[X_2 = 1]) \\
&\leq d/4 + 2 \cdot \binom{d}{2} (Pr[X_2 = 1|X_1 = 1] - Pr[X_2 = 1]) \\
&\leq d/4 + 2 \cdot \binom{d}{2} (1/2 \cdot (1 - 1/2d)^{d-2} - 1/2 \cdot (1 - 1/2(d+1))^{d-1}) \\
&\leq d/4 + \binom{d}{2} ((1 - 1/2d)^{d-2} - (1 - 1/2(d+1))^{d-1}) \\
&\leq d/4 + \binom{d}{2} ((1 + 1/d)(1 - 1/2d)^{d-1} - (1 - 1/2(d+1))^{d-1}) \\
&\leq d/4 + d(d-1)(1/2)(1/d)(1 - 1/2d)^{d-1} \\
&\leq d/4 + d/2 \leq d \\
\sigma &\leq \sqrt{d}
\end{aligned}$$

By Chebychev's inequality, we have

$$\begin{aligned}
Pr[s \geq d(\delta + 1/2\sqrt{e})] &\leq Pr[|s - \mu| \geq \delta d/2] \\
&\leq Pr[|s - \mu| \geq \delta\sqrt{d}\sigma/2] \\
&\leq 4/\delta^2 d
\end{aligned}$$

Let R_i be the number of nodes remaining after i rounds of the algorithm. The above inequality along with induction on i implies that

$$\begin{aligned}
Pr[R_i \leq d(1 - \delta - 1/2\sqrt{e})^i] &\leq \sum_{j=0}^{i-1} 4/(\delta^2 d(1 - \delta - 1/2\sqrt{e})^j) \\
&\leq 8\sqrt{e}/(\delta^2 d(1 - \delta - 1/2\sqrt{e})^i)
\end{aligned}$$

This concludes the proof of claim 6.4.