# Routing Regardless of Network Stability

Bundit Laekhanukit[1], Adrian Vetta[1], and Gordon Wilfong[2]

[1] McGill University
[2] Bell Laboratories

**Abstract.** How effective are interdomain routing protocols, such as the *Border Gateway Protocol*, at routing packets? Theoretical analyses have attempted to answer this question by ignoring the packets and instead focusing upon protocol stability. To study stability, it suffices to model only the control plane (which determines the routing graph) – an approach taken in the *Stable Paths Problem*. To analyse packet routing, it requires modelling the interactions between the control plane and the forwarding plane (which determines when packets are forwarded), and our first contribution is to introduce such a model. We then examine the effectiveness of packet routing in this model for the broad class *next-hop preferences with filtering*. Here each node $v$ has a *filtering list* $\mathcal{D}(v)$ consisting of nodes it does not want its packets to route through. Acceptable paths (those that avoid nodes in the filtering list) are ranked according to the *next-hop*, that is, the neighbour of $v$ that the path begins with. On the negative side, we present a strong inapproximability result. For filtering lists of cardinality at most one, given a network in which an equilibrium is guaranteed to exist, it is NP-hard to approximate the maximum number of packets that can be routed to within a factor of $O(n^{1-\epsilon})$, for any constant $\epsilon > 0$. On the positive side, we give algorithms to show that in two fundamental cases *every* packet will eventually route with probability one. The first case is when each node's filtering list contains only itself, that is, $\mathcal{D}(v) = \{v\}$. Moreover, with positive probability every packet will be routed before the control plane reaches an equilibrium. The second case is when all the filtering lists are empty, that is, $\mathcal{D}(v) = \emptyset$. Thus, with probability one packets will route even when the nodes don't care if their packets cycle! Furthermore, with probability one every packet will route even when the control plane has *no* equilibrium at all. To our knowledge, these are the first results to guarantee the possibility that all packets get routed without stability. These positive results are tight – for the general case of filtering lists of cardinality one, it is not possible to ensure that every packet will eventually route.

## 1 Introduction

In the *Stable Paths Problem* (SPP) [1], we are given a directed graph $G = (V, A)$ and a sink (or destination) node $r$. Furthermore, each node $v$ has a ranked list of some of its paths to $r$ and the lowest ranked entry in the list is the "empty

path"[3]. This preference list is called $v$'s list of *acceptable paths*. A set of paths, one path $\mathcal{P}(v)$ from each node $v$'s list of acceptable paths, is termed *stable* if (i) they are *consistent*: if $u \in \mathcal{P}(v)$, then $\mathcal{P}(u)$ must be the subpath of $\mathcal{P}(v)$ beginning at $u$, and (ii) they form an *equilibrium*: for each node $v$, $\mathcal{P}(v)$ is the path ranked highest by $v$ of the form $vP(w)$ where $w$ is a neighbour of $v$. The stable paths problem asks whether a stable set of paths exists in the network. The SPP has risen to prominence as it is viewed as a *static* description of the problem that the Border Gateway Protocol (BGP) is trying *dynamically* to solve. BGP can be thought of as trying to find a set of stable routes to $r$ so that routers can use these routes to send packets to $r$.

Due to the importance of BGP, both practical and theoretical aspects of the SPP have been studied in great depth. To avoid overloading the reader with practical technicalities and for reasons of space, we defer discussions on a sample of this vast literature and on the technical aspects of BGP to the full paper. Two observations concerning the SPP, though, are pertinent here and motivate our work:
(1) Even if a stable solution exists, the routing tree induced by a consistent set of paths might not be spanning. Hence, a stable solution may **not** actually correspond to a functioning network – there may be isolated nodes that cannot route packets to the sink! Disconnectivities arise because nodes may prefer the empty-path to any of the paths proffered by its neighbours; for example, a node might not trust certain nodes to handle its packets securely or in a timely fashion, so it may reject routes traversing such unreliable domains. This problem of non-spanning routing trees has quite recently been studied in the context of a version of BGP called iBGP [4]. In Section 3, we show that non-connectivity is a very serious problem (at least, from the theoretical side) by presenting an $O(n^{1-\epsilon})$ hardness result for the combinatorial problem of finding a *maximum cardinality stable subtree*.
(2) The SPP says nothing about the dynamic behaviour of BGP. Stable routings are significant for many practical reasons (e.g., network operators want to know the routes their packets are taking), but while BGP is operating at the control plane level, packets are being sent at the forwarding plane level without waiting for stability (if, indeed, stability is ever achieved). Thus, it is important to study network performance in the dynamic case. For example, what happens to the packets whilst a network is unstable? This is the main focus of our paper: to investigate packet routing under network dynamics.

Towards this goal, we define a distributed protocol, inspired by BGP, that stops making changes to the routing graph (i.e., becomes stable) if it achieves a stable solution to the underlying instance of SPP. The current routing graph itself is determined by the control plane but the movement of packets is determined by the *forwarding plane*. Thus, our distributed protocol provides a framework under which the control and forwarding planes interact; essentially, this primarily

[3] Clearly, the empty path is not a real path to the sink; we call it a path for clarity of exposition.

means that we need to understand the relative speeds at which links change and packets move.

Given this model we analyse the resulting trajectory of packets. In a stable solution, a node in the stable tree containing the sink would have its packets route whereas an isolated node would not. For unstable networks, or for stable networks that have not converged, things are much more complicated. Here the routes selected by nodes are changing over time and, as we shall see, this may cause the packets to cycle. If packets can cycle, then keeping track of them is highly non-trivial. Our main results, however, are that for two fundamental classes of preference functions (i.e., two ways of defining acceptable paths and their rankings) all packets will route with probability one in our model. That is, there is an execution of the our distributed protocol such that *every* packet in the network will reach the destination (albeit, possibly, slowly) even in instances where the network has no stable solution. (Note that we are ignoring the fact that in BGP packets typically have a *time-to-live* attribute meaning that after traversing a fixed number of nodes the packet will be dropped.) Furthermore, when the network does have a stable solution, we are able to guarantee packet routing even before the time when the network converges.

These positive results on the routing rate are to our knowledge, the first results to guarantee the possibility of packet routing without stability. The results are also tight in the sense that, for any more expressive class of preference function, our hardness results show that guaranteeing that all packet eventually route is not possible – thus, packets must be lost.

## 2 The Model and Results

We represent a network by a directed graph $G = (V, A)$ on $n$ nodes. The destination node in the network is denoted by a distinguished node $r$ called a *sink* node. We assume that, for every node $v \in V$, there is a directed path in $G$ from $v$ to the sink $r$, and the sink $r$ has no outgoing arc. At any point in time $t$, each node $v$ chooses at most one of its out-neighbours $w$ as its *chosen next-hop*; thus, $v$ selects one arc $(v, w)$ or selects none. These arcs form a *routing graph* $\mathcal{R}_t$, each component of which is a 1-*arborescence*, an *in-arborescence*[4] $T$ plus possibly one arc $(v, w)$ emanating from the root $v$ of $T$, i.e, $T$ and $T \cup \{(v, w)\}$ are both 1-arborescences. (If the root of a component does select a neighbour, then that component contains a unique cycle.) When the context is clear, for clarity of exposition, we abuse the term *tree* to mean a 1-arborescence, and we use the term *forest* to mean a set of trees. A component (tree) in a routing graph is called a *sink-component* if it has the sink $r$ as a root; other components are called *non-sink components.*

Each node selects its outgoing arc according to its preference list of acceptable paths. We examine the case where these lists can be generated using two of the most common preference criteria in practice: *next-hop preferences* and *filtering.*

[4] An *in-arborescence* is a graph $T$ such that the underlying undirected graph is a tree and every node has a unique path to a root node.

For next-hop preferences, each node $v \in V$ has a ranking on its *out-neighbours*, nodes $w$ such that $(v, w) \in A$. We say that $w$ is the *$k$-th choice* of $v$ if $w$ is an out-neighbour of $v$ with the $k$-th rank. For $k = 1, 2, \ldots, n$, we define a set of arcs $A_k$ to be such that $(v, w) \in A_k$ if $w$ is the $k$-th choice of $v$, i.e., $A_k$ is the set of *the $k$-th choice arcs*. Thus, $A_1, A_2, \ldots, A_n$ partition a set of arcs $A$, i.e., $A = A_1 \cup A_2 \cup \ldots A_n$. We call the entire graph $G = (V, A)$ an *all-choice* graph. A *filtering list*, $\mathcal{D}(v)$, is a set of nodes that $v$ never wants its packets to route through. We allow nodes to use filters and otherwise rank routes via next-hop preferences, namely *next-hop preferences with filtering*.

To be able to apply these preferences, each node $v \in V$ is also associated with a path $\mathcal{P}(v)$, called $v$'s *routing path*. The routing path $\mathcal{P}(v)$ may **not** be the same as an actual $v, r$-path in the routing graph. A routing path $\mathcal{P}(v)$ (resp., a node $v$) is *consistent* if $\mathcal{P}(v)$ is a $v, r$-path in the routing graph; otherwise, we say that $\mathcal{P}(v)$ (resp., $v$) is *inconsistent*. A node $v$ is *clear* if the routing path $\mathcal{P}(v) \neq \emptyset$, i.e., $v$ has a path to the sink; otherwise, $v$ is *opaque*. (The node $v$ will never keep a path that is not a $v, r$-path.) We say that a node $w$ is *valid* for $v$ or is a *valid choice* for $v$ if $w$ is clear and $\mathcal{P}(w)$ contains no nodes in the filtering list $\mathcal{D}(w)$. If $w$ is a valid choice for $v$, and $v$ prefers $w$ to all other valid choices, then we say that $w$ is the *best valid choice* of $v$. A basic step in the dynamic behaviour of BGP is that, at any time $t$, some subset $V_t$ of nodes is *activated* meaning that every node $v \in V_t$ chooses the most highest ranked acceptable path $\mathcal{P}(v)$ that is consistent with one of its neighbours' chosen paths at time $t - 1$. The routing graph $\mathcal{R}_t$ consists of the first arc in each routing path at time $t$.

Protocol variations result from such things as restricting $V_t$ so that $|V_t| = 1$, specifying the relative rates that nodes are chosen to be activated and allowing other computations to occur between these basic steps. In our protocol, we assume that activation orderings are *fair* in that each node activates exactly once in each time period – a *round* – the actual ordering however may differ in each round. While our protocol is not intended to model exactly the behaviour of BGP, we tried to let BGP inspires our choices and captures the essential coordination problem that makes successful dynamic routing hard. Again, a detailed discussion on these issues an on the importance of a fairness-type criteria is deferred to the full paper.

**Procedure 1** Activate($v$)

**Input:** A node $v \in V - \{r\}$.
1: **if** $v$ has a valid choice **then**
2: Choose a best valid choice $w$ of $v$.
3: Change the outgoing arc of $v$ to $(v, w)$.
4: Update $\mathcal{P}(v) := v\mathcal{P}(w)$ (the concatenation of $v$ and $\mathcal{P}(w)$).
5: **else**
6: Update $\mathcal{P}(v) := \emptyset$.
7: **end if**

**Procedure 2** Protocol($G$,$r$,$\mathcal{R}_0$)

**Input:** A network $G = (V, A)$, a sink node $r$ and a routing graph $\mathcal{R}_0$

1: Initially, every node generates a packet.
2: **for** round $t = 1$ to $\dots$ **do**
3: Generate a permutation $\pi_t$ of nodes in $V - \{r\}$ using an external algorithm $\mathbb{A}$.
4: **Control Plane:** Apply Activate($v$) to activate each node in the order in $\pi_t$. This forms a routing graph $\mathcal{R}_t$.
5: **Forwarding Plane:** Ask every node to forward packets it has, and wait until every packet is moved by at most $n$ hops (forwarded $n$ times) or gets to the sink.
6: **Route-Verification:** Every node learns which paths it has in the routing graph, i.e., update $\mathcal{P}(v) := v, r$-path in $\mathcal{R}_t$.
7: **end for**

This entire mechanism can thus be described using two algorithms as follows. Once activated, a node $v$ updates its routing path $\mathcal{P}(v)$ using the algorithm in Procedure 1. The generic protocol is described in Procedure 2. This requires an external algorithm $\mathbb{A}$ which acts as a *scheduler* that generates a permutation – an order in which nodes will be activated in each round. We will assume that these permutations are independent and randomly generated. Our subsequent routing guarantees will be derived by showing the existence of specific permutations that ensure all packets route. These permutations are different in each of our models, which differ only in filtering lists. We remark that our model is incorporated with a *route-verification* step, but this is not a feature of BGP (again, see the full version for a discussion).

With the model defined, we examine the efficiency of packet routing for the three cases of next-hop preferences with filtering:

- `General Filtering.` The general case where the filtering list $\mathcal{D}(v)$ of any node $v$ can be an arbitrary subset of nodes.
- `Not me!` The subcase where the filtering list of node $v$ consists only of itself, $\mathcal{D}(v) = \{v\}$. Thus, a node does not want a path through itself, but otherwise has no nodes it wishes to avoid.
- `Anything Goes!` The case where every filtering list is empty, $\mathcal{D}(v) = \emptyset$. Thus a node does not even mind if its packets cycle back through it!

We partition our analyses based upon the types of filtering lists. Our first result is a strong hardness result presented in Section 3. Not only can it be hard to determine if every packet can be routed but the maximum number of packets that can be routed cannot be approximated well even if the network can reach equilibrium. Specifically,

**Theorem 1.** *For filtering lists of cardinality at most one, it is NP-hard to approximate the maximum stable subtree to within a factor of $O(n^{1-\epsilon})$, for any constant $\epsilon > 0$.*

**Corollary 1.** *For filtering lists of cardinality at most one, given a network in which an equilibrium is guaranteed to exist, it is NP-hard to approximate the*

*maximum number of packets that can be routed to within a factor of $O(n^{1-\epsilon})$, for any constant $\epsilon > 0$.*

However, for its natural subcase where the filtering list of a node consists only of itself (that is, a node doesn't want to route via a cycle!), we obtain a positive result in Section 5.

**Theorem 2.** *If the filtering list of a node consists only of itself, then an equilibrium can be obtained in $n$ rounds. However, every packet will be routed in $\frac{n}{3}$ rounds, that is, before stability is obtained!*

Interestingly, we can route every packet in the case $\mathcal{D}(v) = \emptyset$ for all $v \in V$; see Section 4. Thus, even if nodes don't care whether their packets cycle, the packets still get through!

**Theorem 3.** *If the filtering list is empty then every packet can be routed in* 4 *rounds, even when the network has no equilibrium.*

Theorems 2 and 3 are the first theoretical results showing that packet routing can be done in the absence of stability. For example, every packet will be routed even in the presence of dispute wheels. Indeed, packets will be routed even if some nodes *never* actually have paths to the sink. Note that when we say that every packet will route with probability one we mean that, assuming permutations are drawn at random, we will eventually get a fair activation sequence that routes every packet. It is a nice open problem to obtain high probability guarantees for fast packet routing under such an assumption.

## 3 General Filtering.

Here we consider hardness results for packet routing with general filtering lists. As discussed, traditionally the theory community has focused upon the stability of $\mathcal{R}$ – the routing graph is stable if every node is selecting their best valid neighbour (and is consistent). For example, there are numerous intractability results regarding whether a network has an equilibrium. However, that the routing graph may be stable even if it is not spanning! There may be singleton nodes that prefer to stay disconnected rather than take any of the offered routes. Thus, regardless of issues such as existence and convergence, an equilibrium may not even route the packets. This can be particularly problematic when the nodes use filters. Consider our problem of maximising the number of nodes that can route packets successfully. We show that this cannot be approximated to within a factor of $n^{1-\epsilon}$, for any $\epsilon > 0$ unless P = NP. The proof is based solely upon a control plane hardness result: it is NP-hard to approximate the maximum-size stable tree to within a factor of $n^{1-\epsilon}$. Thus, even if equilibria exist, it is hard to determine if there is one in which the *sink-component* (the component of $\mathcal{R}$ containing the sink) is large.

Formally, in the maximum-size stable tree problem, we are given a directed graph $G = (V, E)$ and a sink node $r$; each node $v \in V$ has a ranking of its

neighbours and has a filtering list $\mathcal{D}(v)$. Given a tree (arborescence) $T \subseteq G$, we say that a node $u$ is *valid* for a node $v$ if $(u, v) \in E$ and a $v, r$-path in $T$ does not contain any node of $\mathcal{D}(v)$. We say that $T$ is *stable* if, for every arc $(u, v)$ of $T$, $v$ is valid for $u$, and $u$ prefers $v$ to any of its neighbours in $G$ that are valid for $u$ (w.r.t. $T$). Our goal is to find the stable tree (sink-component) with the maximum number of nodes. We will show that even when $|\mathcal{D}(v)| = 1$ for all nodes $v \in V$, the maximum-size stable tree problem cannot be approximated to within a factor of $n^{1-\epsilon}$, for any constant $\epsilon > 0$, unless P = NP.

The proof is based on the hardness of $3SAT$ [2]: given a CNF-formula on $N$ variables and $M$ clauses, it is NP-hard to determine whether there is an assignment satisfying all the clauses. Take an instance of 3SAT with $N$ variables, $x_1, x_2, \ldots, x_N$ and $M$ clauses $C_1, C_2, \ldots, C_M$. We now create a network $G = (V, A)$ using the following gadgets:

- Var-Gadget: For each variable $x_i$, we have a gadget $H(x_i)$ with four nodes $a_i, u_i^T, u_i^F, b_i$. The nodes $u_i^T$ and $u_i^F$ have first-choice arcs $(u_i^T, a_i)$, $(u_i^F, a_i)$ and second-choice arcs $(u_i^T, b_i)$, $(u_i^F, b_i)$. The node $a_i$ has two arcs $(a_i, u_i^T)$ and $(a_i, u_i^F)$; the ranking of these arcs can be arbitrary. Each node in this gadget has itself in the filtering list, i.e., $\mathcal{D}(v) = \{v\}$ for all nodes $v$ in $H(x_i)$.
- Clause-Gadget: For each clause $C_j$ with three variables $x_{i(1)}, x_{i(2)}, x_{i(3)}$, we have a gadget $Q(C_j)$. The gadget $Q(C_j)$ has four nodes $s_j, q_{1,j}, q_{2,j}, q_{3,j}, t_j$. The nodes $q_{1,j}$, $q_{2,j}$, $q_{3,j}$ have first-choice arcs $(q_{1,j}, t_j)$, $(q_{2,j}, t_j)$, $(q_{3,j}, t_j)$. The node $s_j$ has three arcs $(s_j, q_{1,j})$, $(s_j, q_{2,j})$, $(s_j, q_{3,j})$; the ranking of these arcs can be arbitrary, so we may assume that $(s_j, q_{z,j})$ is a $z$th-choice arc. Define the filtering list of $s_j$ and $t_j$ as $\mathcal{D}(s_j) = \{s_j\}$ and $\mathcal{D}(t_j) = \{d_0\}$. (The node $d_0$ will be defined later.) For $z = 1, 2, 3$, let $u_{i(z)}^T$ be a node in a Var-Gadget $H(x_{i(z)})$; the node $q_{z,j}$ has a filtering list $\mathcal{D}(q_{z,j}) = \{u_{i(z)}^T\}$, if assigning $x_{i(z)} = \mathsf{True}$ satisfies the clause $C_j$; otherwise, $\mathcal{D}(q_{z,j}) = \{u_{i(z)}^F\}$.

To build $G$, we first add a sink node $r$ and a *dummy sink* $d_0$; we connect $d_0$ to $r$ by a first-choice arc $(d_0, r)$. We arrange Var-Gadgets and Clause-Gadgets in any order. Then we add a first-choice arc from the node $a_1$ of the first Var-Gadget $H(x_1)$ to the sink $r$. For $i = 2, 3, \ldots, N$, we add a first-choice arc $(b_i, a_{i-1})$ joining gadgets $H(x_{i-1})$ and $H(x_i)$. We join the last Var-Gadget $H(x_N)$ and the first Clause-Gadget $Q(C_1)$ by a first-choice arc $(t_1, a_N)$. For $j = 2, 3, \ldots, M$, we add a first-choice arc $(t_j, s_{j-1})$ joining gadgets $Q(C_{j-1})$ and $Q(C_j)$. This forms a line of gadgets. Then, for each node $q_{z,j}$ of each Clause-Gadget $Q(C_j)$, we add a second-choice arc $(q_{z,j}, d_0)$ joining $q_{z,j}$ to the dummy sink $d_0$. Finally, we add $L$ *padding* nodes $d_1, d_2, \ldots, d_L$ and join each node $d_i$, for $i = 1, 2, \ldots, L$, to the last Clause-Gadget $Q(c_M)$ by a first-choice arc $(d_i, s_M)$; the filtering list of each node $d_i$ is $\mathcal{D}(d_i) = \{d_0\}$, for all $i = 0, 1, \ldots, L$. The parameter $L$ can be any positive integer depending on a given parameter. Observe that the number of nodes in the graph $G$ is $4N + 5M + L + 2$, and $|\mathcal{D}(v)| = 1$ for all nodes $v$ of $G$. The reduction is illustrated in Figure 3(a).

The correctness of the reduction is proven in the next theorem. The proof is provided in the full version.

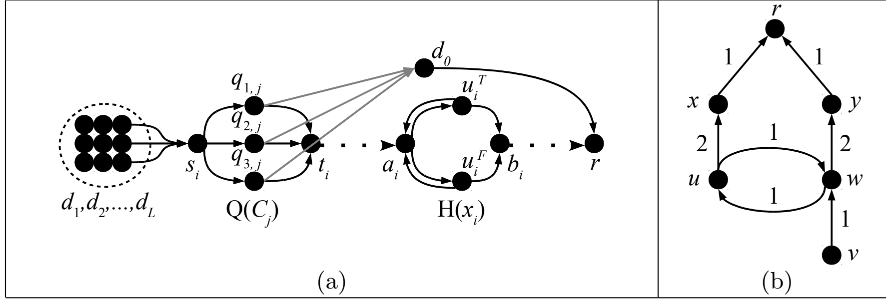


**Fig. 1.** (a) The reduction from 3SAT. (b) A network with no stable solution.

**Theorem 4.** *For any constant $\epsilon > 0$, given an instance of the maximum-size stable tree problems with a directed graph $G$ on $n$ nodes and filtering lists of cardinality $|\mathcal{D}(v)| = 1$ for all nodes $v$, it is NP-hard to distinguish between the following two cases of the maximum-size stable tree problem.*

- YES-INSTANCE: *The graph $G$ has a stable tree spanning all the nodes.*
- NO-INSTANCE: *The graph $G$ has no stable tree spanning $n^\epsilon$ nodes.* □

From the perspective of the nodes, it is NP-hard to determine whether adding an extra node to its filtering list can lead to solutions where none of its packets ever route. In other words, it **cannot** avoid using intermediate node it dislikes!

## 4 Filtering: Anything-Goes!

Here we consider the case where every node has an empty filtering list. This case is conceptually simple but still contains many technical difficulties involved in tracking packets when nodes become mistaken in their connectivity assessments. In this case, networks with no stable solutions can exist (for example, see Figure 3(b)), and there can be fair activation sequences under which a node will never be in the sink-component. We show, however, that even in such circumstances, every packet still reaches the sink, and this is the case for all networks. Specifically, we present a fair activation sequence of four rounds that routes every packet, even when there is no equilibrium.

When filtering lists are empty, a node $v$ only needs to known whether its neighbour $u$ has a path to the sink since $v$ does not have any node it dislikes. Thus, we can view each node as having two states: clear or opaque. A node is *clear* if it is in the routing-tree (the nomenclature derives from the fact that a packet at such a node will then reach the sink – that is, "clear"); otherwise, a node is *opaque*. Of course, as nodes update their chosen next-hop over time, they may be mistaken in their beliefs (inconsistent) as the routing graph changes. In other words, some clear nodes may not have "real" paths to the sink. After the learning step at the end of the round, these clear-opaque states are correct again.

Our algorithm and analysis are based on properties of the network formed by the first-choice arcs, called the *first class network*. We say that an arc $(u, v)$ of $G$ is a *first-choice* arc if $v$ is the most preferred neighbour of $u$. We denote the first class network by $F = (V, A_1)$, where $A_1$ are the first-choice arcs. As in a routing graph $\mathcal{R}$, every node in $F$ has one outgoing arc. Thus, every component of $F$ is a 1-*arborescences*, a tree-like structure with either a cycle or a single node as a root. We denote the components of $F$ by $F_0, F_1, \ldots, F_\ell$, where $F_0$ is the component containing the sink $r$. Observe that, when activated, every node in $F_0$ will always choose its neighbour in $F_0$. So, we may assume wlog that $F_0$ is a singleton. Each $F_j$ has a unique cycle $C_j$, called a *first class cycle* (We may assume the directed cycle in $F_0$ is a self-loop at the sink $r$.) The routing graph at the beginning of Round $t$ is denoted by $\mathcal{R}_t$. We denote by $\mathcal{K}_t$ and $\mathcal{O}_t$ the sets of clear and the set of opaque nodes at the start of Round $t$. Now, we will show that there is an activation sequence which routes every packet in four rounds.

The proof has two parts: a *coordination phase* and a *routing phase*. In the first phase, we give a coordination algorithm that generates a permutation that gives a red-blue colouring of the nodes with the following three properties: (i) For each $F_j$, every node in $F_j$ has the same colour, i.e., the colouring is coordinated. (ii) If the first class cycle $C_j$ of $F_j$ contains a clear node then all nodes in $F_j$ must be coloured blue. (iii) Subject to the first two properties the number of nodes coloured red is maximised. The usefulness of this colouring mechanism lies in the fact that the corresponding permutation is a fair activation sequence that will force the red nodes to lie in the sink-component and the blue nodes in non-sink-components. Moreover, bizarrely, running this coordination algorithm four times in a row ensures that every packet routes! So in the second phase, we run the coordination algorithm three more times.

---

**Procedure 3** Coordinate($\mathcal{K}_t$)

---

**Input:** A set of clear nodes $\mathcal{K}_t$.
**Output:** A partition $(R, B)$ of $V$.

1: Let $B_0 := \bigcup_{i: V(C_j) \cap \mathcal{K}_t \neq \emptyset} V(F_i)$ be a set of nodes containing in an $F$-component whose first class cycle $C_i$ has a clear node.
2: Initialise $q := 0$.
3: **repeat**
4:     Update $q := q + 1$.
5:     Initialise $B_q := B_{q-1}$, $R_q := \{r\}$ and $U := V - (R_q \cup B_q)$.
6:     **while** $\exists$ a node $v \in U$ that prefers a node in $R_q$ to nodes in $B_q \cup (U \cap K_t)$ **do**
7:         Move $v$ from $U$ to $R_q$.
8:     **end while**
9:     **while** $\exists$ a node $v \in U$ that prefers a node in $B_q$ to nodes in $R_q \cup (U \cap K_t)$ **do**
10:         Move $v$ from $U$ to $B_q$.
11:     **end while**
12:     Move $U \cap \mathcal{K}_t$ from $U$ to $B_q$.
13: **until** $B_q = B_{q-1}$.
14: **return** $(R_q, B_q)$.

---

### 4.1 Coordination Phase.

The algorithm Coordinate($\mathcal{K}_t$) in Procedure 3 constructs a red-blue colouring of the nodes, i.e. the final partition $(R, B)$ of $V$. At the termination of Coordinate($\mathcal{K}_t$), by the construction, any node $v \in R$ prefers some node in $R$ to any node $w \in B$, and any node $v \in B$ prefers some node in $B$ to any node $w \in R$.

Given a partition $(R, B)$, we generate an activation sequence as follows. First, we greedily activate nodes in $R - \{r\}$ whenever their most-preferred *clear* neighbours are in $R$. (We activate nodes of $R$ in the same order as we constructed $R$.) This forms a sink-component on $R$. Next, we activate nodes in $B$. We start by activating nodes in $B_0 = \bigcup_{i:C_i \cap K_i \neq \emptyset} V(F_i)$ – the components of $F$ whose first class cycles contain at least one clear node. For each $F_i$ with $V(F_i) \subseteq B_0$, take a clear node $v \in C_i \cap \mathcal{K}_t$. Then activate the nodes of $F_i$ (except $v$) in an increasing order of distance from $v$ in $F_i$, and after that we activate $v$. This forms a non-sink-component $F_i$ in the routing graph as every node can choose its first-choice. Finally, we activate nodes in $B - B_0$ whenever their most-preferred *clear* neighbours are in $B$ (we use the same order as in the construction of $B$). This creates non-sink-components on $B$ and implies the next lemma.

**Lemma 1.** *Let $\pi_t$ be an activation sequence generated from $(R, B)$ as above. At the end of the round, the following hold:*

- *The sink-component includes $R$ and excludes $B$.*
- `Coordination:` *For each $F_i$, either all the nodes of $F_i$ are in the sink-component or none of them are.*
- *Let $B_0 = \bigcup_{i:V(C_i) \cap \mathcal{K}_t \neq \emptyset} V(F_i)$, and suppose $\mathcal{K}_t = B_0$. If a packet travels for $n$ hops but does not reach the sink, then it must be at a node in $\mathcal{K}_t$.*

*Proof.* The first statement follows from the construction. For the second statement, it suffices to show that, for each $F_i$, either $V(F_i) \subseteq R$ or $V(F_i) \subseteq B$. Suppose not. Then there is a $V(F_i)$ crossing $R$ or $B$. But, then some node in $R$ (respectively, $B$) would have a first-choice in $B$ (respectively, $R$), and this is not possible by the construction of $(R, B)$.

For the third statement, note that a packet that travels for $n$ hops but does not reach the sink must be stuck in some cycle. Consider the construction of $(R, B)$. Since $\mathcal{K}_t = B_0$, we only add node to $B$ whenever it prefers some node in $B$ to any node in $R$. Because $U \cap \mathcal{K}_t = \emptyset$, nodes in $B - B_0$ cannot create a cycle on their own. Thus, the packet is stuck in a cycle that contains a clear node; the only such cycles are the first class cycles of $B_0$ since $\mathcal{K}_t = B_0$. □

The following lemma follows by the construction of a partition $(R, B)$.

**Lemma 2.** *Let $(R', B')$ be any partition generated from Coordinate(.), and let $(R, B)$ be a partition obtained by Coordinate($\mathcal{K}_t$), where $\mathcal{K}_t \subseteq B'$. Then $R' \subseteq R$.*

*Proof.* Consider a partition $(R_q, B_q)$ constructed during a call to Coordinate($\mathcal{K}_t$). Observe that $B_0 \subseteq B'$ because $B_0 = \bigcup_{i:V(C_j) \cap \mathcal{K}_t \neq \emptyset} V(F_i)$ and Lemma 1 implies

that each $F_1$ is contained entirely in $R'$ or $B'$. By the construction of $(R', B')$, since $B_0 \subseteq B'$, every node of $R'$ must have been added to $R_1$, i.e., $R' \subseteq R_1$. Inductively, if $R' \subseteq R_q$ for some $q \geq 1$, then $B_q \subseteq B'$ and thus $R' \subseteq R_{q+1}$ by the same argument. □

### 4.2 Routing Phase: A Complete Routing in Four Rounds.

Running the coordination algorithm four times ensures every packet will have been in the sink-component at least once, and thus, every packet routes.

**Theorem 5.** *In four rounds every packet routes.*

*Proof.* The first round $t = 1$ is simply the coordination phase. We will use subscripts on $R$ and $B$ (e.g., $R_t$ and $B_t$) to denote the final colourings output in each round and not the intermediate sets $R_q/B_q$ used in Coordinate(.). Now, consider a packet generated by any node of $V$. First, we run Coordinate($\mathcal{K}_1$) and obtain a partition $(R_1, B_1)$. By Lemma 1, if the packet is in $R_1$, then it is routed successfully, and we are done. Hence, we may assume that the packet does not reach the sink and the packet is in $B_1$. Note that, now, each $F_i$ is either contained in $R_1$ or $B_1$ by Lemma 1.

We now run Coordinate($\mathcal{K}_2$) and obtain a partition $(R_2, B_2)$. By Lemma 1, $\mathcal{K}_2 = R_1$. So, if the packet does not reach the sink, it must be in $B_2$. Since no $F$-component crosses $R_1$, we have $R_1 = \mathcal{K}_2 = \bigcup_{i:V(C_i)\cap\mathcal{K}_2 \neq \emptyset} V(F_i)$. So, $R_1 \subseteq B_2$ (since $\mathcal{K}_2 \subseteq B_2$) and $R_2 \subseteq B_1$, and Lemma 1 implies that the packet is in $R_1$.

Third, we run Coordinate($\mathcal{K}_3$) and obtain a partition $(R_3, B_3)$. Applying the same argument as before, we have that the packet is in $R_2$ (or it is routed), $R_2 \subseteq B_3$ and $R_3 \subseteq B_2$. Now, we run Coordinate($\mathcal{K}_4$) and obtain a partition $(R_4, B_4)$. Since $R_3 = \mathcal{K}_4 \subseteq B_2$, Lemma 2 implies that $R_2 \subseteq R_4$. Thus, the packet is routed successfully since $R_4$ is contained in the sink-component. □

## 5 Filtering: Not-Me!

In practice, it is important to try to prevent cycles forming in the routing graph of a network. To achieve this, *loop-detection* is implemented in the BGP-4 protocol [3]. The "Not-Me" filtering encodes loop-detection as in the BGP-4 protocol simply by having a filtering list $\mathcal{D}(v) = \{v\}$, for every node $v$. In contrast to Theorem 4, which says that it is NP-hard to determine whether we can route every packet, here we show that every packet will route. Moreover, we exhibit a constructive way to obtain a stable spanning tree via fair activation sequences. Interestingly, all of the packets will have routed before stability is obtained. In particular, we give an algorithm that constructs an activation sequence such that every packet routes successfully in $\frac{1}{3}n$ rounds, and the network itself becomes stable in $n$ rounds. This is the most complex result in the paper; just the algorithm itself is long. So here we give a very high level overview and defer the algorithm and its proof of performance to the full paper.

When filtering lists are non-empty, a complication arises since even if $w$ is the most preferred choice of $v$ and $w$ has non-empty routing path, $v$ still may not be able to choose $w$. This makes the routing graph hard to manipulate. The key idea is to manipulate the routing graph a little-by-little in each round. To do this, we find a spanning tree with a *strong stability property* – a spanning tree $S$ has the strong stability property on $\mathbb{O}$ if, for every node $v \in \mathbb{O}$, the most preferred choice of $v$ is its parent $w$, even if it may choose any node except those that are descendants and in $\mathbb{O}$. Thus, if we activate nodes of $S$ in increasing order of distance from the sink $r$, then every node $v \in \mathbb{O}$ will always choose $w$.

It is easy to find a *stable spanning tree*, a tree where no node wants to change its choice, and given a stable spanning tree $S$, it is easy to force opaque nodes in $\mathcal{O}_t$ to make the same choices as in $S$. But, this only applies to the set of opaque nodes, so it may not hold in the later rounds. The strong stability property allows us to make a stronger manipulation. Intuitively, the strong stability property says that once we force every node $v \in \mathbb{O}$ to make the same choice as in $S$, we can maintain these choices in all the later rounds. Moreover, in each round, if we cannot route all the packets, then we can make the strong stability property span three additional nodes; if so, the property spans one more node. Thus, in $\frac{1}{3}n$ rounds, every packet will route, but we need $n$ rounds to obtain stability.

**Theorem 6.** *There is an activation sequence that routes every packet in* $\lfloor n/3 \rfloor$ *rounds and gives a stable spanning tree in* $n$ *rounds.*

**Acknowledgements** We thank Michael Schapira and Sharon Goldberg for interesting discussions on this topic.

## References

1. Timothy Griffin, F. Bruce Shepherd, and Gordon T. Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Trans. Netw.*, 10(2):232–243, 2002.
2. Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.
3. John W. Stewart, III. *BGP4: Inter-Domain Routing in the Internet.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.
4. Stefano Vissicchio, Luca Cittadini, Laurent Vanbever, and Olivier Bonaventure. ibgp deceptions: More sessions, fewer routes. In *INFOCOM*, 2012.