Safe and Stabilizing Distributed Multi-Path Cellular Flows

Taylor T. Johnson[a,*], Sayan Mitra[b]

[a]Computer Science and Engineering, University of Texas at Arlington, Arlington, TX 76019, USA
[b]Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Abstract

We study the problem of distributed traffic control in the partitioned plane, where the movement of all entities (robots, vehicles, etc.) within each geographic partition (cell) is the same. Each cell is controlled by software to move entities across the cell to route entities from sources to targets without collisions. We present a formal model of a distributed traffic control protocol that guarantees minimum separation between entities, even as the software controlling some cells fails by crashing. The distributed traffic control protocol relies on two principles: (a) temporary blocking entity transfers between adjacent cells for maintenance of safety and (b) local geographical routing for guaranteeing progress of entities to their targets. Establishing liveness in distributed traffic control systems is challenging, but liveness analysis will be necessary to apply distributed algorithms in applications like coordinating robot swarms and intelligent highway systems. Once new failures stop occurring, in the case of a single target cell, the protocol is guaranteed to self-stabilize and entities with feasible paths to the target cell make progress towards it. For multiple targets, failures may cause deadlocks in the system, so we identify a class of non-deadlocking failures where all entities are guaranteed to make progress to their respective targets. Our assertional proofs may serve as templates for the analysis of other distributed traffic control protocols. We also present simulation results to validate the formal model, and to provide estimates of entity throughput as a function of entity velocity, safety separation distance, single-target path complexity, failure-recovery rates, and multi-target path complexity.

Keywords: distributed systems, swarm robotics, formal methods, traffic control, liveness

*Corresponding author
Email addresses: taylor.johnson@acm.org (Taylor T. Johnson), mitras@illinois.edu (Sayan Mitra)

1. Introduction

Highway and air traffic flows are nonlinear switched dynamical systems that give rise to complex phenomena such as abrupt phase transitions from fast to sluggish flow [1, 2, 3]. Our ability to monitor, predict, and avoid such phenomena can have a significant impact on the reliability and capacity of physical traffic networks. Traditional traffic protocols, such as those implemented for air traffic control are *centralized* [4], where a *coordinator* periodically collects information from the vehicles, decides and disseminates waypoints, and subsequently the vehicles try to blindly follow a path to the waypoint. Wireless vehicular networks [5, 6, 7, 8] and autonomous vehicles [9, 10] present new opportunities for *distributed* traffic monitoring [11, 12, 13] and control [14, 15, 16, 17, 18, 19]. While these protocols may still rely on some centralized coordination, they should scale and be less vulnerable to failures compared to their centralized counterparts. In this paper, we propose a fault-tolerant distributed traffic control protocol, formally model it, and prove its correctness, namely that it satisfies certain safety and progress properties.

A *traffic control protocol* is a set of rules that determines the routing and movement of certain physical *entities*, such as vehicles, robots, or packages, over an underlying *graph*, such as a road network, air-traffic network, or warehouse conveyor system. Any correct traffic control protocol guarantees:

(a) *safety*: that the entities always maintain some minimum physical separation, and

(b) *progress*: that the entities eventually arrive at a given a destination (or target) vertex.

In a distributed traffic control protocol, each entity determines its own next waypoint, or each vertex in the underlying graph determines the next waypoints for the entities in an appropriately defined neighborhood.

In this paper, we study the problem of distributed traffic control in a partitioned plane where the motions of entities within a partition are *identically coupled*. The problem is as follows (refer to Figures 1 and 2). The *environment* is the geographical space of interest and is partitioned into regions or *cells*. Each entity is assigned to a certain finite type or *color*. For each color, we assume for ease of exposition that there is one *source cell* and one *target cell* of the same color. A source cell of color c produces entities of color c, and a target cell of color c only consumes entities of color c, so the traffic control progress goal is to move entities of color c to the target of color c. The motion of all entities within a cell is the same, in the sense that they all either move identically, or they all remain stationary (we discuss the motivation for this below). If some entities within some cell i touch the boundary of a neighboring cell j, those entities are transferred to cell j. Thus, the role of the distributed traffic control protocol is to control the cell motion so that: (a) entities always have the required safe separation, and (b) entities eventually reach their respective targets, when feasible.

The identical coupling requirement mentioned above—that entities within a cell move identically—may appear strong at first sight. After all, under low traffic conditions, individual drivers control the movement of their cars within a particular region of the highway, somewhat independently of other drivers in that region. However, on highways under high-traffic, high-velocity conditions, it is known that similar coupling may emerge spontaneously, causing the vehicles to form a fixed lattice structure and move with near-zero relative speed [1, 20]. In the steady-state, highway vehicles have emergent behavior that could be modeled in a manner similar to the system presented here, as there are various regions on the highway where all vehicles are traveling at the same speed. In other scenarios, coupling arises because passive entities are moved around by active cells. For example, this occurs with packages being routed on a grid of multi-directional conveyors [21, 22], or molecules moving on a medium according to some controlled chemical gradient.

Even where the entities are active and cells are not, the entities can cooperate to emulate a virtual active cell expressly for the purposes of distributed coordination. This idea has been explored for mobile robot coordination in [23] using a cooperation strategy called *virtual stationary automata* [24, 25]. Within robotics, discrete abstractions of the environment in which the robots operate are commonly used [26, 27, 28]. Typically, the robots are assumed to operate according to the same dynamics within a given cell, under the assumption there is a single robot in each cell. Our framework is more general in that it allows multiple entities per cell, but with the restriction that the entities move identically. Our framework covers both scenarios where the entities are passive and the cells are active (such as in warehouse conveyance) and where the entities

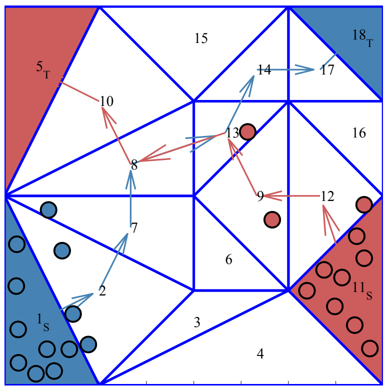

Figure 1: Source cells (1 and 11) produce entities that flow toward the target cell (18 and 5) of the appropriate color. Source-to-target paths overlap at cells 8 and 13. In this execution, the blue entity on cell 7 is waiting until no red entities are on the overlapping cells (8 and 13).

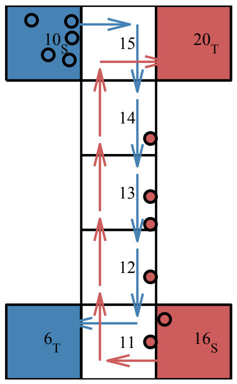

Figure 2: If cell 10 moves its entities onto the shared one-lane "bridge" (cells 11, 12, 13, 14, 15), then all entities would be deadlocked.

are active and the cells are passive, but the entities coordinate to move identically (such as in some robot coordination). Alternative frameworks such as where the entities move completely independently are beyond the scope of this framework, but would require coordination between entities to ensure safety.

The distributed traffic control protocol we present in this paper guarantees *safety at all times*, even as some cells fail permanently by crashing or transiently by crashing and recovering. The protocol also guarantees *eventual progress* of entities toward their targets, provided that: (a) there exists a path through non-faulty cells to the entities' respective targets, and (b) failures have not introduced unrecoverable deadlocks. Specifically, the protocol is *self-stabilizing* [29, 30], in that once new failures stop occurring, the composed system automatically returns to a state from which progress can be made. The distributed traffic control protocol relies on the following four mechanisms to enable safety and eventual progress of entities to their targets.

(a) A *routing* rule maintains local routing tables to each target at each non-faulty cell. This routing protocol is self-stabilizing and allows the protocol to tolerate these cell failures.

(b) A *mutual exclusion and scheduling mechanism* ensures moving entities over distinctly colored overlapping paths do not introduce deadlocks. The locking and scheduling mechanism ensures one-way traffic can make progress over shared routes (traffic intersections).

(c) A *signaling* rule between neighboring cells guarantees safety while preventing deadlocks. Roughly speaking, the signaling mechanism at a cell fairly chooses among neighboring cells that contain entities, indicating if it is safe for one of these neighboring cells to apply a movement in the direction of the cell performing the signaling. This permission-to-move policy turns out to be necessary, because movement of neighboring cells may otherwise result in a violation of safety in the signaling cell, if transferal of entities between neighboring cells occurs.

(d) A *movement* policy causes all entities on a cell to either move with the same constant velocity in the direction of their destination, or remain stationary to ensure safety. This policy abstracts more complex motion modeling as discussed above.

We establish these safety and progress properties through systematic assertional reasoning. For safety properties, we establish inductive invariants, and for route stabilization properties, we use global ranking functions. To show that all entities reach their destinations (when feasible), we use a combination of ranking functions and fairness-based reasoning on infinite executions. These proof techniques may serve as templates for the analysis of other distributed traffic control protocols, where liveness is particularly challenging to reason about, and is exacerbated when sub-components may fail. Our analysis is generally independent of the size of the environment, number of cells, and number of entities. Additionally, only neighboring cells communicate with one another and the communication topology is fixed (aside from failures).

Throughput is roughly the average rate at which entities arrive at their target. We present simulation results that illustrate the influence (or the lack thereof) of several factors on throughput. (a) Throughput decreases exponentially with path length until saturation, as which point it decreases roughly linearly with path length. (b) Throughput decreases roughly linearly with required safety separation and cell velocity. (c) Throughput decreases roughly exponentially until it saturates as a function of path complexity measured in number of turns along a path. (d) Throughput decreases roughly exponentially with failure rate, and increases linearly with recovery rates, under a model where crash failures are not permanent and cells may recover from crashing. (e) Throughput decreases roughly exponentially until it saturates as a function of the percentage of overlapping cells between different colored targets. Overall, the simulator and simulations validate the formal model and theoretical claims, as it also employs runtime monitoring of the safety, liveness, and progress properties to check that they are satisfied over executions.

Contributions over Previous Work. We build on our previous work where we analyze a similar problem [31], but have significantly generalized our results in this paper.

(A) We consider general convex tessellations (including certain triangulations and rectangular tessellations) that define the partitioning, while we consider uniform square partitions in [31]. We also present results on partitioning schemes that cannot work for the problem formulation, as there are certain partitions that violate necessary geometric assumptions.

(B) We allow entities of multiple colors, each destined for a different target, while in [31], we only allow entities of one color, all of which were destined for the nearest target. This generalization lets different source-to-target paths overlap, creating traffic intersections, and requires several changes to the distributed traffic control protocol, including adding mutual exclusion and fair scheduling mechanisms used to control traffic intersections. This generalization is significant because it makes the problem and solution applicable to a much wider class of realistic systems, such as swarm robots and intelligent highway systems.

(C) We reimplemented the simulator [31] to account for these generalizations and extended our simulation results, in particular to characterize the cost on throughput due to the additional coordination required to allow multiple targets.

Paper Organization. The rest of the paper is organized as follows. First, Section 2 introduces the model of the physical environment and system. Next in Section 3, we present the distributed traffic control algorithm. Then in Section 4, we define and prove the safety and progress properties. Section 4.1 summarizes the formal results, then Section 4.2 formally establishes safety. Subsequently, we establish a progress property that shows entities eventually reach their targets in spite of failures (when possible). First in Section 4.3, it is shown that the routing protocol to find any target from any cell with a physical path through non-faulty cells to that target is self-stabilizing. Then in Section 4.4, we show how overlapping paths to different targets (traffic intersections) can be scheduled. Finally, in Section 4.5, it is shown that entities on any cell with a feasible physical path to their target eventually reach their target. Simulation results and interpretation are presented in Section 5, followed by related work, a discussion, and a conclusion, respectively in Sections 6, 7, and 8.

2. Overview and Physical System Model

In this section, we describe the physical environment and system model. For a set K, we define $K_\perp \triangleq K \cup \{\perp\}$ and $K_\infty \triangleq K \cup \{\infty\}$. For $N \in \mathbb{N}$, let $[N] \triangleq \{1, \ldots, N\}$. The $||\cdot||$ brackets are used for the Euclidean norm of a vector.

2.1. Cells and Partitioning

The system consists of N convex polygonal *cells* partitioning a planar polygonal environment. Let $ID \triangleq [N]$ be the set of unique identifiers for all cells in the system.The planar environment Env is some given simply connected polygon. A partition P of Env is a set of closed, convex polygonal cells $\{P_i\}_{i \in ID}$ such that:

(a) the interiors of the cells are pairwise disjoint,

(b) the union of the cells is the original polygonal environment, and

(c) cells only touch one another at a point or along an entire side.

The first two conditions are the standard definition of a partition, while the third restricts any cell from being adjacent along one of its sides to more than one other cell. For example, the definition of a polygon triangulation satisfies these requirements. Thus, cell i occupies a convex polygon P_i in the Euclidean plane. The boundary of cell i is denoted by ∂P_i. We denote the vertices (extreme points) of P_i as V_i. We denote the number of sides of P_i as $ns(i)$. Let $Side(i, j) \triangleq \partial P_i \cap \partial P_j$ be the common side of adjacent cells i and j—we will refer to $Side(i, j)$ as both (a) an index for referring to a given side and (b) a line segment (set of points).

2.2. Neighbors and Communications

Cell i is said to be a *neighbor* of cell j if the cells share a common side. The set of identifiers of all neighbors of cell i is denoted by $Nbrs_i$. This definition of neighbors naturally induces a graph $G = \langle V, E \rangle$ where the vertices V of the graph are the cells and there is an edge $e \in E$ between cells i and j if and only if cells i and j are neighbors. This graph represents a communication graph for the cells that may communicate directly with one another, and is connected without failures. Let Δ be the worst-case diameter of this graph[1]. For each cell $i \in ID$ and each neighboring cell $j \in Nbrs_i$, let the *side normal vector* from i to j, denoted $n(i, j)$, be the unit vector orthogonal to $Side(i, j)$ and pointing into cell j from the common side $Side(i, j)$.

Each cell is controlled by software that implements the distributed traffic control protocol described in the next section. We consider synchronous protocols that operate in rounds. At each round, each cell exchanges messages bearing state information with its neighbors. Then, each cell updates its software state and decides the (possibly zero) velocity with which to move any entities on it. Until the beginning of the next round, the cells continue to operate according to this velocity, which may lead to entity transfers.

2.3. Entities and Colors

Each cell may contain a number of *entities*. Each entity occupies a circular area and represents a physical object (or overapproximation thereof) such as an aircraft, car, robot, or package. Every entity that may ever be in the system has a unique identifier drawn from an index set I. This assumption is for presentation only, and the protocol does not rely on knowing entity identifiers. For an entity $p \in I$, we denote the coordinates of its center by $\overline{p} \triangleq (p_x, p_y) \in \mathbb{R}^2$.

The open circular area (disc) centered at $\overline{p}$ of radius r representing entity p is denoted $B(p, r)$. The radius of an entity is l, and r_s is the minimum required inter-entity safety gap. We define the total safety spacing radius as $d \triangleq r_s + l$. Looking ahead, the safety specification will ensure the centers of any entities come no closer than d apart from one another. For simplicity of presentation, we work with uniform entity radii l and safety gaps r_s. If they differ, we would take l and r_s to be the maximums over all entities. We instantiate $B(p, l)$, which represents the physical space occupied by entity p, and we also instantiate $B(p, d)$, which is entity p's total safety area.

Entity Colors, Source Cells, and Target Cells. There are $|C|$ types (or *colors*) of entities, where C is some finite, ordered set. The color of some entity $p \in I$ is denoted as $color(p)$. For each $c \in C$, there is a *source cell* sid_c and a *target cell* tid_c. All other cells are *ordinary cells*. For simplicity of presentation, we assume there is a unique source and target, but the algorithms and the results generalize for when sid_c and tid_c are sets. Entity p's color $color(p)$ designates the target cell entity p should eventually reach. The source sid_c produces entities of color c and the target tid_c consumes entities of color c. Entities appear at sources and disappear at targets. The sets of target and source identifiers are denoted $ID_T \subseteq ID$ and $ID_S \subseteq ID$, respectively.

[1]The diameter of this graph is not static, it may change due to failures, but the worst case is always a path graph, so $\Delta = N - 1$.

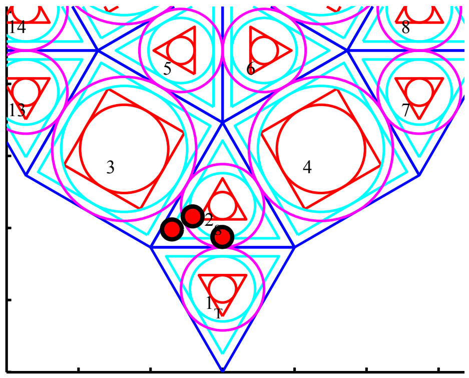

Figure 3: Safety regions (areas between red [innermost] and blue [outermost] polygons) and transfer regions (areas between cyan [middle] and blue [outermost] polygons) for the squares and triangles composing the snub square tiling tessellation. The magenta (outermost) circles are incircles used to compute the cyan (middle) transfer and red (innermost) safety regions.

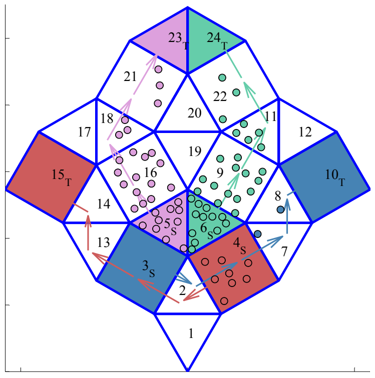

Figure 4: Blue and red paths overlap at cells 2, 3, and 4, corresponding to the color-shared cells definition. Blue entities on cells 7 and 8 have traversed the intersection and then the red source (4) produces entities. Red (cell 4) and blue (cell 3) sources producing entities simultaneously would cause a deadlock.

Entity Movement. All the entities within a cell move identically—either they remain stationary, or they move with some constant velocity $0 < v < l$ in the direction of one of the sides of the cell. Here v is the maximum cell velocity, or the greatest distance traveled by any entity over one synchronous round. We require $v > 0$ to ensure progress. We require that $v < l$ to ensure entities do not collide when transfers of entities between different cells occur. The actual cell velocity may differ in each cell so long as each is upper bounded by v. This movement is determined by the algorithm controlling each cell. When a moving entity touches a side of a cell, it is instantaneously transferred to the neighboring cell beyond that side, so that the entity is entirely contained in the new cell.

2.4. *Safety and Transfer Regions*

The safety region on side s of a cell is the area within the cell where new entities entering the cell from side s can be placed. For a side s of some cell i, the *safety region on side s* $SR_i(s)$ is the area on P_i at most $3d$ distance measured orthogonally from side s. Analogously, the transfer region on side s of a cell is the area within a cell where (the centers of) entities reside when those entities will be transferred to the neighboring cell on that side. The *transfer region on side s* is denoted $TR_i(s)$ and is the region in the partition P_i at most l distance measured orthogonally from side s. That is, $TR_i(s)$ and $SR_i(s)$ are respectively the set of points at most l or $3d$ distance from side s of P_i. For a cell i, the *transfer region* TR_i and *safety region* SR_i are respectively the unions of $TR_i(s)$ and $SR_i(s)$ for each side s of P_i. We refer to the *inner side(s)* of TR_i, $TR_i(s)$,

SR_i, or $SR_i(s)$, as the side(s) touching the inside of the annulus, and denote them by ITR_i, $ITR_i(s)$, etc.

For example, in Figure 3, the transfer region for the square cell 3 is the square annulus between the smaller cyan square and the larger blue square (the boundary ∂P_3 of cell 3). Similarly, for the triangular cell 1 in Figure 3, the transfer region is the triangular annulus between the smaller cyan triangle and the larger blue triangle. Thus, the distance measured orthogonally between the sides of the cyan polygons representing the boundary of the transfer region, and the sides of the blue polygons is always l. In Figure 3, for the square cell 3, the safety region is the square annulus between the smaller red square and the larger blue square.

2.5. Geometric Assumptions on Environment and Its Partition

We assume that the polygonal environment Env and its partition P have shapes and sizes such that each cell in the partition is large enough for an entity to lie completely on it. Particularly, we require for each cell $i \in ID$ that the transfer region TR_i is nonempty. We also make the following assumptions to ensure that the transfer of entities between cells is well-defined.

Assumption 1. *(Projection Property): For each $i \in ID$, for each side s of P_i, there exists a constant vector field over P_i that drives every point in P_i to some point on side s without exiting P_i.*

By definition, the cells form a partition. However, in part because there is "empty space" between the transfer regions of the cells, the transfer regions do not form a partition. Even if we remove this empty space by translating the transfer regions so the sides of transfer regions of neighboring cells coincide, they still may not form a partition. See Figure 3 for an example where the transfer regions cannot form a partition. This is because, for the shared side s of neighboring cells i and j, the inner sides of the transfer regions on P_i and P_j may have different lengths, even though the shared side s obviously had the same length for P_i and P_j. For this reason, we need the next assumption to ensure entity transfers between different cells are well-defined.

Assumption 2. *(Transfer Feasibility): For any $i \in ID$ and any $j \in Nbrs_i$, consider their common side $Side(i, j)$. The length of the inner side $ITR_i(Side(i, j))$ line segment equals the length of the inner side $ITR_j(Side(i, j))$ line segment.*

3. Distributed Traffic Control Protocol

We first give an informal overview of the distributed traffic control protocol. The protocol operates synchronously through a sequence of four operations: routing, locking, signaling, and moving. For routing, each cell communicates with its neighbors to determine the minimum path to each target, where ties are broken by cell identifiers. For locking, due to traffic intersections (see, e.g., Figures 5 and 6), part of the path to some target must be given exclusive access to the intersection, so a mutual exclusion protocol determines a lock that has this exclusive access. To ensure safety, a traffic signaling mechanism operates locally at each cell to give permission to neighboring cells to move toward it, much like a typical traffic light gives cars permission to enter an intersection. Finally, to move entities so they make progress toward their targets, the moving operation corresponds to the entity positions being physically updated due to the cell's velocity.

Next, we describe the *discrete transition system* Cell_i that specifies the software controlling an individual cell P_i of the partition P.

3.1. Preliminaries

A *variable* is a name with an associated type. For a variable x, its type is denoted by $\mathit{type}(x)$ and it is the set of values that x can take. A *valuation* (or state) for a set of variables X is denoted by $\mathbf{x}$, and is a function that maps each $x \in X$ to a point in $\mathit{type}(x)$. Given a valuation $\mathbf{x}$ for a set of variables X, the valuation for a particular variable $v \in X$ is denoted by $\mathbf{x}.v$, which is the restriction of $\mathbf{x}$ to $\{v\}$. The set of all possible valuations of a set of variables X is denoted by $val(X)$. Many variables return cell identifiers that we use to access variables of other cells using subscripts, and if the valuation of these variables are restricted to the same state, we will drop the particular state on the subscripted variables for more concise notation. For instance, suppose $\mathbf{x}.next_i \in ID$, then $\mathbf{x}.next_{\mathbf{x}.next_i}$ would be written $\mathbf{x}.next_{next_i}$.

A *discrete transition system* $\mathcal{A}$ is a tuple $\langle X, Q_0, A, \rightarrow\rangle$, where:

(i) X is a set of variables and $val(X)$ is set of the valuations of the variables, called the set of *states*,

(ii) $Q_0 \subseteq val(X)$ is the set of *start states*,

(iii) A is a set of *transition names*, and

(iv) $\rightarrow \subseteq val(X) \times A \times val(X)$ is a set of *discrete transitions*. For $(\mathbf{x}_k, a, \mathbf{x}_{k+1}) \in \rightarrow$, we also use the notation $\mathbf{x}_k \overset{a}{\rightarrow} \mathbf{x}_{k+1}$.

An *execution fragment* of a discrete transition system $\mathcal{A}$ is a (possibly infinite) sequence of states $\alpha = \mathbf{x}_0, \mathbf{x}_1, \ldots$, such that for each index k appearing in α, $(\mathbf{x}_k, a, \mathbf{x}_{k+1}) \in\rightarrow$ for some $a \in A$. An *execution* is an execution fragment with $\mathbf{x}_0 \in Q_0$. A state $\mathbf{x}$ is said to be *reachable* if there exists a finite execution that ends in $\mathbf{x}$. $\mathcal{A}$ is said to be *invariant* with respect to a set $S \subseteq val(X)$ if all reachable states are contained in S. A set S is said to be *stable* if, for each $(\mathbf{x}, a, \mathbf{x}') \in\rightarrow$, $\mathbf{x} \in S$ implies that $\mathbf{x}' \in S$. $\mathcal{A}$ is said to *self-stabilize* to S if S is stable and every execution fragment eventually enters S [32, 29, 33, 30].

3.2. Transition System for each Cell

We assume messages are delivered within bounded time and computations are instantaneous. Under these assumptions, the system can be modeled as a collection of discrete transition systems. The overall system is obtained by

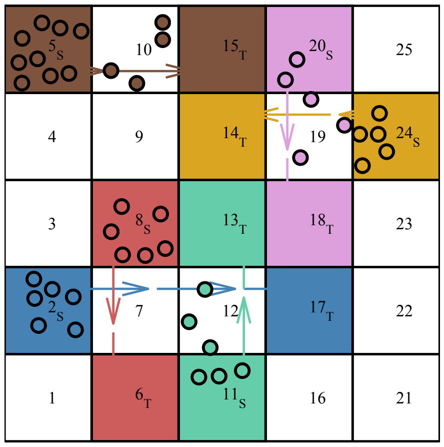

Figure 5: Example illustrating the computation of the color-shared cells $CSC(\mathbf{x}, c)$ and shared colors $SC(\mathbf{x}, c)$, stored in the $pint[c]$ and $lcs_i[c]$ variables, respectively. The color-shared cells contain all cells on overlapping paths, and $lcs_i[c]$ corresponds to the colors of each disjoint set of color-shared cells. For instance, cells 7 and 12 make up one set of color-shared cells for three colors, and cell 19 makes up another one for two other colors.

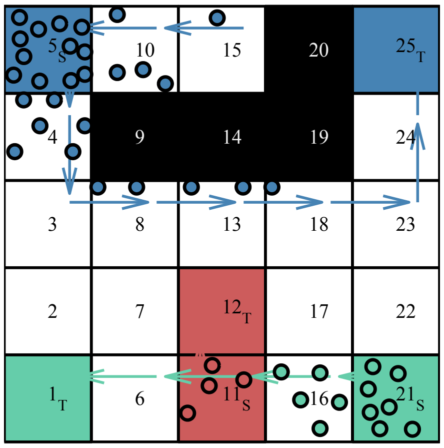

Figure 6: Example illustrating the two fairness requirements (Assumption 4) for proving liveness. Cells 9, 14, 19, and 20 failed, causing the original source-target path for blue to change from cells 5, 10, 15, 20, 25. If source cell 5 does not place new entities fairly, then entities on cells 10 and 15 may never reach the target. A similar situation occurs with paths of multiple colors in the lower part of the image, where source cell 11 may deadlock entities from source cell 21.

composing the transition systems of the individual cells. We first present the discrete transition system corresponding to each cell, and then describe the composition.

The variables associated with each Cell_i are as follows, with initial values of the variables shown in Figure 7 using the ':=' notation.

(a) $Entities_i$ is the set of identifiers for entities located on cell i. Cell i is said to be *nonempty* if $Entities_i \neq \emptyset$, and *empty* otherwise.

(b) $color_i$ designates the entity colors on the cell, or $\perp$ if there are none[2].

(c) $failed_i$ indicates whether or not i has failed.

(d) $NEPrev_i$ are the nonempty neighbors attempting to move entities (of any color) toward cell i.

(e) $token_i$ is a token used for fairness to indicate which neighbor may move toward i.

(f) $signal_i$ is the identifier of a neighbor of Cell_i that has permission to move toward Cell_i.

[2]It will be established that cells contain entities of only a single color, see Invariant 3.

```
private variables
    $token : ID_\perp := \perp$
    $failed : \mathbb{B} := false$
    $NEPrev$: $\texttt{Set}[ID_\perp] := \{\}$
    $lock : [C \to \mathbb{B}]$, init $\forall c \in C,\ lock := false$
    $pint : [C \to \texttt{Set}[ID_\perp]]$, init $\forall c \in C,\ pint[c] := \{\}$
shared variables
    $Entities : \texttt{Set}[P] := \{\}$
    $color : C_\perp := \perp$
    $signal : ID_\perp := \perp$
    $dist : [C \to \mathbb{N}_\infty]$, init $\forall c \in C,\ dist[c] := \infty$
    $next : [C \to ID_\perp]$, init $\forall c \in C,\ next[c] := \perp$
    $path : [C \to \texttt{Set}[ID_\perp]]$, init $\forall c \in C,\ path[c] := \{\}$
    $lcs : [C \to \texttt{Set}[C]]$, init $\forall c \in C,\ lcs[c] := \{\}$

transitions
  fail($i$)
    eff $failed := true$
      for each $c \in C$
        $dist[c] := \infty;\ next[c] := \perp$

  update
    eff $Route;\ Lock;\ Signal;\ Move$
```

Figure 7: Specification of Cell_i listing its variables, initial conditions, and transitions. Subscripts are dropped for readability.

Additionally, the following variables are defined as arrays for each color $c \in C$. The notation $next_i[c]$ means the c^{th} entry of the $next$ variable of cell i, and so on for the other variables.

(a) $next_i[c]$ is the neighbor towards which i attempts to move entities of color c.

(b) $dist_i[c]$ is the estimated distance—the number of cells—to the nearest target cell consuming entities of color c.

(c) $lock_i[c]$ is a Boolean variable for a lock of color c that some cells require to be able to move entities.

(d) $path_i[c]$ is the set of cell identifiers from any source of color c (and any nonempty cell with entities of color c) to the target of color c. This variable and the next two are local variables for cell i, but they are storing some global information.

(e) $pint_i[c]$ is the set of cell identifiers in traffic intersections with cells of color c (where $path_i[c]$ and $path_i[d]$ have nonempty intersection for some $d \neq c$).

(f) $lcs_i[c]$ is the set of colors that are involved in traffic intersections with the color c path.

When clear from context, the subscripts in the variable names are dropped. A state of Cell_i refers to a valuation of all these variables, i.e., a function that maps each variable to a value of the corresponding type. The complete system is an automaton, called System, consisting of the composition of all the cells. A state of System is a valuation of all the variables for all the cells. We refer to states of System with bold letters $\mathbf{x}$, $\mathbf{x}'$, etc.

Variables $token_i$, $failed_i$, $lock_i$, $NEPrev_i$, and $pint_i$ are private to Cell_i, while $Entities_i$, $dist_i$, $next_i$, $path_i$, $color_i$, $signal_i$, and lcs_i can be read by neighboring cells of Cell_i. This has the following interpretation for an actual message-passing implementation. At the beginning of each round, Cell_i broadcasts messages containing the values of these variables and receives similar values from its neighbors. Then, the computation of this round updates the local variables for each cell based on the values collected from its neighbors.

Variable $Entities_i$ is a special variable because it can also be written to by the neighbors of i. This is how we model transfer of entities between cells. For a state $\mathbf{x}$, for some $a \in A$ such that $\mathbf{x} \overset{a}{\rightarrow} \mathbf{x}'$, for some $i \in ID$, for some $j \in Nbrs_i$, for some entity $p \in \mathbf{x}.Entities_i$, then entity p transfers from cell i to j when $p \in \mathbf{x}'.Entities_j$. We use the notation p' to denote the state of entity p at $\mathbf{x}'$ where $\mathbf{x} \overset{a}{\rightarrow} \mathbf{x}'$ for some $a \in A$.

3.3. *Actions and the Composed System*

System is a discrete transition system modeling the composition of all the cells, and has two types actions: fails and updates. A fail(i) transition models the crash failure of the i^{th} cell and sets $failed_i$ to $true$, $dist_i[c]$ to ∞ for each $c \in C$, and $next_i[c]$ to $\perp$ for each $c \in C$. Cell i is called *faulty* if $failed_i$ is $true$, otherwise it is called *non-faulty*. The set of identifiers of all faulty and non-faulty cells at a state $\mathbf{x}$ is denoted by $F(\mathbf{x})$ and $NF(\mathbf{x})$, respectively. A faulty cell does nothing—it never moves and it never communicates[3].

An update transition models the evolution of all non-faulty cells over one synchronous round. For readability, we describe the state-change caused by an update transition as a sequence of four functions (subroutines), where for each non-faulty i,

(a) $Route$ computes the variables $dist_i$ and $next_i$,

(b) $Lock$ computes the variables $path_i$, $pint_i$, lcs_i, and $lock_i$,

(c) $Signal$ computes (primarily) the variable $signal_i$, and

(d) $Move$ computes the new positions of entities.

We note that in the single-color case considered in [31], the $Lock$ subroutine is unnecessary. The entire update transition is atomic, so there is no possibility to interleave fail transitions between the subroutines of update. Thus, the state of System at (the beginning of) round $k+1$ is obtained by applying these four subroutines to the state at round k. Next, we describe the distributed traffic control protocol that is implemented through these subroutines.

3.3.1. *Route Subroutine*

For each cell and each color, the $Route$ subroutine (Figure 8) constructs a distance-based routing table to the target cell of that color. This relies only on neighbors' estimates of distance to the target. Recall that failed cells have $dist[c]$ set to ∞ for every color $c \in C$. From a state $\mathbf{x}$, for each $i \in NF(\mathbf{x})$, the

[3]That $dist_i = \infty$ can be interpreted as i's neighbors not receiving a timely response from i.

```
if $\neg failed_i$ then
  $color_i := \{c \in C \mid \exists p \in Entities_i \wedge color(p) = c\}$
  if $i \notin ID_T$ then
    for each $c \in C$
      $dist_i[c] := \left(\min_{j \in Nbrs_i} dist_j[c]\right) + 1$
      if $dist_i[c] = \infty$ then $next_i[c] := \perp$
      else $next_i[c] := \operatorname{argmin}_{j \in Nbrs_i} \langle dist_j[c], j \rangle$
```

Figure 8: *Route* subroutine for Cell$_i$. This subroutine computes a minimum distance vector routing spanning tree rooted composed of non-faulty cells for each color, rooted at each target.

variable $dist_i[c]$ is updated as 1 plus the minimum value of $dist_j[c]$ for each neighbor j of i. If this results in $dist_i[c]$ being infinity, then $next_i[c]$ is set to $\perp$, but otherwise it is set to be the identifier with the minimum $dist[c]$ where ties are broken with neighbor identifiers.

Next, we introduce some definitions used to relate the system state to the variables used in the protocol. For a state $\mathbf{x}$, we inductively define the *color c target distance* ρ_c of a cell $i \in ID$ as the smallest number of non-faulty cells between i and tid_c:

$$\rho_c(\mathbf{x}, i) \triangleq \begin{cases} \infty & \text{if } \mathbf{x}.failed_i, \\ 0 & \text{if } i = tid_c \wedge \neg \mathbf{x}.failed_i, \\ 1 + \min_{j \in \mathbf{x}.Nbrs_i} \rho_c(\mathbf{x}, j) & \text{otherwise.} \end{cases}$$

A cell is said to be *target-connected to color c* if ρ_c is finite. We define the set of cells that are target-connected to tid_c as

$$TC(\mathbf{x}, c) \triangleq \{i \in NF(\mathbf{x}) \mid \rho_c(\mathbf{x}, i) < \infty\}.$$

For a state $\mathbf{x}$ and a color $c \in C$, we define the *routing graph* as $G_R(\mathbf{x}, c) \triangleq \langle V_R(\mathbf{x}, c), E_R(\mathbf{x}, c)\rangle$, where the vertices and directed edges are, respectively,

$$\begin{aligned} &V_R(\mathbf{x}, c) \triangleq NF(\mathbf{x}) \text{ and} \\ &E_R(\mathbf{x}, c) \triangleq \{(i, j) \in V_R(\mathbf{x}, c) \mid \rho_c(\mathbf{x}, j) = \rho_c(\mathbf{x}, i) + 1\}. \end{aligned}$$

Under this definition, $G_R(\mathbf{x}, c)$ is a spanning tree rooted at tid_c. We will show that the graph induced by the $next_i[c]$ variables self-stabilizes to the routing graph $G_R(\mathbf{x}, c)$ at some state $\mathbf{x}$ (Corollary 7). We previously introduced Δ as the worst-case diameter of the communication graph, and will refer to $\Delta(\mathbf{x})$ as the actual diameter at some state $\mathbf{x}$. We note that $\Delta(\mathbf{x}) \leq \Delta$.

3.3.2. *Lock Subroutine*

The *Lock* subroutine (Figure 9) executes after *Route*, and schedules traffic over intersections (the cells where source-to-target paths of different colors

```
if $\neg failed_i$
  for each $c \in C$
    if $i = sid_c \vee color_i = c \vee i \in path_i[c]$ then
      $path_i[c] := path_i[c] \cup \{i\} \cup \{next_i[c]\}$
    for each $j \in Nbrs_i$ // gossip the entity graph
      $path_i[c] := path_i[c] \cup path_j[c]$
    $pint_i[c] := \{j \in path_i[c] \cap path_i[d] \mid \exists c \neq d \in C\}$ // compute the set of color-shared cells
    if $pint_i[c] \neq \emptyset$
      $lcs_i[c] := \{d \in C \mid c \neq d \wedge path_i[c] \cap path_i[d] \neq \emptyset\}$
    // routing graphs stabilized and i needs a lock for color c
    if $detectRoutesStabilized() \wedge i \in pint_i[c] \wedge \neg lock_i[c]$
      Initiate mutual exclusion algorithm between all color-shared cells in $pint_i[c]$ using $lcs_i$ as input.
      Eventually, a color $d$ is returned. On return, if $d = c$ then $lock_i[c] := true$
    // detect if color-shared cells are empty
    if $detectRoutesStabilized() \wedge i \in pint_i[c] \wedge lock_i[c]$
      Initiate distributed snapshot algorithm to decide if all color-shared cells are empty after previously
      being nonempty with entities of color c. On return, if all cells are empty then
        $lock_i[c] := false$
```

Figure 9: *Lock* subroutine for Cell_i. This subroutine computes the color-shared cells—the cells in intersections—for each color, and then ensures liveness by giving a lock to only one color on each intersection.

overlap). To avoid deadlock scenarios, *Lock* maintains an invariant that entities of at most one color are on these intersections.

Moving entities over intersections requires some global coordination as illustrated by the following analogy. Consider the policy used to coordinate entities going in opposite directions over a one-lane bridge (see Figure 2), where there is a traffic signal on each side of the bridge. The protocol chooses one traffic light, allowing some entities to safely travel over the bridge in one direction. After some time, the protocol switches the lights (first turning green to red, and after the bridge is empty, turning red to green) allowing traffic to flow in the opposite direction. Then this process repeats.

Two parts of the previous example require global coordination and are included in the *Lock* subroutine. The first is how to choose the direction in which entities are allowed to travel—this is accomplished through the use of a mutual exclusion algorithm. The second is when to allow entities to travel the other direction—this is accomplished by determining when the intersection is empty. We now describe this global coordination more formally.

For defining the locking algorithm, we first define intersections. For this, we introduce the notion of an entity graph. Cell i is said to be in the *entity graph* of some color c at state $\mathbf{x}$ if one of the following conditions hold: (a) i is sid_c, (b) in state $\mathbf{x}$, i has entities of color c, or (c) in state $\mathbf{x}$, i is the neighbor closest—as defined by the number of cells—to tid_c of a cell already in the entity graph. We define the *color c entity graph* at state $\mathbf{x}$ as $G_E(\mathbf{x}, c) \triangleq \langle V_E(\mathbf{x}, c), E_E(\mathbf{x}, c)\rangle$, which is the following subgraph of the color c routing graph $G_R(\mathbf{x}, c)$. The vertices of $G_E(\mathbf{x}, c)$ are inductively defined as, $V_E(\mathbf{x}, c) \triangleq$

$$\{i \in NF(\mathbf{x}) \mid i = sid_c \vee \mathbf{x}.color_i = c \vee (\exists j \in V_E(\mathbf{x}, c).(i, j) \in E_R(\mathbf{x}, c))\}.$$

The edges of $G_E(\mathbf{x}, c)$ are $E_E(\mathbf{x}, c) \triangleq \{(i, j) \in V_E(\mathbf{x}, c) \times V_E(\mathbf{x}, c) \mid (i, j) \in E_R(\mathbf{x}, c)\}$. For example, if all cells are empty and non-faulty, then $V_E(\mathbf{x}, c)$ is the sequence of cell identifiers defined by following the minimum distance (as defined by ρ_c) from the source to the target of color c. That is, each $G_E(\mathbf{x}, c)$ is a simple path graph from source to target[4].

Now we describe how the entity graph of each color c is computed by each cell i as the $path_i[c]$ variable. If i is on the entity graph of color c, then we add i and i's $next$ variable for color c to the entity graph (see Figure 9, lines 3 and 4). Once the $next_i[c]$ variables stabilize (Corollary 7) and after an additional order of diameter rounds, the variable $path_i[c]$ contains all the nodes of the entity graphs since we gossip these graphs (line 6). That is, the graph formed by the $path_i[c]$ variables self-stabilizes to equal $G_E(\mathbf{x}, c)$, and contains the sequence of identifiers from any source or nonempty cell of color c to the target of color c (Corollary 8).

Next, the variable $pint_i[c]$ is computed to be the set of cell identifiers on the color c entity graph that overlaps with any other colored entity graph (line 7). The cells involved in such non-empty intersections represent physical traffic intersections, and are called *color-shared cells*. These cells require coordinated locking for traffic flow to progress. Cell i is in $pint_i[c]$ if and only if it will need a lock for color c.

For a color c, the c color-shared cells are the identifiers of those cells intersecting the entity graph of some other color. First, we define the c *shared colors* as the colors involved in intersections between entity graphs. For a state $\mathbf{x}$ and for any $c \in C$:

$$SC(\mathbf{x}, c) \triangleq \{d \in C \mid c \neq d \Rightarrow V_E(\mathbf{x}, c) \cap V_E(\mathbf{x}, d) \neq \emptyset\}.$$

Note that we always have $c \in SC(\mathbf{x}, c)$. Formally, we define the c *color-shared cells*, for a state $\mathbf{x}$ and for each $c \in C$, as

$$CSC(\mathbf{x}, c) \triangleq \{V_E(\mathbf{x}, d) \mid d \in SC(\mathbf{x}, c) \wedge d \neq c\}.$$

Note that, if there is no intersection between color c and some other color d, then there are no color-shared cells ($\emptyset$). In Figure 1, $CSC(\mathbf{x}, red) = CSC(\mathbf{x}, blue)$ $= \{8, 13\}$, that is, the red and blue color-shared cells are cells 8 and 13. Consider now Figure 5 with 6 colors at some state $\mathbf{x}$. Then $SC(\mathbf{x}, c)$ is $\{blue, red, green\}$ for c equal to blue, red, or green, $SC(\mathbf{x}, c)$ is $\{yellow, purple\}$ for c equal to yellow or purple, and $SC(\mathbf{x}, c)$ is $\{brown\}$ for c equal to brown. Here, $CSC(\mathbf{x}, red)$ $= CSC(\mathbf{x}, blue) = CSC(\mathbf{x}, green) = \{7, 12\}$, $CSC(\mathbf{x}, purple) = CSC(\mathbf{x}, yellow)$ $= \{19\}$, and $CSC(\mathbf{x}, brown) = \emptyset$. The $pint_i[c]$ variables stabilize to be $CSC(\mathbf{x}, c)$, at some state $\mathbf{x}$, for any color c (see Corollary 9).

Next, we determine the colors that will need to coordinate to schedule traffic through the color-shared cells. For these colors, a mutual exclusion algo-

[4]Once cells have failed, this may self-stabilize to be a tree from any cell with entities of color c to the target of color c.

```
if $\neg failed_i \wedge detectRoutesStabilized()$ then
  $cn := \{d \in C \mid \exists j \in Nbrs_i$ s.t. $next_j[d] = i$
                $\wedge color_j = d\}$

  if $color_i = \bot$ then $c :=$ choose from $cn$
  else $c := color_i$

  $NEPrev_i :=$
    $\{j \in Nbrs_i \mid next_j[c] = i \wedge Entities_j \neq \emptyset\}$

  if $token_i = \bot$ then
    $token_i :=$ choose from $NEPrev_i$
  let $j = token_i$
  if $\forall p \in Entities_i \mid \overline{p} \notin SR(i, j)$
    $\wedge (color_i \neq \bot \Rightarrow color_i = color_j)$
    $\wedge (j \in pint_i[c] \Rightarrow lock_j[c])$
  then
    $signal_i := j$
    if $|NEPrev_i| > 1$ then
      $token_i :=$ choose from $NEPrev_i \setminus \{j\}$
    elseif $|NEPrev_i| = 1$ then
      $token_i :=$ choose from $NEPrev_i$
    else $token_i := \bot$
  else $signal_i := \bot$; $token_i := j$
```

Figure 10: $Signal$ subroutine for Cell$_i$. Cell i signals fairly to some neighbor cell j if it is safe for j to move its entities toward i.

rithm is initiated between all cells for each disjoint set of cell colors in $pint_i[c]$. The $lcs_i[c]$ variables stabilize at some state $\mathbf{x}$ to equal $SC(\mathbf{x}, c)$, for any color c.

In general, up to $|C|$ colors could be involved in intersections, as well as all the smaller combinations. For instance, consider again Figure 5 with 6 colors at some state $\mathbf{x}$. Here, the blue and red entity graphs overlap, green and blue entity graphs overlap, but red and green do not, and independently, the purple and yellow entity graphs overlap (that is, not with blue, red, nor green), but no colors overlap with brown. Two mutual exclusion algorithms would be initiated, one with blue, red, and green as the input set of values, and another with yellow and purple as the input set. No mutual exclusion algorithm is initiated for brown since $SC(\mathbf{x}, brown) = \{brown\}$. Upon these two mutual exclusion instances terminating, one element of the first set, say $green$, would be chosen and given a lock, and one element, say $yellow$, of the second set would also be given a lock. The entities of these colors progress over the color-shared cells toward their intended targets. Finally, once the color-shared cells are empty again, $green$ and $yellow$ would each be removed from the respective input sets for fairness, and another mutual exclusion algorithm is initiated.

3.3.3. *Signal Subroutine*

The $Signal$ subroutine (Figure 10) executes after $Lock$. It is the key part of the protocol for maintaining safe entity separations, guaranteeing each cell has entities of only a single color, and ensuring progress of entities to the target.

The $Signal$ subroutine will only be executed for non-faulty cells after routes have stabilized, which may take rounds on the order of the diameter of the communication graph (Δ). One approach would be to wait until this stabilization occurs in terms of rounds (e.g., and only execute this subroutine if $round > 2\Delta$, or this many rounds after the last failure), but we instead suppose there exists a crash-tolerant predicate detection algorithm to detect if routes have stabilized [34, 35, 36]. We note that routes stabilizing is a stable predicate (which will be formally established in Lemma 6 and Corollary 8), and we abstract this predicate detection as a routine $detectRoutesStabilized()$. Specifically, $detectRoutesStabilized()$ returns true if and only if routes have stabilized (defined in Lemma 6), otherwise it returns false. This predicate detection is not necessary for safety, only for ensuring progress.

Roughly, each cell implements Signal through the next policies: (a) only accept entities from a neighbor when it is safe to do so, (b) only accept entities with the same color as the entities currently on the cell (or an arbitrary color if the cell is empty), (c) if a lock is needed, then only let entities move if it is acquired, and (d) ensure fairness by providing opportunities infinitely often for each nonempty neighbor to make progress.

First i computes a temporary variable cn, which is the set of colors for any neighbor that has entities of some color, with the corresponding next variable set to cell i. Next, cell i picks a color c from this set if it is empty, or the color of its own entities if it is nonempty, and will attempt to allow some cell with this chosen color to move toward itself. Then, cell i sets NEPrev_i to be the subset of Nbrs_i for which next has been set to i *and* $\mathit{Entities}$ is nonempty. If token_i is $\perp$, then it is set to some arbitrary value in NEPrev_i, but it continues to be $\perp$ if NEPrev_i is empty. Otherwise, $\mathit{token}_i = j$ for some neighbor j of i with nonempty $\mathit{Entities}_j$. This is accomplished through the conditional in line 6 as a step in guaranteeing fairness.

Next, it checks if there is any entity p with center $\overline{p}$ in the safety region of Cell_i on the side corresponding to token_i. If there is such an entity, then signal_i is set to $\perp$, which blocks the neighboring cell with identifier token_i from moving its entities in the direction of i, thus preventing entity transfers and ensuring safety. Otherwise, if there is no entity with its center in the safety region on side token_i, then signal_i is set to token_i to allow token_i to move its entities toward i. Subsequently, token_i is updated to a value in NEPrev_i that is different from its previous value, if that is possible according to the rules just described (lines 20–22).

3.3.4. *Move Subroutine*

Finally, the Move subroutine (Figure 11) models the physical movement of all the entities on cell i over a given round. For cell i, let j be $\mathit{next}_i[c]$, where c is color_i (which may be $\perp$ if cell i has no entities). Every entity in $\mathit{Entities}_i$ moves in the direction of j if and only if signal_j is set to i. The direction followed from cell i to j is $u(i,j)$, which is any vector satisfying Assumption 1. For example, for a square (or rectangular) cell i, one choice for $u(i,j)$ is the unit vector orthogonal to $\mathit{Side}(i,j)$ and pointing into j. In the case of an equilateral triangular cell i, one choice for $u(i,j)$ is also any orthogonal vector pointing into j.

The movement toward cell j may lead to some entities crossing the boundary of Cell_i into Cell_j, in which case, they are removed from $\mathit{Entities}_i$. If j is not the target matching the transferred entities' color, then the removed entities are added to $\mathit{Entities}_j$. In this case (line 9), any transferred entity p is placed so that $B(p,l)$ touches a single point of (is tangent to) $\mathit{Side}(i,j)$, the shared side of cells i and j, and lies on the inner side of the transfer region of cell j on side $\mathit{Side}(i,j)$. Resetting entity positions is a conservative overapproximation to the actual physical movement of entities, as this moves entities a distance greater

```
let $c = color_i$
let $j = next_i[c]$
if $\neg failed_i \wedge signal_j = i$ then
  for each $p \in Entities_i$
    $p := \overline{p} + vu(i, j)$
    if $\overline{p} \in TR_i$ then
      $Entities_i := Entities_i \setminus \{p\}$
      if $j \neq tid_c$ then
        $Entities_j := Entities_j \cup \{p\}$
        $\overline{p} := (\overline{p} + vu(i, j)) \cap Side(i, j)$ // point on shared side along movement vector
        $\overline{p} := (\overline{p} + n(i, j)) \cap ITR_j(Side(i, j))$ // inner transfer region along orthogonal line
```

Figure 11: *Move* subroutine for Cell$_i$. If i has received a signal to move from j, it updates the positions of all entities on it to move in j's direction, which may lead to some entities transferring from cell i to j.

than or equal to the distance they would travel if allowed to reside on cell boundaries. If j is the target matching the transferred entities' color, then the removed entities are not added to any cell and thus no longer exist in System.

Each source cells $i \in ID_S$, in addition to the above, adds a finite number of entities in each round to $Entities_i$, such that (a) the addition of these entities does not violate the minimum gap between entities at Cell$_i$, and (b) fairness is not violated. In the remainder of the paper, we will analyze System to show that in spite of failures, it maintains safety and liveness properties to be introduced in the next section.

4. Safety and Liveness of Distributed Traffic Control

In this section, we present an analysis of the safety and liveness properties of the distributed traffic control protocol modeled as the transition system System. We first summarize the results informally.

4.1. Summary of Results

Informally, the safety property requires that there is a minimum gap between entities on any cell, and the liveness property requires that all entities that reside on cells with feasible paths to the corresponding target eventually reach that target. In Section 4.2, we establish several invariant properties culminating in proving safety (Theorem 1): entities may never collide, even in spite of failures. Next in Section 4.3, we prove that the routing algorithm used to construct paths to the destinations is self-stabilizing in spite of arbitrary crash failures (Lemma 6). Then in Section 4.4, we show (Lemma 11)—under an assumption that failures do not introduce deadlock scenarios (Assumption 5)—that the locking algorithm allows multi-color flows to mutual-exclusively take control the color-shared cells (intersections). Finally in Section 4.5, under a fairness assumption, we establish the main progress property (Theorem 2) through two results, that any cell gets permission to move infinitely often, and that any cell with a permission to move decreases the distance of any entities on it from its destination.

4.2. *Safety and Collision Avoidance*

A state is safe if, for every cell, the boundaries of all entities in the cell are separated by a distance of r_s. For any state $\mathbf{x}$ of System, we define:

$$Safe_i(\mathbf{x}) \triangleq \forall p, q \in \mathbf{x}.Entities_i . p \neq q \Rightarrow ||\overline{p} - \overline{q}|| \geq 2l + r_s, \text{ and}$$
$$Safe(\mathbf{x}) \triangleq \forall i \in ID, Safe_i(\mathbf{x}).$$

This definition allows entities in different cells to be closer than $2l + r_s$ apart, but their centers will be spaced by at least $2l$. We proceed by proving some preliminary properties of System that will be used for proving *Safe* is an invariant.

The first property asserts that entities' cannot come close enough to the sides of cells to reside on multiple cells. This is because any entity whose boundary touches the side of a cell is transferred to the neighboring cell on that side (if one exists), and then the entity's position is reset to be completely within the new cell. Assumption 2 restricts the allowed partitions to ensure entity transfers are well-defined. For instance, some of the cells in the snub square tiling in Figure 3 do not satisfy Assumption 2. Consider an entity transfer from cell 3 to cell 5. There is no constant vector connecting the transfer regions of cell 3 to those of cell 5. This is because the side length of the transfer region of the triangular cell 5 is shorter than the side length of the transfer region of the square cell 3. However, in a transfer from cell 1 to cell 2 or vice-versa, the side lengths are the same. We also note that the assumption is only necessary for entity transfers from a cell with a longer transfer side length to a neighboring cell with smaller corresponding transfer side length. For example, a transfer from cell 5 to cell 3 is feasible.

Under Assumption 2, we have the following invariant, which states that the l-disc around each entity in a cell is completely contained within the cell.

Invariant 1. *In any reachable state* $\mathbf{x}$, $\forall i \in ID$, $\forall p \in \mathbf{x}.Entities_i$, $B(p, l) \setminus P_i = \emptyset$.

The next invariant states that cells' *Entities* sets are disjoint. This is immediate from the *Move* function since entities are only added to one cell's *Entities* upon being removed from a different cell's *Entities*.

Invariant 2. *In any reachable state* $\mathbf{x}$, *for any* $i, j \in ID$, *if* $i \neq j$, *then* $\mathbf{x}.Entities_i \cap \mathbf{x}.Entities_j = \emptyset$.

The following invariant states that cells contain entities of a single color in spite of failures. This follows from the *Signal* routine in Figure 10, where line 16 requires that if some neighbor j is attempting to move entities toward cell i, then the color of i is either $\perp$ or equal to the color of j.

Invariant 3. *In any reachable state* $\mathbf{x}$, *for all* $i \in ID$, *for all* $p, q \in \mathbf{x}.Entities_i$, $color(p) = color(q)$.

Next, we define a predicate that states that if $signal_i$ is set to the identifier of some neighbor $j \in Nbrs_i$, then there is a large enough area from the common side between i and j where no entities reside in Cell_i. Recall that $Side(i, j)$ is the line segment shared between neighboring cells i and j. For a state $\mathbf{x}$, $H(\mathbf{x}) \triangleq \forall i \in ID$, $\forall j \in Nbrs_i$, if $\mathbf{x}.signal_i = j$, then the following holds:

$$\forall p \in \mathbf{x}.Entities_i, \min_{x \in Side(i,j)} ||\overline{p} - x|| \geq 3d.$$

$H(\mathbf{x})$ is not an invariant property because once entities move the property may be violated. However, for proving safety, all that needs to be established is that at the *point of computation of the $signal$ variable* this property holds. The next key lemma states this.

Lemma 4. *For all reachable states $\mathbf{x}$, $H(\mathbf{x}) \Rightarrow H(\mathbf{x}_S)$ where $\mathbf{x}_S$ is the state obtained by applying the $Route$, $Lock$, and $Signal$ functions to $\mathbf{x}$.*

Proof. Fix a reachable state $\mathbf{x}$, an $i \in ID$, and a $j \in Nbrs_i$ such that $\mathbf{x}.signal_i = j$. Let $\mathbf{x}_R$ be the state obtained by applying the $Route$ function to $\mathbf{x}$, $\mathbf{x}_L$ be the state obtained by applying the $Lock$ function to $\mathbf{x}_R$, and $\mathbf{x}_S$ be the state obtained by applying the $Signal$ function to $\mathbf{x}_L$.

First, observe that both $H(\mathbf{x}_R)$ and $H(\mathbf{x}_L)$ hold. This is because the $Route$ and $Lock$ functions do not change any of the variables involved in the definition of $H(\cdot)$. Next, we show that $H(\mathbf{x}_L)$ implies $H(\mathbf{x}_S)$. If $\mathbf{x}_S.signal_i \neq j$ then the statement holds vacuously. Otherwise, $\mathbf{x}_S.signal_i = j$, then since (a) $H(\mathbf{x}_L)$ holds, and (b) Figure 11, line 6 is satisfied, we have that $H(\mathbf{x}_S)$. □

The following lemma asserts that if there is a cycle of length two formed by the $signal$ variables—which could occur due to failures—then entity transfers cannot occur between the involved cells in that round.

Lemma 5. *Let $\mathbf{x}$ be any reachable state and $\mathbf{x}'$ be a state that is reached from $\mathbf{x}$ after a single* `update` *transition (round). If $\mathbf{x}.signal_i = j$ and $\mathbf{x}.signal_j = i$, then $\mathbf{x}.Entities_i = \mathbf{x}'.Entities_i$ and $\mathbf{x}.Entities_j = \mathbf{x}'.Entities_j$.*

Proof. No entities enter either $\mathbf{x}'.Entities_i$ or $\mathbf{x}'.Entities_j$ from any other $m \in Nbrs_i$ or $n \in Nbrs_j$ since $\mathbf{x}.signal_i = j$ and $\mathbf{x}.signal_j = i$. It remains to be established that $\nexists p \in \mathbf{x}.Entities_j$ such that $p' \in \mathbf{x}'.Entities_i$ where $p = p'$ or vice-versa. Suppose such a transfer occurs. For the transfer to have occurred, $\overline{p}$ must be such that $\overline{p}' = (p_x, p_y) + vu(i, j)$ by Figure 11, line 5. But for $\mathbf{x}.signal_i = j$ to be satisfied, it must have been the case that $B(p, l) \cap P_i = \emptyset$ by Figure 11, line 6 and since $v < l$, a contradiction is reached. □

Using the previous results, we now prove that System preserves safety even when some cells fail.

Theorem 1. *In any reachable state $\mathbf{x}$ of* System, *$Safe(\mathbf{x})$.*

Proof. The proof is standard by induction over the length of any execution of System. The base case is satisfied by the assumption that initial states $\mathbf{x} \in Q_0$ satisfy $\mathit{Safe}(\mathbf{x})$. For the inductive step, consider any reachable states $\mathbf{x}$, $\mathbf{x}'$ and an action $a \in A$ such that $\mathbf{x} \xrightarrow{a} \mathbf{x}'$. Fix $i \in \mathit{ID}$ and assuming $\mathit{Safe}_i(\mathbf{x})$, we show that $\mathit{Safe}_i(\mathbf{x}')$. If $a = \mathsf{fail}_i$, then $\mathit{Safe}_i(\mathbf{x}')$ since no entities move.

For $a = \mathsf{update}$, then there are two cases to consider by Invariant 2. First, $\mathbf{x}'.\mathit{Entities}_i \subseteq \mathbf{x}.\mathit{Entities}_i$, that is, no new entities were added to i, but some may have transferred off i. There are two sub-cases. For the first sub-case, if $\mathbf{x}'.\mathit{Entities}_i = \mathbf{x}.\mathit{Entities}_i$, then all entities in $\mathbf{x}.\mathit{Entities}$ move identically with the same velocity, so the spacing between two distinct entities p, $q \in \mathbf{x}'.\mathit{Entities}_i$ is unchanged. Let $j = \mathit{next}_i[c]$ where $c = \mathit{color}_i$ by Invariant 3. Then, $\forall p, q \in \mathbf{x}.\mathit{Entities}_i$, $\forall p', q' \in \mathbf{x}'.\mathit{Entities}_i$ such that $p' = p$ and $q' = q$ and where $p \neq q$, we have (by Figure 11, line 5):

$$\left|\left|(p'_x, p'_y) - (q'_x, q'_y)\right|\right| = \left|\left|((p_x, p_y) + vu(i,j)) - ((q_x, q_y) + vu(i,j))\right|\right|.$$

It follows by the inductive hypothesis that $\left|\left|(p'_x, p'_y) - (q'_x, q'_y)\right|\right| \geq d$. The second sub-case arises if $\mathbf{x}'.\mathit{Entities}_i \subsetneq \mathbf{x}.\mathit{Entities}_i$, then $\mathit{Safe}_i(\mathbf{x}')$ is either vacuously satisfied or it is satisfied by the same argument just stated.

The second case is when $\mathbf{x}'.\mathit{Entities}_i \nsubseteq \mathbf{x}.\mathit{Entities}_i$, that is, there was at least one entity transfered to i. Consider any such transferred entity $p' \in \mathbf{x}'.\mathit{Entities}_i$ where $p' \notin \mathbf{x}.\mathit{Entities}_i$. There are two sub-cases. The first sub-case is when p' was added to $\mathbf{x}'.\mathit{Entities}_i$ because i is a source, that is, $i \in \mathit{ID}_S$. In this case, the specification of the source cells states that the entity p' was added to $\mathbf{x}'.\mathit{Entities}_i$ without violating $\mathit{Safe}_i(\mathbf{x}')$, and the proof is complete. Otherwise, p' was added to $\mathbf{x}'.\mathit{Entities}_i$ by some neighbor $j \in \mathbf{x}.\mathit{Nbrs}_i$, so $p' \in \mathbf{x}.\mathit{Entities}_j$ but $p' \notin \mathbf{x}.\mathit{Entities}_i$, and $p' \in \mathbf{x}'.\mathit{Entities}_i$ but $p' \notin \mathbf{x}'.\mathit{Entities}_j$. From lines 9–11 of Figure 11, we have the new position (p'_x, p'_y). The fact that p' transferred from Cell_j in $\mathbf{x}$ to Cell_i in $\mathbf{x}'$ implies that $\mathbf{x}.\mathit{next}_j = i$ and $\mathbf{x}.\mathit{signal}_i = j$—these are necessary conditions for the transfer by Figure 10, line 15. Thus, applying the predicate $H(\mathbf{x})$ at state $\mathbf{x}$ and by Lemma 4, it follows that for every $q \in \mathbf{x}.\mathit{Entities}_i$, $(q_x, q_y) \notin \mathit{SR}_i(s)(\mathit{Side}(i,j))$. It must now be established that if p' is transferred to $\mathbf{x}'.\mathit{Entities}_i$, then every $q' \in \mathbf{x}'.\mathit{Entities}_i$, where $q' \neq p'$ satisfies $(q'_x, q'_y) \notin \mathit{SR}_i(s)(\mathit{Side}(i,j))$, which means that any entity q already on cell i did not move toward the transferred entity p that is now on cell i. This follows by application of Lemma 5, which states that if entities on adjacent cells move towards one another simultaneously, then a transfer of entities cannot occur. This implies that the discs of all entities q' in $\mathbf{x}'.\mathit{Entities}_i$ are farther than r_s of the borders of any transferred entity p', implying $\mathit{Safe}_i(\mathbf{x}')$. Finally, since i was chosen arbitrarily, $\mathit{Safe}(\mathbf{x}')$. $\square$

Theorem 1 shows that System is safe in spite of failures.

4.3. Stabilization of Spanning Routing Trees

Next, we show under some additional assumptions, that once new failures cease to occur, System recovers to a state where each non-faulty cell with a feasible path to its target computes a route toward it. This route stabilization is then used in showing that any entity on a non-faulty cell with a feasible path to its target makes progress toward it. Our analysis relies on the following assumptions on cell failures and the placement of new entities on source cells. The first assumption states that no target cells fail, and is reasonable and necessary because if any target cell did fail, entities of that color obviously cannot make progress.

Assumption 3. *No target cells $t \in ID_T$ may fail.*

The next assumption ensures source cells place entities fairly so that they may not perpetually prevent any neighboring cell or any color-shared cell from making progress. The assumption is needed because it provides a specification of how the source cells behave, which has not been done so far. The assumption is reasonable because it essentially says that traffic is not produced perpetually without any break.

Assumption 4. *(Fairness): Source cells place new entities without perpetually blocking either: (i) any of their nonempty non-faulty neighbors, nor (ii) any cell $i \in CSC(\mathbf{x}, c)$, where c is source s's color.*

Formally, the first fairness condition states, for any execution α of System, for any color $c \in C$, for any source cell sid_c, if there exists an $i \in Nbrs_s$, such that for every state $\mathbf{x}$ in α after a certain round, $i \in \mathbf{x}.NEPrev_s$, then eventually $signal_s$ becomes equal to i in some round of α. The second fairness condition states, for any execution α of System, for any state $\mathbf{x} \in \alpha$, for any color $c \in C$, for any source cell sid_c, if there exists an $i \in NF(\mathbf{x})$ such that $i \in CSC(\mathbf{x}, c)$, and for every state $\mathbf{x}$ in α after a certain round, if cell i is nonempty, then eventually $signal_j$ becomes equal to i in some round of α, where j is a neighbor of i. Such conditions can be ensured if we suppose some oracle placing entities on source cells follows the same round-robin like scheme defined in the $Signal$ subroutine in Figure 10. Scenarios where each of these cases can arise are illustrated in Figure 6.

A fault-free execution fragment α be a sequence of states starting from $\mathbf{x}$ and along which no fail(i) transitions occur. That is, a fault-free execution fragment is an execution fragment with no new failure actions, although there may be existing failures at the first state $\mathbf{x}$ of α, so $F(\mathbf{x})$ need not be empty. Throughout the remainder of this section, we will consider fault-free executions that satisfy Assumptions 3 and 4.

Lemma 6. *Consider any reachable state $\mathbf{x}$ of* System*, any color $c \in C$, and any $i \in TC(\mathbf{x}, c) \setminus \{tid_c\}$. Let $h = \rho_c(\mathbf{x}, i)$. Any fault-free execution fragment α starting from $\mathbf{x}$ self-stabilizes within h rounds to a set of states S with all elements satisfying:*

$$\begin{aligned} dist_i[c] &= h, \text{ and} \\ next_i[c] &= i_n, \text{ where } \rho_c(\mathbf{x}, i_n) = h - 1. \end{aligned}$$

Proof. Fix an arbitrary state $\mathbf{x}$, a fault-free execution fragment α starting from $\mathbf{x}$, a color $c \in C$, and $i \in TC(\mathbf{x}, c) \setminus \{tid_c\}$. We have to show that (a) the set of states S is closed under update transitions and (b) after h rounds, the execution fragment α enters S.

First, by induction on h we show that S is stable. Consider any state $\mathbf{y} \in S$ and a state $\mathbf{y}'$ that is obtained by applying an update transition to $\mathbf{y}$. We have to show that $\mathbf{y}' \in S$. For the base case, $h = 1$, so $\mathbf{y}.dist_i[c] = 1$ and $\mathbf{y}.next_i[c] = tid_c$. From lines 5 and 7 of the $Route$ function in Figure 8, and that there is a unique tid_c for each color c, it follows that $\mathbf{y}'.dist_i[c]$ remains 1 and $\mathbf{y}'.next_i[c]$ remains tid_c. For the inductive step, the inductive hypothesis is, for any given h, if for any $j \in NF(\mathbf{x})$, $\mathbf{y}.dist_j[c] = h$ and $\mathbf{y}.next_j[c] = m$, for some $m \in ID$ with $\rho_c(\mathbf{x}, m) = h - 1$, then

$$\mathbf{y}'.dist_j[c] = h \text{ and } \mathbf{y}'.next_j[c] = m.$$

Now consider i such that $\rho_c(\mathbf{y}, i) = \rho_c(\mathbf{y}', i) = h + 1$. In order to show that S is closed, we have to assume that $\mathbf{y}.dist_i[c] = h+1$ and $\mathbf{y}.next_i[c] = m$, and show that the same holds for $\mathbf{y}'$. Since $\rho_c(\mathbf{y}', i) = h + 1$, i does not have a neighbor with target distance smaller than h. The required result follows from applying the inductive hypothesis to m and from lines 5 and 7 of Figure 8.

Second, we have to show that starting from $\mathbf{x}$, α enters S within h rounds. Once again, this is established by induction on h, which is $\rho_c(\mathbf{x}, i)$. Consider any state $\mathbf{y}$ such that $\rho_c(\mathbf{x}, i) = \rho_c(\mathbf{y}, i)$. The base case only includes the target distances satisfying $h = \rho_c(\mathbf{y}, i) = 1$ and follows by instantiating $i_n = tid_c$. For the inductive case, assume for the inductive hypothesis that at some state $\mathbf{y}$, $\mathbf{y}.dist_j[c] = h$ and $\mathbf{y}.next_j[c] = i_n$ such that $\rho_c(\mathbf{y}, i_n) = h - 1$, where i_n is the minimum identifier among all such cells (since we used cell identifiers to break ties). Observe that there is one such $j \in \mathbf{y}.Nbrs_i$ by the definition of TC. Then at state $\mathbf{y}'$, by the inductive hypothesis and lines 5 and 7 of Figure 8, $\mathbf{y}'.dist_i[c] = \mathbf{y}'.dist_j[c] + 1 = h + 1$. □

The next corollary of Lemma 6 states that, after new failures cease occurring, for all target-connected cells, the graph induced by the $next[c]$ variables self-stabilizes to the color c routing graph, $G_R(\mathbf{x}, c)$, within at most the diameter of the communication graph number of rounds, which is bounded by $\Delta(\mathbf{x})$. This implies that the predicate detection algorithm $detectRoutesStabilized()$ that detects that routes have stabilized also terminates.

Corollary 7. *Consider any execution α of* System *with an arbitrary but finite sequence of* fail *transitions. For any state $\mathbf{x} \in \alpha$ at least $2\Delta(\mathbf{x})$ rounds after the last* fail *transition, for any $c \in C$, every cell i target-connected to color c has $\mathbf{x}.next_i[c]$ equal to the identifier of the next cell along such a route.*

The next corollary of Lemma 6 and Corollary 7 states that within $2\Delta(\mathbf{x})$ rounds after routes self-stabilize, for each color $c \in C$, the identifiers in the $path_i[c]$ variables equal the vertices of the color c entity graph $G_E(\mathbf{x}, c)$. The result follows since routes self-stabilize and that $Lock$ is a function of $next$ and $path$ variables only, and that $path_i$ variables are gossiped in Figure 9, line 6.

Corollary 8. *Consider any execution α of* System *with an arbitrary but finite sequence of* fail *transitions. For any state $\mathbf{x} \in \alpha$ at least $2\Delta(\mathbf{x})$ rounds after the last* fail *transition, for every $c \in C$, every cell i target-connected to color c has $path_i[c] = V_E(\mathbf{x}, c)$.*

The next corollary of Lemma 6 states that eventually the values of the $pint[c]$ variables equal the set of color-shared cells $CSC(\mathbf{x}, c)$ for any cell i and color c. This is important because the mutual exclusion algorithm is initiated between the cells in $pint[c]$ (Figure 9, line 11).

Corollary 9. *Consider any execution α of* System *with an arbitrary but finite sequence of* fail *transitions. For any state $\mathbf{x} \in \alpha$ at least $2\Delta(\mathbf{x})$ rounds after the last* fail *transition, for every $c \in C$, every cell i target-connected to color c has $\mathbf{x}.pint[c] = CSC(\mathbf{x}, c)$.*

4.4. Scheduling Entities through Color-Shared Cells

In this section, we show that there is at most a single color on the set of color-shared cells if there are no failures. We then show that any cell that requests a lock eventually gets one, under an additional assumption that failures do not cause entities of more than one color to reside on the set of color-shared cells. Because failures cause the routing graphs and entity graphs to change, the color-shared cells that could previously be scheduled may now be deadlocked. Additionally, because we separately lock each disjoint set of color-shared cells to allow entities of some color to flow toward their target, it could be the case that the intermediate states between when the failure occurred and when routes have self-stabilized allowed entities to move in such a way that deadlocks the system. Such deadlocks could be avoided if a centralized coordinator informs every non-faulty cell to disable their signals when a failure is detected. The assumption states that with failures, the color-shared cells either all have the same-colored entities, or have no entities (and combinations thereof).

Assumption 5. Feasibility of Locking after Failures*: For any reachable state $\mathbf{x}$, for any color $c \in C$, consider the color-shared cells $CSC(\mathbf{x}, c)$. For all distinct cells $i, j \in CSC(\mathbf{x}, c)$ either $\mathbf{x}.color_i = \mathbf{x}.color_j$ or $\mathbf{x}.color_i = \perp$.*

The next lemma states that without failures, there are entities of at most a single color on the set of color-shared cells. The result is not an invariant because failures may cause the set of color-shared cells to change, resulting in deadlocks, which is why we need Assumption 5. By Invariant 3, we know that there are entities of at most a single color in each cell, so the following invariant is stated in terms of the color $color_i$ of each cell. We emphasize that Assumption 5 is unnecessary if there are no failures, as the algorithm ensures there are entities of at most a single color on the color-shared cells by the following lemma.

Lemma 10. *If there are no failures, for any reachable state $\mathbf{x}$, for any $c \in C$, for any $i \in CSC(\mathbf{x}, c)$, if $\neg\mathbf{x}.lock_i[c]$, then for all $j \in CSC(\mathbf{x}, c)$, we have $\mathbf{x}.color_j \neq c$.*

Proof. The proof is showing an inductive invariant, supposing no failures occur. For the initial state, all cells are empty, so we have $\mathbf{x}.color_i = \bot$ for any $i \in ID$. For the inductive step, we are only considering update actions by assumption. In the pre-state, we have $\neg\mathbf{x}.lock_i[c]$ and $\forall j \in CSC(\mathbf{x}, c)$, we have $\mathbf{x}.color_j \neq c$. Fix some $c \in C$ and some $i \in CSC(\mathbf{x}, c)$. For any subsequent state $\mathbf{x}'$, if $\mathbf{x}'.lock_i[c]$, the result follows vacuously. If $\neg\mathbf{x}'.lock_i[c]$, we must show $\forall j \in CSC(\mathbf{x}, c)$ that $\mathbf{x}.color_j \neq c$, so fix some $j \in CSC(\mathbf{x}, c)$. If $j \in CSC(\mathbf{x}', c)$, the result follows, since by the inductive hypothesis, $\mathbf{x}.color_j = \mathbf{x}'.color_j \neq c$. If $j \notin CSC(\mathbf{x}', c)$, the condition in *Signal* (Figure 10, line 17) cannot be satisfied since $\neg\mathbf{x}'.lock_i[c]$. Thus, no cell with entities of color c could move toward any cell in $CSC(\mathbf{x}', c)$, and we have $\mathbf{x}'.color_j \neq c$. □

The next lemma states that without failures, or with "nice" failures as described by Assumption 5, that any cell requesting a lock of some color will eventually get it, and thus it may move entities onto the color-shared cells.

Lemma 11. *For any reachable state $\mathbf{x}$ satisfying Assumption 5, for any $c \in C$, for any $i \in NF(\mathbf{x})$, if $i \in \mathbf{x}.pint[c]$ and all cells in $CSC(\mathbf{x}, c)$ are empty, then eventually a state $\mathbf{x}'$ is reached where $\mathbf{x}'.lock_i[c]$.*

Proof. By correctness of the mutual exclusion algorithm, eventually a color $d \in SC(\mathbf{x}', c)$ is returned and $\mathbf{x}'.lock_i[d] = true$ (Figure 9, line 11). If $c = d$, then the result follows. If $c \neq d$, by Lemma 10 and Assumption 5, we know that no other color aside from c has entities on any cell $j \in CSC(\mathbf{x}', c)$. The next time the mutual exclusion algorithm is initiated, d is excluded from the input set to the mutual exclusion algorithm (Figure 9, line 15), and by repeated argument, eventually $lock_i[c]$. □

4.5. Progress of Entities towards Targets

Using the results from the previous sections, we show that once new failures stop occurring, for every color $c \in C$, every entity of color c on a cell that is target-connected eventually gets to the target of color c. The result (Theorem 2) uses two lemmas which establish that, along every infinite execution with a finite number of failures, every nonempty target-connected cell gets permission to move infinitely often (Lemma 13), and a permission to move allows the entities on a cell to make progress towards the target (Lemma 12).

For the remainder of this section, we fix an arbitrary infinite execution α of System with a finite number of failures, satisfying Assumption 5. Let $\mathbf{x}_f$ be any state of System at least $2\Delta(\mathbf{x})$ rounds after the last failure, and α' be the infinite failure-free execution fragment $\mathbf{x}_f, \mathbf{x}_{f+1}, \ldots$ of α starting from $\mathbf{x}_f$. Note that in any such state $\mathbf{x}_f$, we have that $detectRoutesStabilized()$ (used in Figure 9, lines 11 and 15 and Figure 10, line 1) will return true since routes have stabilized. For any $c \in C$, observe that the number of target-connected cells remains constant starting from $\mathbf{x}_f$ for the remainder of the execution. That is, $TC(\mathbf{x}_f, c) = TC(\mathbf{x}_{f+1}, c) = TC(\ldots, c)$, so we fix $TC(c) = TC(\mathbf{x}_f, c)$.

Lemma 12. *For any $c \in C$, for any $i \in TC(c)$, for some $j \in \mathbf{x}_f.Nbrs_i$, if $k > f$, $\mathbf{x}_k.signal_j = i$, and $\mathbf{x}_k.next_i[c] = j$, for any entity $p \in \mathbf{x}_k.Entities_i$, let the distance function be defined by the lexicographically ordered tuple*

$$R(\mathbf{x}, p) = \langle \rho_c(\mathbf{x}, i), ds - \overline{p} \rangle ,$$

where ds is the point on the shared side $Side(i, j)$ defined by the line passing through $\overline{p}$ with direction $u(i, j)$. Then, $R(\mathbf{x}_{k+1}, p) < R(\mathbf{x}_k, p)$.

Proof. The first case is when no entity transfers from i to j in the $k + 1^{th}$ round: if $p' \in \mathbf{x}_{k+1}.Entities_i$ such that $p' = p$, then $||ds - \overline{p}'|| < ||ds - \overline{p}||$. In this case, the result follows since a velocity $v > 0$ is applied towards cell j by $Move$ in Figure 11, line 5. The second case is when some entity p transfers from i to j, so $p' \in \mathbf{x}_{k+1}.Entities_j$ such that $p' = p$. In this case, we have $\rho_c(\mathbf{x}_k, j) < \rho_c(\mathbf{x}_k, i)$, since the distance between j and tid_c is smaller than the distance between i and tid_c since routes have self-stabilized by Lemma 6. In either case, $R(\mathbf{x}_{k+1}, p) < R(\mathbf{x}_k, p)$, so entity p is closer to the appropriate target. □

The following lemma states that all cells with a path to the target receive a signal to move infinitely often, so Lemma 12 applies infinitely often.

Lemma 13. *For any $c \in C$, consider any $i \in TC(c) \setminus tid_c$, such that for all $k > f$, if $\mathbf{x}_k.Entities_i \neq \emptyset$, then $\exists k' > k$ such that $\mathbf{x}_{k'}.signal_{next_i[c]} = i$.*

Proof. Fix some $c \in C$. Since $i \in TC(c)$, there exists $h < \infty$ such that for all $k > f$, $\rho_c(\mathbf{x}_k, i) = h$. We prove the lemma by inducting on h. The base case is $h = 1$. Fix i and instantiate $k' = f + ns(tid_c)$. By Lemma 6, for any $t \in ID_T$, for all non-faulty $i \in Nbrs_t$, $\mathbf{x}_f.next_i[c] = t$ since $k > f$. For all $k > f$, if $\mathbf{x}_k.Entities_i \neq \emptyset$, then $signal_{tid_c}$ changes to a different neighbor with entities every round. It is thus the case that $|\mathbf{x}_k.NEPrev_{tid_c}| \leq ns(tid_c)$ and since $Entities_{tid_c} = \emptyset$ always, exactly one neighbor satisfies the conditional of Figure 10, line 6 in any round, then within $ns(tid_c)$ rounds, $signal_{tid_c} = i$.

For the inductive case, let $k_s = k + h$ be the step in α after which all non-faulty $a \in Nbrs_i$ have $\mathbf{x}_{k_s}.next_a[c] = i$ by Lemma 6. Also by Lemma 6, $\exists m \in Nbrs_i$ such that $\mathbf{x}_{k_s}.dist_m < \mathbf{x}_{k_s}.dist_i$, implying that after k_s, $|\mathbf{x}_{k_s}.NEPrev_i| \leq ns(i)$ since $\mathbf{x}_{k_s}.next_i = m$ and $\mathbf{x}_{k_s}.next_m \neq i$. By the inductive hypothesis, $\mathbf{x}_{k_s}.signal_{next_i[c]} = i$ infinitely often. If $i \in ID_S$, then entity initialization does not prevent $\mathbf{x}_k.signal_i = a$ from being satisfied infinitely often by the second assumption introduced in Section 4.3. It remains to be established that $signal_i = a$ infinitely often. Let $a \in \mathbf{x}_{k_s}.NEPrev_i$ where $\rho_c(\mathbf{x}_{k_s}, a) = h + 1$.

In any of the following cases, if $i \in \mathbf{x}_{k_s}.pint[c]$ and every cell $j \in CSC(\mathbf{x}_{k_s}, c)$ is empty, then Lemma 11 implies that eventually $lock_i[c]$. If $|\mathbf{x}_{k_s}.NEPrev_i| = 1$, then because the inductive hypothesis satisfies $signal_{next_i[c]} = i$ infinitely often, then Lemma 12 applies infinitely often, and thus $Entities_i = \emptyset$ infinitely often, finally implying that $signal_i = a$ infinitely often.

If $|\mathbf{x}_{k_s}.NEPrev_i| > 1$, there are two sub-cases. The first sub-case is when no entity enters i from some $d \neq a \in \mathbf{x}_{k_s}.NEPrev_i$, which follows by the same reasoning used in the $|\mathbf{x}_{k_s}.NEPrev_i| = 1$ case. The second sub-case is when a entity enters i from d, in which case it must be established that $signal_i = a$ infinitely often. This follows since if $\mathbf{x}_{k'}.token_i = a$ where $k' > k_t > k_s$ and k_t is the round at which an entity entered i from d, and the appropriate case of Lemma 4 is not satisfied, then $\mathbf{x}_{k'+1}.signal_i = \perp$ and $\mathbf{x}_{k'+1}.token_i = a$ by Figure 10, line 25. This implies that no more entities enter i from either cell d satisfying $d \neq a$. Thus $token_i = a$ infinitely often follows by the same reasoning $|\mathbf{x}_{k_s}.NEPrev_i| = 1$ case. □

The final theorem establishes that entities on any cell in $TC(c)$ eventually reach the target in α'.

Theorem 2. *For any $c \in C$, consider any $i \in TC(c)$, $\forall k > f$, $\forall p \in \mathbf{x}_k.Entities_i$, $\exists k' > k$ such that $p \in \mathbf{x}_{k'}.Entities_{next_i[c]}$.*

Proof. Fix $c \in C$, $i \in TC(c)$, a round $k > f$ and $p \in \mathbf{x}_k.Entities_i$. Let $h = \max_{i \in TC(c)} \rho_c(\mathbf{x}_f, i)$ which is finite. By Lemma 6, at every round after $k_s = k + h$ for any $i \in TC(c)$, the sequence of identifiers

$$\beta = i, \mathbf{x}_{k_s}.next_i[c], \mathbf{x}_{k_s}.next_{next_i[c]}[c], \ldots$$

forms a fixed path of cells to tid_c. Applying Lemma 13 to $i \in TC(c)$ shows that there exists $k_m \geq k_s$ such that $\mathbf{x}_{k_m}.signal_{next_i[c]} = i$. Now applying Lemma 12 to $\mathbf{x}_{k_m}$ shows p moves towards $\mathbf{x}_{k_s}.next_i[c]$, which is also $\mathbf{x}_{k_m}.next_i[c]$ since routes are stabilized. Lemma 13 further establishes that this occurs infinitely often, thus there exists a round $k' > k_m$ such that p gets transferred to the set $\mathbf{x}_{k_m}.Entities_{next_i[c]}$. □

By an induction on the sequence of identifiers in the path β, it follows that entities on any cell in $TC(c)$ eventually get consumed by the target.

5. Simulation Experiments

We have performed numerous simulation studies of the distributed traffic control protocol for validating the theoretical model and results developed in this paper, and for evaluating the protocol's throughput performance. In this section, we discuss the main findings with illustrative examples taken from the simulation results. We implemented the simulator in Matlab, and all the partition figures displayed in the paper are created using it[5]. The distributed traffic control protocol and simulator are entirely deterministic, except for (a) the choice of the initial positions of new entities in source cells, and (b) failure and recovery rates in scenarios that consider random failures and recoveries, as detailed below.

[5]The simulator source code repository is available online: `https://bitbucket.org/verivital/cell_flows/`. Videos of several scenarios such as the environments considered in this paper are available online `https://www.youtube.com/user/verivital/`, specifically in the playlist `https://www.youtube.com/playlist?list=PLZVw65tJnqISpctDeLMcIsascuXLvyXnY`.

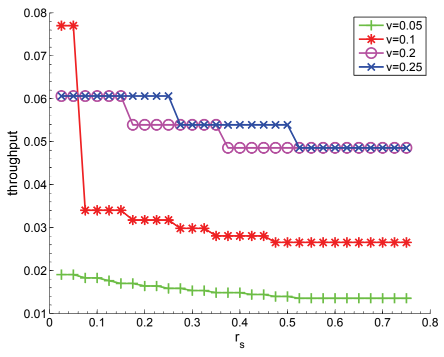

Figure 12: Throughput versus safety spacing r_s for several values of v, for $K = 2500$, $l = 0.25$ for System with an 8×8 unit square tessellation.

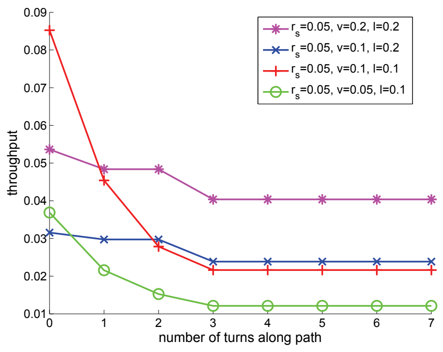

Figure 13: Throughput versus number of turns along a path, for a path of length 8, where $K = 2500$, $r_s = 0.05$, and each of l and v are varied for System with an 8×8 unit square tessellation.

Let the *K-round throughput* of System be the total number of entities arriving at the target over K rounds, divided by K. We define the *average throughput* (henceforth throughput) as the limit of K-round throughput for large K. All simulations start at a state where all cells are empty and subsequently entities are added to the source cells.

5.1. Single-color throughput without failures as a function of r_s, l, v

Rough calculations show that throughput should be proportional to cell velocity v, and inversely proportional to safety distance r_s and entity radius l. Figure 12 shows throughput versus r_s for several choices of v for an 8×8 unit square tessellation instance of System with a single entity color. The parameters are set to $l = 0.25$ and $K = 2500$. The entities move along a line path where the source is the bottom left corner cell and the target is the top left corner cell. For the most part, the inverse relationship with v holds as expected: all other factors remaining the same, a lower velocity makes each entity take longer to move away from the boundary, which causes the predecessor cell to be blocked more frequently, and thus fewer entities reach tid from any element of ID_S in the same number of rounds. In cases with low velocity (for example $v = 0.1$) and for very small r_s, however, the throughput can actually be greater than that at a slightly higher velocity. We conjecture that this somewhat surprising effect appears because at very small safety spacing, the potential for safety violation is higher with faster speeds, and therefore there are many more blocked cells per round. We also observe that the throughput saturates at a certain value of r_s (≈ 0.55). This situation arises when there is roughly only one entity in each cell.

5.2. Single-color throughput without failures as a function of the path

For a sufficiently large number of rounds K, throughput is independent of the length of the path. This of course varies based on the particular path and

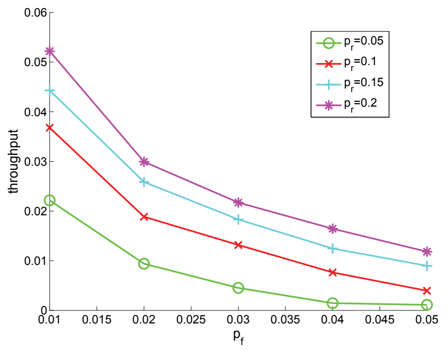

Figure 14: Throughput versus failure rate p_f for several recovery rates p_r with an initial path of length 8, where $K = 20000$, $r_s = 0.05$, $l = 0.2$, and $v = 0.2$ for System with an 8×8 unit square tessellation.

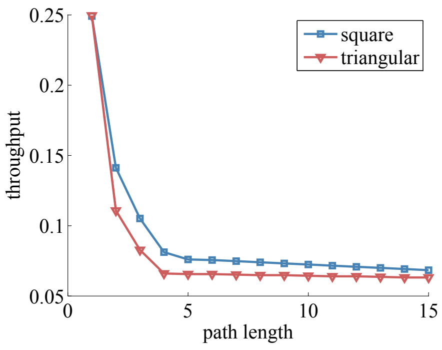

Figure 15: Throughput versus increasing path length of square (blue) and equilateral triangular (red) partitions.

instance of System considered, but all other variables fixed, this relationship is observed. More interesting however, is the relationship between throughput and path complexity, measured in the number of turns along a path. A turn represents a change in direction along a path. For example, in Figure 19 there are three turns, between cells 12 and 16, cells 15 and 14, and cells 6 and 2. In Figure 2, there are two turns for each the blue and red paths. Figure 13 shows throughput versus the number of turns along paths of length 8. This illustrates that throughput decreases as the number of turns increases, up to a point at which the decrease in throughput saturates. This saturation is due to signaling and indicates that there is only one entity per cell.

5.3. *Single-color throughput under failure and recovery of cells*

Finally, we considered a random failure and recovery model in which at each round each non-faulty cell fails with some probability p_f and each faulty cell recovers with some probability p_r [37]. A *recovery* sets $failed_i = false$ and in the case of tid also resets $dist_{tid} = 0$, so that eventually $Route$ will correct $next_j$ and $dist_j$ for any $j \in TC$. Intuitively, we expect that throughput will decrease as p_f increases and increase as p_r increases. Figure 14 demonstrates this result for $0.01 \leq p_f \leq 0.05$ and $0.05 \leq p_r \leq 0.2$. There is a diminishing return on increasing p_r for a fixed p_f, in that for a fixed p_f increasing p_r results in smaller throughput gains.

5.4. *Multi-color throughput as a function of the number of intersecting cells*

Now we discuss the influence of multi-color throughput. In the case where the paths between different sources and targets do not overlap, all the results from the single-color simulation results apply. In the case where the paths do overlap, the mutual exclusion algorithm runs to ensure no deadlocks occur. This additional control logic will have an influence on the throughput. For the

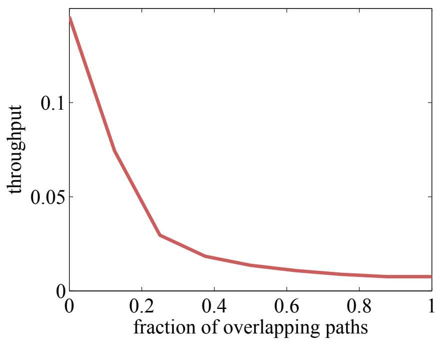

Figure 16: Throughput versus fraction of path overlap for two colors on a 1×16 unit square tessellation.

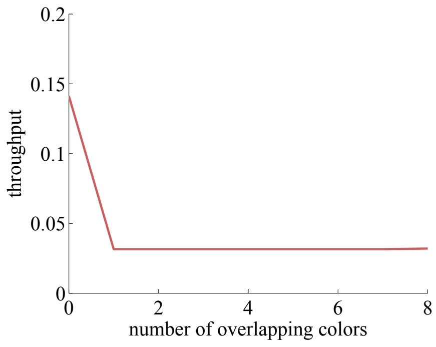

Figure 17: Throughput versus number of overlapping colors on a 1×3 unit square tessellation.

multi-color cases, we consider the summed throughput, which is the sum of the throughput for each color.

Figure 16 shows the roughly exponential decrease in throughput as the fraction of overlapping paths increases for two colors with path length 8 and no turns. The fraction of overlapping paths is defined as the number of vertices in the color-shared cells $CSC(\mathbf{x}, c)$. As the fraction increases, the paths lie completely on top of one another, so in this case with path length 8, we have no overlap, 1 cell overlap, etc.

5.5. *Multi-color throughput as a function of the number of intersecting colors*

Intersections (that is, scenarios that have at least one color-shared cell) have a fixed cost on throughput. Specifically, the summed throughput of there being two overlapping colors on a cell is the same as the summed throughput of three or more. Figure 17 shows this fixed decrease in summed throughput as the number of overlapping colors increases for a fixed path of length 3 with 3 color-shared cells, where the decrease in summed throughput from having no overlaps to having one color overlapping is about 4.5 times. Once there are two colors, all additional colors do not decrease the summed throughput. This observation agrees with intuition—the decrease in summed throughput due to an intersection is independent of the number of destinations for the entities that must pass through that intersection.

6. Related Work

There is a large amount of work on traffic control in transportation systems (see, e.g., [4, 38]) and robotics (see, e.g., [39]). We briefly summarize some of the more related work, but highlight that we are presenting a formal model of an example of such systems. Distributed air and automotive traffic control have been studied in many contexts. Human-factors issues are considered in [40, 41] to ensure collision avoidance between the coordination of numerous pilots and a supervisory controller modeling the semi-centralized air traffic control components. The Cell Transmission Model and variants thereof are used for modeling highway traffic flows and throughput, but do not allow for reasoning about system safety or liveness [42, 43]. The Small Aircraft Transportation Protocol (SATS) is semi-distributed air traffic control protocol designed for small airports without radar, so pilots and their aircraft coordinate among themselves to land after being assigned a landing sequence order by an automated system at the airport [16]. SATS has been formally modeled and analyzed using a combination of model checking and automated theorem proving [44]. SATS and this paper share an abstraction: the physical environment is a priori partitioned into a set of regions of interest, and properties about the whole system are proved using compositional analysis. Safe conflict resolution maneuvers for distributed air traffic control are designed in [45]. A formal model of the traffic collision avoidance system (TCAS) is developed and analyzed for safety in [46]. TCAS is a system deployed on aircraft that alerts pilots when other aircraft are in close proximity and guides them along safe trajectories.

A distributed algorithm (executed by entities, vehicles in this case) for controlling automotive intersections without any stop signs is presented in [18]. Some methods for ensuring liveness for automotive intersections are presented in [47]. A method to detect the mode of a hybrid system control model of an autonomous vehicle in intersections is developed in [48], and is used to reduce conservatism of the maximally controlled invariant set (the set of collision-free controls). Efficient distributed intersection control algorithms are developed in [49]. There is a large amount of work on flocking [50] and platooning [51, 52, 53, 54]. Only a few works consider failures in such systems, like the arbitrary failures considered in [55, 56], the actuator failures considered in [54], or in synchronization of swarm robot systems in [57].

Robot coordination based on discrete abstractions like [58, 23, 28, 27, 59, 60, 61, 62, 63, 64] can be viewed as traffic control. For instance, [23] establishes a formal connection between the continuous and the discrete parts of these protocols, and also presents a self-stabilizing algorithm with similar analysis to the analysis in this paper. These works also decompose the continuous problem into a discrete abstraction by partitioning the environment, but all these works allow at most a single entity (robot) in each partition, while our framework allows numerous entities in each partition. If several entities are to visit some destination in [59, 62, 63], like our targets here, that destination is represented as the union of a set of partitions and each entity must reside in one of these partitions. Based on the size of the entity, it may be possible that only a single entity can geometrically fit on a cell in our partition, and we call such scenarios *single-entity partitions*. We can thus include some of the results of [28, 27, 59, 63] within our framework, but mention that these works allow linear or nonlinear dynamics within a single partition, and also automatically synthesize the distributed behavior from a given temporal logic specification. We take the alternative approach and design the routing and control mechanism manually.

The Kiva Systems robotic warehouse [58] is a robotic traffic control system on square partitions, and can be described in our framework by allowing a single entity per cell. In these warehouse systems, there is a central coordinator scheduling tasks, but the robots are responsible for path planning using an A*-like search algorithm [58]. However, several deadlock scenarios are identified when performing such path planning [60]. The *Adaptive Highways Algorithm* presented in [60] for scheduling entities relies on using the tentative trajectories of other robots collected by the central controller. Deadlocks are also observed in other distributed robotics path-planning algorithms on discrete partitions in [65]. Deadlock scenarios can also arise without a discrete abstraction, such as in the doorways considered in [66], the path formation algorithms of [67], or the warehouse automation system of [68]. Lastly, we mention that most of these works on traffic control from aviation, automotive, swarm robotics, and warehouse automation applications can be modeled within the framework of spatial computing [69, 70, 71].

7. Discussion

In this section, we discuss some ways to generalize assumptions used in the paper and some alternative methods. In this paper, we presented a distributed traffic control algorithm for the partitioned plane, which moves entities without collision to their destinations, in spite of failures. While our algorithm is presented for two-dimensional partitions, an extension to some three-dimensional partitions (e.g., cubes and tetrahedra) follows in an obvious way. An extension to the more general case where there are multiple sources and multiple targets of each color—and entities of each color move toward the nearest target of that color—is straightforward, but complicates notation.

7.1. *Self-Stabilizing Mutual Exclusion and Distributed Snapshot Algorithms*

There are a variety of mutual exclusion algorithms that could be used to determine locks (Figure 9, line 11). For this paper, we require the overall system to be stabilizing and therefore the locking algorithm itself should be stabilizing. To this end, any of the following algorithms could be adapted to our framework: the token circulation algorithm [72], mutual exclusion [73], group mutual exclusion [74], snap-stabilizing propagation of information with feedback (PIF) algorithm [75], or k-out-of-l mutual exclusion [76]. A self-stabilizing distributed snapshot algorithm (see [30, Ch. 5]) can be used to determine if all c color-shared cells are empty, after having had some entity of color c (Figure 9, line 15). If all cells are empty, then another round of mutual exclusion commences, excluding color c from the input set.

7.2. *Other Failure Classes*

In this paper, we have considered only crash (fail-stop) failures and limited crash-recovery scenarios. Other failure models are possible to incorporate, such as failure of the actuators or sensors in the cells or entities. One interesting variant of failures are Byzantine failures, which would need to be defined within the context of this framework. Supposing a Byzantine failure represents arbitrary communications behavior of a faulty cell only (and not, e.g., moving entities arbitrarily to neighboring cells so that they violate safety by inducing collisions), then a Byzantine faulty cell could lie about its state to neighbors. Several situations could arise here, such as a failed cell lying that it is safe for the failed cell to receive entities when it is not, lying about its distance to the target cell of a given color, etc. Some form of voting scheme (as is common in Byzantine tolerant algorithms) could potentially be employed to avoid safety violations, but it is hard to imagine an algorithm that could avoid progress and liveness violations. We thus leave this as an interesting and nontrivial direction for future work.

7.3. General Triangulations and Affine Dynamics

We assume in Section 2 that the partitions satisfy several geometric assumptions for feasibility of entity transfers. We considered using vector fields generated by a discrete abstraction like those presented in [26, 77, 78, 79, 28, 27]. The affine vector fields generated on simplices in [26, 79] can be used to move an entity (with potentially nonholonomic or nonlinear dynamics) through any side of a cell in a triangulation (simplex) [26, 78] or rectangle [77]. However, it turns out that it is impossible to maintain our notion of safety for such vector fields without additional collision avoidance mechanisms implemented on each entity. This is due to a simple geometric observation—moving entities through a shorter side than the side they entered through may require the entities to come closer together. For example, if a cell in the triangulation has an obtuse angle, then the vector field generated by [26] flowing from the longest edge to the shortest edge has negative divergence. Furthermore, a vector field having negative divergence implies the flow corresponding to any two distinct points starting in that field come closer together, hence safety cannot be maintained. The distributed problems using these discrete abstractions [59, 62, 63] avoid this by requiring at most one entity in any (triangular) partition at a time.

We also mention a simple condition to ensure that triangulations have the required geometric partition properties (Assumptions 1 and 2). If all the triangles in the triangulation are non-obtuse, then the triangulation satisfies these assumptions. We also note that restricting allowable triangulations of an environment to ones without obtuse angles is not restrictive, since any polygon can be efficiently partitioned into a triangulation with non-obtuse [80, 81] or acute [82] angles.

7.4. Inter-Cell Entity Motion Coupling

In our framework, due to the presence of failures, it is nontrivial to generalize from identical motion of entities within a given cell to coupled motion. Without failures, we could generalize this notion to allow for upper and lower bounds of entity velocity magnitudes in each cell, in which case we could reduce the assumption of identical coupling of entity motion within cells described in Section 2. This represents the case where the entity velocities are all approximately equal and is more realistic. A sufficiently large transfer region could be chosen based on the maximum length of the cells and difference between minimum and maximum velocities to ensure a result that no two entities on the cell may move closer than a constant bound toward one another while remaining on that cell, and the safety invariant would hold.

Relaxing the identical movement assumption is nontrivial to accomplish when failures are considered. Without failures (and thus with stable routing graphs), we can handle this as described above using a sufficiently large transfer region. With failures, it is problematic, as we would need to bound the number of times the outgoing direction from a given cell may change, as otherwise one can imagine a scenario where the route toggles between different sides of a cell, causing the net velocity of the cells to be toward the interior of the cell, and with different velocity bounds, some entities may move too close together to satisfy safety. Alternatively, perhaps the failure tolerance mechanism could wait and prevent entity movement if a failure is detected (which is implied if a route changes after initial stabilization) until the routes have stabilized again. This approach may reduce throughput due to waiting, but we leave this general direction of relaxing the identical movement assumption to coupling as an interesting direction for future research

7.5. Insufficiency of Disjoint Routing Paths

Finding disjoint paths, such as by using the algorithms from [83, 84, 85, 86], could be another approach to solving the multi-color problem, but the locking mechanism used here solves a more general problem. Even without failures, there are many environments and choices of sources and targets for which there are no disjoint paths between sources and targets. One such environment is shown in Figure 4, where for two distinct colors c and d, the paths between the respective sources and targets necessarily overlap, so an algorithm for finding disjoint paths cannot be used as there are no disjoint paths between sources and targets. However, there are disjoint paths in some cases, so no scheduling would be necessary if these are found, but our routing algorithm does not necessarily find these, as the disjoint paths may not be shortest distance. A self-stabilizing algorithm for finding disjoint paths on planar graphs would be an enhancement to our algorithm, as it would increase throughput in the case that paths need not overlap.

7.6. Back-Pressure and Wormhole Routing

Back-pressure routing [87, 88] is an algorithm for dynamically routing traffic over an underlying graph using congestion gradients. If we view the color of each entity as its intended address and consider this problem from the perspective of queuing theory, one might think back-pressure routing could provide a throughput-optimal solution for the problem. However, our physical motion model is incompatible with back-pressure routing. For a given cell, our model does not allow arbitrary choice of the next neighbor for each entity on that cell. In particular, when one cell moves its entities toward a neighboring cell, all entities sufficiently near the shared side between the two neighbors would transfer.

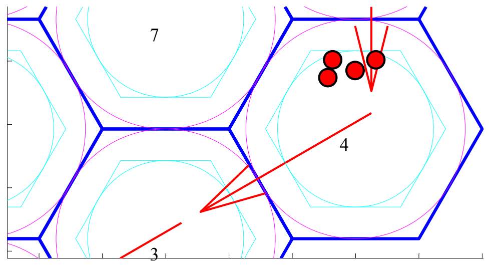

Figure 18: Hexagonal partition that does not satisfy the projection property (Assumption 1). An extension to allow such partitions would require enlarging the transfer region and receiving a signal from all of the potential next neighbors, which would require cells 3 and 7 both to signal cell 4 to move. This is because when cell 4 moves its entities toward cell 3, it may be possible for entities to transfer to either cells 3 or 7.

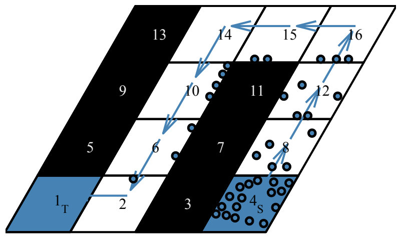

Figure 19: Example system on a parallelogram partition with failed cells in black. The several turns along the path from the source to the target cause a saturation of entities on cells 6, 10, and 14. The movement vector $u(i, j)$ is defined as the unit vector parallel to the x axis for movement between horizontal neighbors, and the unit vector parallel to the vertical sides of the parallelograms between vertical neighbors.

Wormhole routing [89] is a flow control policy over a fixed underlying graph for determining when packets move to the node on the graph. Addresses in wormhole routing are very short and come at the beginning of a packet, so a packet can be subdivided into pieces or *flits* and begin being forwarded after the address is received, yielding a snake-like sequence of flits in transfer. One could also view the sequence of entities on a path toward the appropriately-colored target (see Figure 19) sequence of flits flowing to a destination in wormhole routing. While similar deadlock scenarios can arise in our system and wormhole routing, wormhole routing is incompatible with our system due to the motion model just like back-pressure routing.

8. Conclusion

We present a self-stabilizing distributed traffic control protocol for the partitioned plane, where each cell (partition) controls the motion of all entities within that cell. The algorithm guarantees separation between entities in the face of crash failures of the software controlling a cell. Once new failures cease occurring, it guarantees progress of all entities that are neither isolated by (a) failed cells, nor (b) cells with entities of other colors that become deadlocked due to failures, to the respective targets. Through simulations, we presented estimates of throughput as a function of velocity, minimum separation, single-target path complexity, failure-recovery rates, and multi-target path complexity.

For practical applications, we need algorithms that tolerate a relaxed coupling between entities and allow entities some degree of independent movement while preserving safety and progress. The goal would be to design efficient control algorithms for this relaxed setting under different assumptions about the behavior of the free entities. Additionally, it would be interesting to develop strategies allowing entities of different colors on a single cell. Our strategy of preventing entities of different colors from residing on a single cell simplified some analysis, but it also complicated some analysis, particularly by making it harder to prove progress because deadlock scenarios may frequently arise. It would be interesting to develop algorithms allowing mixing and sorting of colors using different types of motion coupling and to reduce the restriction of identical entity motion within cells. It would also be interesting to design algorithms that can allow relaxing the assumption on what failures may occur to ensure progress. For the class of crash failures considered in this paper, the throughput of the algorithm could be improved by utilizing self-stabilizing disjoint path routing algorithms, when it is possible to find such disjoint paths, since we identified that it is not possible in general. This may require a more complex routing algorithm to temporarily move entities of some colors off the color shared cells, thus allowing some other color on the color shared cells to make progress. Other interesting avenues for extension would be to consider and tolerate more general forms of failures, such as Byzantine failures of the cells that may either cause incorrect or malicious communication or physical movements that violate safety or progress.

9. Acknowledgments

This work was supported in part by NSF CAREER grant number 1054247, AFOSR YIP grant number FA9550-12-1-0336, and the Air Force Research Laboratory's Information Directorate (AFRL/RI) through the Visiting Faculty Research Program (VFRP) under contract number FA8750-13-2-0115. The authors thank Zhongdong Zhu for helping develop the current version of the simulator, Karthik Manamcheri for helping develop an earlier version of the simulator, and Nitin Vaidya for feedback and pointers to wormhole routing. The authors thank Jeffrey Lale, Hershed Tilak, Jerry Sun, and Dongeek Shin for their efforts working with and extending the earlier version of the simulator. The authors are grateful for interesting discussions with Michael Loui, Daniel Liberzon, and PR Kumar on this work. The authors thank the anonymous reviewers whose feedback helped improve this paper and earlier revisions [90, 31].

References

[1] D. Helbing, M. Treiber, Jams, waves, and clusters, Science 282 (1998) 2001–2003.

[2] B. S. Kerner, Experimental features of self-organization in traffic flow, Phys. Rev. Lett. 81 (17) (1998) 3797–3800.

[3] C. Daganzo, M. Cassidy, R. Bertini, Possible explanations of phase transitions in highway traffic, Transportation Research A 33 (1999) 365–379.

[4] M. Nolan, Fundamentals of air traffic control, Wadsworth Publishing Company, 1994.

[5] F. Borgonovo, L. Campelli, M. Cesana, L. Coletti, Mac for ad hoc inter-vehicle network: services and performance, in: IEEE Vehicular Technology Conf., Vol. 5, 2003, pp. 2789–2793.

[6] M. Karpiriski, A. Senart, V. Cahill, Sensor networks for smart roads, in: Pervasive Computing and Communications Workshops, 2006. PerCom Workshops 2006. Fourth Annual IEEE International Conference on, 2006, pp. 1–5.

[7] S. S. Manvi, M. S. Kakkasageri, J. Pitt, Multiagent based information dissemination in vehicular ad hoc networks, Mob. Inf. Syst. 5 (4) (2009) 363–389.

[8] S. R. Azimi, G. Bhatia, R. R. Rajkumar, P. Mudalige, Vehicular networks for collision avoidance at intersections, SAE International Journal of Passenger Cars - Mechanical Systems 4 (1) (2011) 406–416.

[9] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, P. Mahoney, Stanley: The Robot That Won the DARPA Grand Challenge, in: M. Buehler, K. Iagnemma, S. Singh (Eds.), The 2005 DARPA Grand Challenge, Vol. 36 of Springer Tracts in Advanced Robotics, Springer Berlin / Heidelberg, 2007, pp. 1–43.

[10] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. M. Howard, S. Kolski, A. Kelly, M. Likhachev, M. McNaughton, N. Miller, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, B. Salesky, Y.-W. Seo, S. Singh, J. Snider, A. Stentz, W. R. Whittaker, Z. Wolkowicki, J. Ziglar, H. Bae, T. Brown, D. Demitrish, B. Litkouhi, J. Nickolaou, V. Sadekar, W. Zhang, J. Struble, M. Taylor, M. Darms, D. Ferguson, Autonomous driving in urban environments: Boss and the urban challenge, Journal of Field Robotics 25 (8) (2008) 425–466.

[11] X. Yang, L. Liu, N. Vaidya, F. Zhao, A vehicle-to-vehicle communication protocol for cooperative collision warning, in: Mobile and Ubiquitous Systems: Networking and Services. MOBIQUITOUS. The First Annual International Conference on, 2004, pp. 114–123.

[12] J. Misener, R. Sengupta, H. Krishnan, Cooperative collision warning: Enabling crash avoidance with wireless technology, in: 12th World Congress on Intelligent Transportation Systems, 2005, pp. 1–11.

[13] B. Hoh, M. Gruteser, R. Herring, J. Ban, D. Work, J.-C. Herrera, A. M. Bayen, M. Annavaram, Q. Jacobson, Virtual trip lines for distributed privacy-preserving traffic monitoring, in: MobiSys '08: Proceeding of the 6th International Conference on Mobile Systems, Applications, and Services, ACM, New York, NY, USA, 2008, pp. 15–28.

[14] A. Girard, J. de Sousa, J. Misener, J. Hedrick, A control architecture for integrated cooperative cruise control and collision warning systems, in: Decision and Control. Proceedings of the 40th IEEE Conference on, Vol. 2, 2001, pp. 1491–1496.

[15] M. Mamei, F. Zambonelli, L. Leonardi, Distributed motion coordination with co-fields: a case study in urban traffic management, in: Autonomous Decentralized Systems. ISADS. The Sixth International Symposium on, 2003, pp. 63–70.

[16] T. S. Abbott, K. M. Jones, M. C. Consiglio, D. M. Williams, C. A. Adams, Small aircraft transportation system, higher volume operations concept: Normal operations, Tech. Rep. NASA/TM-2004-213022, NASA (Aug. 2004).

[17] M. Kelly, G. Di Marzo Serugendo, A decentralised car traffic control system simulation using local message propagation optimised with a genetic algorithm, in: S. Brueckner, S. Hassas, M. Jelasity, D. Yamins (Eds.), Engineering Self-Organising Systems, Vol. 4335 of Lecture Notes in Computer Science, Springer, 2007, pp. 192–210.

[18] H. Kowshik, D. Caveney, P. R. Kumar, Safety and liveness in intelligent intersections, in: Hybrid Systems: Computation and Control (HSCC), 11th International Workshop, Vol. 4981 of LNCS, 2008, pp. 301–315.

[19] K. Dresner, P. Stone, A multiagent approach to autonomous intersection management, Journal of Artificial Intelligence Research 31 (2008) 591–656.

[20] P. Weiss, Stop-and-go science, Science News 156 (1) (1999) 8–10.

[21] Kornylak, Omniwheel brochure (2008).
URL `http://www.kornylak.com/images/pdf/omni-wheel.pdf`.

[22] K. An, A. Trewyn, A. Gokhale, S. Sastry, Model-driven performance analysis of reconfigurable conveyor systems used in material handling applications, in: Cyber-Physical Systems (ICCPS), 2011 IEEE/ACM International Conference on, Vol. 2, IEEE, 2011, pp. 141–150.

[23] S. Gilbert, N. Lynch, S. Mitra, T. Nolte, Self-stabilizing robot formations over unreliable networks, ACM Trans. Auton. Adapt. Syst. 4 (2009) 1–17.

[24] S. Dolev, L. Lahiani, S. Gilbert, N. Lynch, T. Nolte, Virtual stationary automata for mobile networks, in: PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing, ACM, New York, NY, USA, 2005, pp. 323–323.

[25] T. Nolte, N. Lynch, A virtual node-based tracking algorithm for mobile networks, in: Distributed Computing Systems, International Conference on (ICDCS), IEEE Computer Society, Los Alamitos, CA, USA, 2007, pp. 1–9.

[26] C. Belta, V. Isler, G. Pappas, Discrete abstractions for robot motion planning and control in polygonal environments, Robotics, IEEE Transactions on 21 (5) (2005) 864–874.

[27] G. E. Fainekos, A. Girard, H. Kress-Gazit, G. J. Pappas, Temporal logic motion planning for dynamic robots, Automatica 45 (2) (2009) 343 – 352.

[28] H. Kress-Gazit, G. Fainekos, G. Pappas, Temporal-logic-based reactive mission and motion planning, Robotics, IEEE Transactions on 25 (6) (2009) 1370–1381.

[29] A. Arora, M. Gouda, Closure and convergence: A foundation of fault-tolerant computing, IEEE Transactions on Software Engineering 19 (1993) 1015–1027.

[30] S. Dolev, Self-stabilization, MIT Press, Cambridge, MA, 2000.

[31] T. T. Johnson, S. Mitra, K. Manamcheri, Safe and stabilizing distributed cellular flows, in: Proceedings of the 30th IEEE International Conference on Distributed Computing Systems (ICDCS), IEEE, Genoa, Italy, 2010, pp. 577–586.

[32] E. W. Dijkstra, Self-stabilizing systems in spite of distributed control, Commun. ACM 17 (11) (1974) 643–644.

[33] M. Schneider, Self-stabilization, ACM Comput. Surv. 25 (1) (1993) 45–67.

[34] N. A. Lynch, Distributed Algorithms, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[35] V. Garg, J. Mitchell, Distributed predicate detection in a faulty environment, in: Distributed Computing Systems, 1998. Proceedings. 18th International Conference on, 1998, pp. 416–423.

[36] F. Gärtner, S. Pleisch, (im)possibilities of predicate detection in crash-affected systems, in: A. Datta, T. Herman (Eds.), Self-Stabilizing Systems, Vol. 2194 of LNCS, Springer Berlin / Heidelberg, 2001, pp. 98–113.

[37] R. E. L. DeVille, S. Mitra, Stability of distributed algorithms in the face of incessant faults, in: Proceedings of 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS), Springer, 2009, pp. 224–237.

[38] P. A. Ioannou, Automated Highway Systems, Plenum Press, New York, NY, USA, 1997.

[39] F. Bullo, J. Cortés, S. Martínez, Distributed Control of Robotic Networks, Applied Mathematics Series, Princeton University Press, 2009, to appear. Electronically available at http://coordinationbook.info.

[40] N. Leveson, M. de Villepin, J. Srinivasan, M. Daouk, N. Neogi, E. Bachelder, J. Bellingham, N. Pilon, G. Flynn, A safety and human-centered approach to developing new air traffic management tools, in: Proceedings Fourth USA/Europe Air Traffic Management R&D Seminar, 2001, pp. 1–14.

[41] T. Prevot, Exploring the many perspectives of distributed air traffic management: The multi aircraft control system (macs), in: Proceedings of the HCI-Aero, 2002, pp. 149–154.

[42] C. F. Daganzo, The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory, Transportation Research Part B: Methodological 28 (4) (1994) 269 – 287.

[43] C. F. Daganzo, The cell transmission model, part ii: Network traffic, Transportation Research Part B: Methodological 29 (2) (1995) 79 – 93.

[44] C. Muñoz, V. Carreño, G. Dowek, Formal analysis of the operational concept for the small aircraft transportation system, in: M. Butler, C. Jones, A. Romanovsky, E. Troubitsyna (Eds.), Rigorous Development of Complex Fault-Tolerant Systems, Vol. 4157 of LNCS, Springer, 2006, pp. 306–325.

[45] C. Tomlin, G. Pappas, S. Sastry, Conflict resolution for air traffic management: A study in multiagent hybrid systems, IEEE Transactions on Automatic Control 43 (4) (1998) 509–521.

[46] C. Livadas, J. Lygeros, N. A. Lynch, High-level modeling and analysis of TCAS, in: Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS '99), 1999, pp. 115–125.

[47] T.-C. Au, N. Shahidi, P. Stone, Enforcing liveness in autonomous traffic management, in: Proceedings of the Twenty-Fifth Conference on Artificial Intelligence, AAAI, 2011, pp. 1317–1322.

[48] R. Verma, D. Vecchio, Semiautonomous multivehicle safety, Robotics Automation Magazine, IEEE 18 (3) (2011) 44–54.

[49] A. Colombo, D. D. Vecchio, Efficient algorithms for collision avoidance at intersections, in: Hybrid Systems: Computation and Control (HSCC), ACM, 2012, pp. 145–154.

[50] R. Olfati-Saber, Flocking for multi-agent dynamic systems: algorithms and theory, IEEE Transactions on Automatic Control 51 (3) (2006) 401–420.

[51] P. Varaiya, Smart cars on smart roads: Problems of control, IEEE Transactions on Automatic Control 38 (1993) 195–207.

[52] E. Dolginova, N. Lynch, Safety verification for automated platoon maneuvers: A case study, in: HART '97 (International Workshop on Hybrid and Real-Time Systems), Vol. 1201 of LNCS, Springer Verlag, 1997, pp. 154–170.

[53] D. Swaroop, J. K. Hedrick, Constant spacing strategies for platooning in automated highway systems, Journal of Dynamic Systems, Measurement, and Control 121 (1999) 462–470.

[54] T. T. Johnson, S. Mitra, Safe flocking in spite of actuator faults using directional failure detectors, Journal of Nonlinear Systems and Applications 2 (1-2) (2011) 73–95.

[55] V. Gupta, C. Langbort, R. Murray, On the robustness of distributed algorithms, in: Decision and Control. 45th IEEE Conference on, 2006, pp. 3473–3478.

[56] M. Franceschelli, M. Egerstedt, A. Giua, Motion probes for fault detection and recovery in networked control systems, in: American Control Conference, 2008, 2008, pp. 4358–4363.

[57] A. Christensen, R. O'Grady, M. Dorigo, From fireflies to fault tolerant swarms of robots, IEEE Transactions on Evolutionary Computation 13 (4) (2009) 754–766.

[58] P. R. Wurman, R. D'Andrea, M. Mountz, Coordinating hundreds of cooperative, autonomous vehicles in warehouses, AI Magazine 29.

[59] M. Kloetzer, C. Belta, Automatic deployment of distributed teams of robots from temporal logic motion specifications, Robotics, IEEE Transactions on 26 (1) (2010) 48–61.

[60] H. Roozbehani, R. D'Andrea, Adaptive highways on a grid, in: C. Pradalier, R. Siegwart, G. Hirzinger (Eds.), Robotics Research, Vol. 70 of Springer Tracts in Advanced Robotics, Springer, 2011, pp. 661–680.

[61] J. W. Durham, R. Carli, P. Frasca, F. Bullo, Discrete partitioning and coverage control for gossiping robots, Robotics, IEEE Transactions on 28 (2) (2011) 364–378.

[62] X. C. Ding, M. Kloetzer, Y. Chen, C. Belta, Automatic deployment of robotic teams, Robotics Automation Magazine, IEEE 18 (3) (2011) 75–86.

[63] Y. Chen, X. C. Ding, A. Stefanescu, C. Belta, Formal approach to the deployment of distributed robotic teams, Robotics, IEEE Transactions on 28 (1) (2012) 158–171.

[64] Y. Chen, X. Ding, A. Stefanescu, C. Belta, A formal approach to deployment of robotic teams in an urban-like environment, in: A. Martinoli, F. Mondada, N. Correll, G. Mermoud, M. Egerstedt, M. A. Hsieh, L. E. Parker, K. Sty (Eds.), Distributed Autonomous Robotic Systems, Vol. 83 of Springer Tracts in Advanced Robotics, Springer Berlin Heidelberg, 2013, pp. 313–327.

[65] R. Luna, K. Bekris, Network-guided multi-robot path planning in discrete representations, in: Intelligent Robots and Systems (IROS). IEEE/RSJ International Conference on, 2010, pp. 4596–4602.

[66] D. Herrero-Perez, H. Matinez-Barbera, Decentralized coordination of autonomous agvs in flexible manufacturing systems, in: Intelligent Robots and Systems. IROS. IEEE/RSJ International Conference on, 2008, pp. 3674–3679.

[67] S. Nouyan, A. Campo, M. Dorigo, Path formation in a robot swarm: Self-organized strategies to find your way home, Swarm Intelligence 2 (1) (2008) 1–23.

[68] A. Kamagaew, J. Stenzel, A. Nettstrater, M. ten Hompel, Concept of cellular transport systems in facility logistics, in: Automation, Robotics and Applications (ICARA). 5th International Conference on, 2011, pp. 40–45.

[69] F. Zambonelli, M. Mamei, Spatial computing: An emerging paradigm for autonomic computing and communication, in: M. Smirnov (Ed.), Autonomic Communication, Vol. 3457 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2005, pp. 227–228.

[70] J. Beal, J. Bachrach, Infrastructure for engineered emergence on sensor/actuator networks, Intelligent Systems, IEEE 21 (2) (2006) 10–19.

[71] J. Bachrach, J. Beal, J. McLurkin, Composable continuous-space programs for robotic swarms, Neural Computing & Applications 19 (2010) 825–847.

[72] C. Johnen, G. Alari, J. Beauquier, A. Datta, Self-stabilizing depth-first token passing on rooted networks, in: M. Mavronicolas, P. Tsigas (Eds.), Distributed Algorithms, Vol. 1320 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 1997, pp. 260–274.

[73] A. K. Datta, C. Johnen, F. Petit, V. Villain, Self-stabilizing depth-first token circulation in arbitrary rooted networks, Distributed Computing 13 (2000) 207–218.

[74] J. Beauquier, S. Cantarell, A. Datta, F. Petit, Group mutual exclusion in tree networks, in: Parallel and Distributed Systems, 2002. Proceedings. Ninth International Conference on, IEEE Computer Society, 2002, pp. 111–116.

[75] A. Bui, A. Datta, F. Petit, V. Villain, Snap-stabilization and pif in tree networks, Distributed Computing 20 (2007) 3–19.

[76] A. Datta, S. Devismes, F. Horn, L. Larmore, Self-stabilizing k-out-of-l exclusion on tree networks, in: Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, 2009, pp. 1–8.

[77] C. Belta, L. Habets, Controlling a class of nonlinear systems on rectangles, Automatic Control, IEEE Transactions on 51 (11) (2006) 1749–1759.

[78] L. Habets, P. Collins, J. van Schuppen, Reachability and control synthesis for piecewise-affine hybrid systems on simplices, Automatic Control, IEEE Transactions on 51 (6) (2006) 938–948.

[79] M. Kloetzer, C. Belta, A fully automated framework for control of linear systems from temporal logic specifications, Automatic Control, IEEE Transactions on 53 (1) (2008) 287–297.

[80] B. Baker, E. Grosse, C. Rafferty, Nonobtuse triangulation of polygons, Discrete & Computational Geometry 3 (1988) 147–168.

[81] M. Bern, S. Michell, J. Ruppert, Linear-size nonobtuse triangulation of polygons, Discrete & Computational Geometry 14 (1995) 411–428.

[82] H. Maehara, Acute triangulations of polygons, European Journal of Combinatorics 23 (1) (2002) 45–55.

[83] H. Mohanty, G. P. Bhattacharjee, A distributed algorithm for edge-disjoint path problem, in: Proceedings of the Sixth Conference on Foundations of Software Technology and Theoretical Computer Science, Springer-Verlag, London, UK, UK, 1986, pp. 344–361.

[84] R. Ogier, V. Rutenburg, N. Shacham, Distributed algorithms for computing shortest pairs of disjoint paths, Information Theory, IEEE Transactions on 39 (2) (1993) 443 –455.

[85] S.-J. Lee, M. Gerla, Split multipath routing with maximally disjoint paths in ad hoc networks, in: Communications. ICC. IEEE International Conference on, Vol. 10, 2001, pp. 3201–3205.

[86] M. Marina, S. Das, On-demand multipath distance vector routing in ad hoc networks, in: Network Protocols. Ninth International Conference on, 2001, pp. 14–23.

[87] L. Tassiulas, A. Ephremides, Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks, Automatic Control, IEEE Transactions on 37 (12) (1992) 1936–1948.

[88] B. Awerbuch, T. Leighton, A simple local-control approximation algorithm for multicommodity flow, in: Foundations of Computer Science. Proceedings., 34th Annual Symposium on, IEEE, 1993, pp. 459–468.

[89] L. Ni, P. McKinley, A survey of wormhole routing techniques in direct networks, Computer 26 (2) (1993) 62–76.

[90] T. T. Johnson, Fault-tolerant distributed cyber-physical systems: Two case studies, Master's thesis, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 (May 2010).