# Routing regardless of Network Stability

Bundit Laekhanukit* Adrian Vetta† Gordon Wilfong‡

October 24, 2013

**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 requires modelling the interactions between the control plane and the forwarding plane (which determines where 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 $n^{1-\epsilon}$, for any constant $\epsilon > 0$. On the positive side, we give algorithms to show that, in two fundamental cases, there exist activation sequences under which *every* packet will route. The first case is when each node's filtering list contains only itself, that is, $\mathcal{D}(v) = \{v\}$; this is the fundamental case in which a node does not want its packets to cycle. Moreover, 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, every packet will route even when the nodes do not care if their packets cycle! Furthermore, under these activation sequences, every packet will route even when the control plane has *no* equilibrium at all. Our positive results require the periodic application of *route verification*. 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) [5], 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 $r$. The

*School of Computer Science, McGill University. Supported by a Dr. and Mrs. Milton Leong Fellowship and by NSERC grant 288334. Email: blaekh@cs.mcgill.ca

†Department of Mathematics and Statistics and School of Computer Science, McGill University. Supported in part by NSERC grants 288334 and 429598. Email: vetta@math.mcgill.ca

‡Bell Laboratories. Email: gtw@research.bell-labs.com

lowest ranked entry in the list is the "empty path"[1]; paths that are not ranked are considered unsatisfactory. 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 $v\mathcal{P}(w)$ 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 $r$.

There are two major drawbacks to this approach that motivate our work. First, even if a stable solution exists, packets may not reach the sink because the corresponding stable routing tree is not spanning. Second, what happens to the packets before the network converges (if it does at all) to a stable solution? In particular, the main focus of our paper is to investigate packet routing under network dynamics.

To understand the basic combinatorial model, we use to do this. Observe that each node uses it preference list to select at most one outgoing arc. Given an initial routing graph, we will allow each node, one at a time, to change its outgoing arc based upon its preferences and current information. We call this process a *round*. During the course of a round inconsistencies may form and the resultant new routing graph may contain cycles – packets that do not reach the sink may now be at nodes in these cycles. At the end of the round, we assume that the nodes learn the new routing graph (route verification) and we repeat the process. We want to know if every packet will eventually reach the sink.

Given this brief overview, let's elaborate on this discussion and the relationships to BGP.

(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 offered 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 [19]. In Section 3, we show that non-connectivity is a very serious problem (at least, from the theoretical side) by presenting an $n^{1-\epsilon}$ 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).[2] Thus, it is important to study network performance in the dynamic case. For example, what happens to the packets whilst a network is unstable?

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

[2] The control plane is where BGP exchanges routing information between routers. The forwarding plane is where routers use the routing information to forward packets towards their destinations.

In our model, 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. Under this framework we analyse the resulting trajectory of packets.

We have seen that 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) there exist activation sequences for which every packet will route. That is, there is an execution of 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.[3] 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 packets eventually route is not possible – thus, packets must be lost.

To avoid overloading the reader with practical technicalities, we focus in the main text on the combinatorial aspects of packet routing. We refer the interested reader to the Appendix where we discuss more technical aspects and present a motivating sample of the vast literature on BGP. We emphasise here, however, one key distinction between our model and BGP. Our results require the period application of *route verification* (that is, the learning phase at the end of each round). Route verification is not a standard feature of BGP.

## 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 at least one directed path in $G$ from $v$ to the sink $r$, and that 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* $\mathcal{N}_t$, 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 $T$; for example, $T$ 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; all other components are called *non-sink components*.

[3] 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.

[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.

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*. 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 the set of arcs $A$, i.e., $A = A_1 \cup A_2 \cup \ldots \cup A_n$. 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. We say that a routing path $\mathcal{P}(v)$ is *consistent* if $\mathcal{P}(v)$ is a $v, r$-path in the routing graph; otherwise, we say that $\mathcal{P}(v)$ is *inconsistent*. Similarly, we say that a node $v$ is *consistent* if its routing path $\mathcal{P}(v)$ is consistent; otherwise, we say that $v$ is *inconsistent*. A node $v$ is *clear* if the routing path $\mathcal{P}(v) \neq \emptyset$, i.e., $v$ believes it has a path to the sink; otherwise, $v$ is *opaque*. We say that a neighbouring 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*; this means that every node $v \in V_t$ chooses the highest ranked acceptable path $\mathcal{P}(v)$ that is consistent with one of its neighbours' routing paths at time $t - 1$. The routing graph $\mathcal{N}_t$ consists of the first arc in each routing path at time $t$.

See Figure 1 for an example of how the protocol works, and how inconsistent information may arise and cycles form in the routing graph.

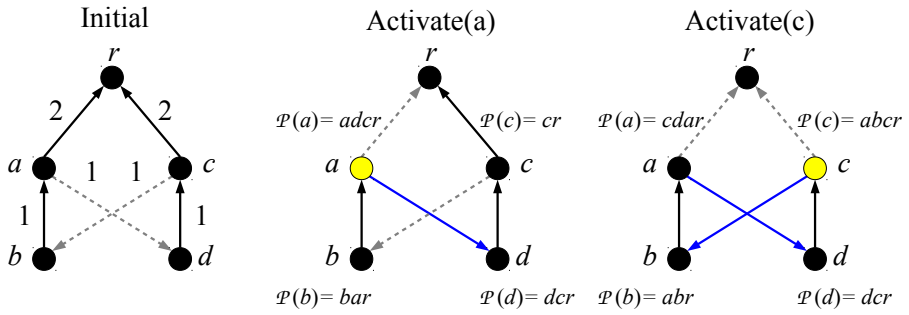


Figure 1: Given the initial consistent state, activate the node $a$ and then $c$. The information becomes inconsistent and a cycle forms in the routing graph.

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 inspire our choices and to capture the essential coordination problem that makes successful dynamic routing hard. A detailed discussion on these issues and on the importance of a

fairness-type criteria is deferred to the Appendix.

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

**Input:** A node $v \in V - \{r\}$.

1: **if** $v$ has a valid choice **then**
2: Choose the 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($G$,$r$,$\mathcal{N}_0$)

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

1: Initially, every node generates a packet.
2: **for** round $t := 1$ to $\ldots$ **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 $\mathcal{N}_t$.
5: **Forwarding Plane:** Ask every node to forward the packets it has, and wait until every packet is moved by at least $n$ hops (forwarded $n$ times) or gets to the sink.
6: **Route-Verification:** Every node learns which path it has in the routing graph, i.e., update $\mathcal{P}(v) := v, r$-path in $\mathcal{N}_t$.
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 the filtering lists. Again, we remark that our model is incorporated with a *route-verification* step, but this is not a feature of BGP (see the Appendix 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!

### 2.1 Our Results.

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 cardinality stable subtree to within a factor of $n^{1-\epsilon}$, for any constant $\epsilon > 0$.*

**Corollary 2.** *Suppose a network eventually reaches a stable state, the protocol runs for infinitely many rounds, and every node generates a single packet in each round. For filtering lists of cardinality at most one, it is NP-hard to approximate the maximum number of packets that can be routed to within a factor of $n^{1-\epsilon}$, 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 does not want to route via a cycle!), we obtain a positive result in Section 5.

**Theorem 3.** *If the filtering list of each node consists only of itself, then there exists an activation sequence that produce a stable spanning tree in $n$ rounds. Moreover, every packet will be routed in $\frac{n}{2}$ 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 do not care whether their packets cycle, the packets still get through!

**Theorem 4.** *If the filtering list is empty, then there exists an activation sequence that routes every packet in 3 rounds, even when the network has no equilibrium.*

Theorems 3 and 4 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* [5]. Indeed, packets will be routed even if some nodes *never* actually have paths to the sink. These results imply that if the permutations for each round are drawn independently and uniformly at random, then every packet will eventually route with probability one. 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 $\mathcal{N}$ – 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; e.g., see [6, 2]. However, notice 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 cardinality 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 $\mathcal{N}$ containing the sink) is large.

Formally, in the *maximum cardinality 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 $T \subseteq G$ (note that by "tree", we mean an in-arborescence), we say that a node $v$ is *valid* for a node $u$ 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, say $(u, v)$, of $T$ we have that $v$ 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 a 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 [10]: 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:

- `Variable-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 five 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 $\mathcal{D}(s_j) = \{d_0\}$ and $\mathcal{D}(t_j) = \{d_0\}$. (The node $d_0$ will be defined later.) For $z = 1, 2, 3$, let $u_{i(z)}^T$ and $u_{i(z)}^F$ be nodes in the corresponding Variable-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 $x_{i(z)} = \mathsf{False}$ 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 then connect $d_0$ to $r$ by a first-choice arc $(d_0, r)$. We number the Variable-Gadgets and Clause-Gadgets in any order. Then we add a first-choice arc from the node $a_1$ of the first Variable-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 Variable-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 2.

The correctness of the reduction is proven in the next theorem.

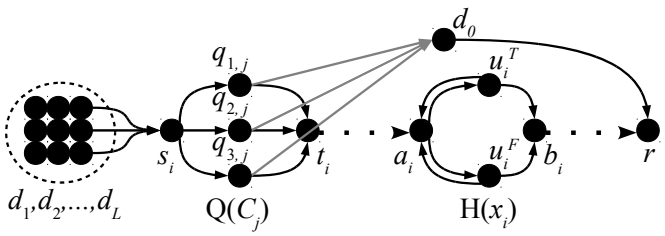


Figure 2: The hardness construction.

**Theorem 5.** *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 cardinality 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.*

*Proof.* We apply the above reduction from 3SAT with a parameter $L = J^{1/\epsilon} - J$, where $J = 4N + 5M + 2$. Thus, the graph $G$ has $n = J^{1/\epsilon}$ nodes and has $n^\epsilon = J$ non-padding nodes.

First, we show that there is a one-to-one mapping between choices of each Variable-Gadget $H(x_i)$ and an assignment of $x_i$. Consider any Variable-Gadget $H(x_i)$. To connect to the next gadget, nodes $u_i^T$ and $u_i^F$ of $H(x_i)$ must choose at least one second-choice arc. However, in a stable tree, they cannot choose both second-choice arcs $(u_i^T, b_i)$ and $(u_i^F, b_i)$; otherwise, $u_i^T$ or $u_i^F$ would prefer to choose the node $a_i$. Thus, the gadget $G(x_i)$ must choose either arcs

$$(1)\ (u_i^T, b_i), (u_i^F, a_i), (a_i, u_i^T) \quad \text{or} \quad (2)\ (u_i^F, b_i), (u_i^T, a_i), (a_i, u_i^F).$$

These two cases correspond to the assignments $x_i = \mathsf{True}$ and $x_i = \mathsf{False}$, respectively. Thus, there is a one-to-one mapping between the choices of gadget $H(x_i)$ in the stable tree and the assignment of $x_i$. We refer to each of these two alternatives as an assignment of $x_i$.

Now, we prove the correctness of the reduction.

YES-INSTANCE: Suppose there is an assignment satisfying all the clauses. We will construct a stable spanning tree $T$ corresponding to such an assignment. Within Variable-Gadget, we select arcs in accordance with the assignment as detailed above. We also choose the arc $(d_0, r)$ and all the horizontal arcs connecting adjacent gadgets in the line (or from the first Variable-Gadget to the sink $r$). For each Clause-Gadget $Q(C_j)$ and each $z = 1, 2, 3$, we choose the first-choice arc $(q_{z,j}, t_j)$ if the assignment to $x_{i(z)}$ satisfies $C_j$; otherwise, we choose the second-choice arc $(q_{z,j}, d_0)$. For the node $s_j$ of $Q(C_j)$, we choose an arc $(s_j, q_{z,j})$, where $z$ is the smallest number such that the assignment to $x_{i(z)}$ satisfies $C_j$ (i.e., $q_{z,j}$ chooses $t_j$); since the given assignment satisfies all the clauses, $s_j$ has at least one valid choice. Also, $s_j$ makes the best valid choice (and thus is stable) because $\mathcal{D}(s_j) = \{d_0\}$. Now, we have that the node $s_M$ of the last Clause-Gadget $Q(C_j)$ has a path $\mathcal{P}(s_M)$ to the sink $r$ that does not contain the dummy sink $d_0$. Thus, every padding node can choose $s_M$ and, therefore, is in the stable tree $T$. This implies that $T$ spans all the nodes.

NO-INSTANCE: Suppose there is no assignment satisfying all the clauses. Let $T$ be any stable tree of $G$. As in the previous discussion, the choices of nodes in Variable-Gadgets correspond to the

assignment of variables of 3SAT.

Consider any Clause-Gadget $Q(C_M)$. Since $\mathcal{D}(s_M) = \{d_0\}$, the node $s_M$ of $Q(C_M)$ has a path to the sink $r$ only if

(1) a $t_M, r$-path $\mathcal{P}(t_M)$ in $T$ does not contain the dummy sink $d_0$, and

(2) one of $q_{1,M}, q_{2,M}, q_{3,M}$ chooses $t_M$.

We claim that these two conditions hold only if $T$ corresponds to an assignment satisfying $C_j$. Suppose the first condition holds. Then $\mathcal{P}(t_M)$ has to visit either $v_i^F$ or $v_i^T$ of every Variable-Gadget $H(x_i)$, depending on the assignment of $x_i$. Thus, by the construction of $\mathcal{D}(q_{z,M})$, $t_M$ is valid for $q_{z,M}$ only if the assignment to $x_{i(z)}$ satisfies $C_M$. Since there is no assignment satisfying all the clauses, a node $s_\ell$ of some Clause-Gadget $Q(C_\ell)$ is not in $T$. This means that nodes in the remaining Clause-Gadget have to use the dummy sink $d_0$ to connect to the sink $r$. Thus, the node $s_M$ of the last Clause-Gadget $Q(C_M)$ is not in $T$ and neither are any of the padding nodes $d_1, d_2, \ldots, d_L$. Therefore, the size of $T$ is at most $J = n^\epsilon$, proving the theorem. □

Corollary 2 then follows. Moreover, it also follows that, 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 an intermediate node it dislikes! More precisely, consider the following *filtering* problem. For a specific node $v \in V - \{r\}$ and a subset of nodes $S_v \subseteq V - \{v, r\}$, can we add a node from $S_v$ to the filtering list $\mathcal{D}(v)$ so that $v$ can still connect to the sink at equilibrium? Using the construction above, it can be shown that this problem is NP-complete. In particular, we can choose $v = a_N$ and $S_v = \{u_N^T, u_N^F\}$; the problem is then equivalent to 3SAT.

# 4 Filtering: Anything-Goes!

Now we present our positive results. In this section, we consider the case where every node has an empty filtering list. This case is conceptually simpler than the case of non-empty filtering lists, but there are still many technical difficulties involved in tracking packets when nodes become mistaken in their connectivity assessments. In this case, networks with no equilibrium can exist. Figure 3 presents such an example. Moreover, in this example, fair activation sequences exist where the node $v$ will never be in the sink component. For example, suppose we start with the routing graph $\mathcal{N}_0 = \{(x, r), (y, r), (u, x), (w, y)\}$ and then repeatedly activate nodes according to the permutation $(v, u, w, x, y)$. The routing graph at the end of Round 1 is then $\mathcal{N}_1 = \{(x, r), (y, r), (u, w), (w, u), (v, w)\}$, and at the end of Round 2 is $\mathcal{N}_2 = \mathcal{N}_0$. Thus, the routing graph never stabilises and the node $v$ is never in the sink component. Despite this, every packet will route in two rounds! To see this, observe that a packet that originates at $v$ will be stuck in the cycle $\{u, w\}$ at the end of Round 1. But, both of these nodes are in the sink-component at the end of Round 2, so the packet will then reach the sink. This example nicely illustrates the need to track packets if we want to understand the efficacy of BGP-like protocols. This can be achieved for this class of preference functions. Specifically, we present a fair activation sequence of three rounds that routes every packet, even when there is no equilibrium.

Observe that when filtering lists are empty, a node $v$ only needs to know whether its neighbour $u$ has a path to the sink, as $v$ will never discount a path because it contains a 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

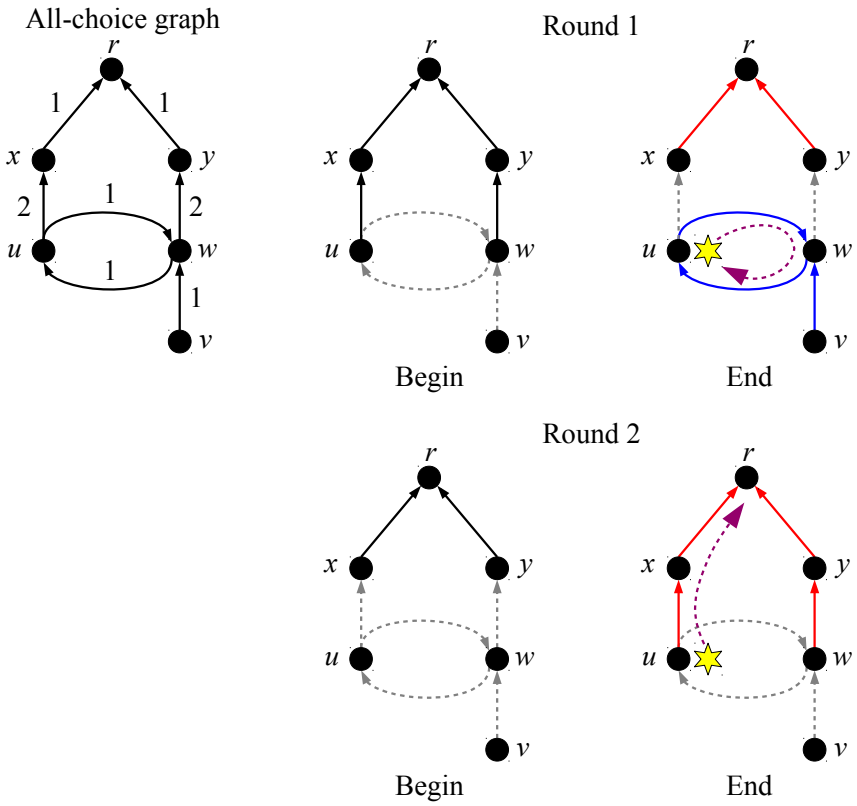


Figure 3: A network with no stable spanning tree. The top-left graph is the all-choice graph where arc numbers indicate rankings (e.g., the number 2 on the arc $(u, x)$ means that $x$ is the second choice of $u$). Two rounds using the activation sequence $(v, u, w, x, y)$ are shown on the right, along with the location of a packet originating at $v$.

sink-component (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 become correct again.

In this section, we show there is an activation sequence which routes every packet in three rounds. For each round, the activation permutation produces a red-blue colouring of the nodes. At the end of the round, the red nodes will be forced to lie in the sink-component and the blue nodes forced to lie in non-sink components. Intuitively, therefore, to route the packets we desire a permutation (activation sequence) that produces as many red nodes as possible. The problem with this basic approach is that any un-routed packet will now be located in a cycle formed by the blue nodes, that is, the non-sink components. But, if we are not careful the locations of such cycles are essentially arbitrary and, consequently, keeping track of the packets becomes extremely difficult.

To rectify this, we will design the permutation judiciously so that we are able to restrict the collection of cycles in which a packet can become stuck – these cycles are the collection of *first-class cycles* (defined below). Given this structure, we will be able to keep track of the packets and prove

that three rounds are sufficient to route each packet.

### 4.1 A First Class Network.

As discussed, our algorithm will keep track of the packets by maintaining a relationship between the current routing graph and 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 $\mathcal{F} = (V, A_1)$, where $A_1$ is the set of the first-choice arcs. As in a routing graph $\mathcal{N}$, every node in $\mathcal{F}$ has one outgoing arc. Thus, every component of $\mathcal{F}$ is a 1-*arborescence*, a tree-like structure with either a cycle or a single node as a root. We denote the components of $\mathcal{F}$ by $F_0, F_1, \ldots, F_k$, where $F_0$ is the component containing the sink $r$. Each $F_j$ has a unique cycle $C_j$, called a *first class cycle*. We may assume the first class cycle in $F_0$ is a self-loop at the sink $r$; we may also assume that $F_0$ is the singleton node $r$ because we can easily make nodes in $F_0$ choose their first-choices by activating them in increasing distance from the sink. (Furthermore, after doing so $F_0$ becomes a stable subgraph.) Thus, in what follows, we need not take $F_0$ into account; so, when referring to first-class components, we will mean $F_1, F_2, \ldots, F_k$. The routing graph at the beginning of round $t$ is denoted by $\mathcal{N}_t$. We denote by $\mathcal{K}_t$ and $\mathcal{O}_t$ the set of clear and the set of opaque nodes at the start of round $t$.

We now describe how to route every packet in three rounds. The proof has two parts: a *coordination phase* and a *routing phase*. In the first phase, in Section 4.2, we give a coordination procedure that generates a permutation producing a red-blue colouring of the nodes that is "coordinated" with the first-class network. In the second phase, in Section 4.3, we show that two further applications of the coordination procedure will route every packet.

### 4.2 The Coordination Phase.

The algorithm Coordinate($\mathcal{K}$) presented in Procedure 3 constructs a red-blue colouring of the nodes. (Throughout the paper, we assume that all network information and all preferences lists are part of the input to our procedures.) Furthermore, this partition $(R, B)$ of $V$ (where $v \in R$ means that $v$ is coloured red and $v \in B$ means that $v$ is coloured blue) also has the following properties:

(i) The colouring is *coordinated*: for each $F_j$, every node in $F_j$ has the same colour. That is, the first choice of any node $v \in R$ is also in $R$ and the first choice of any node $u \in B$ is also in $B$.

(ii) If the first class cycle $C_j$ of $F_j$ contains a clear node, then every node in $F_j$ must be coloured blue.

Some remarks are in order here. First, nodes are clear if they are in the sink component at the start of the current round (end of the previous round). Otherwise, they are opaque. Nodes will be red if they are in the sink component at the end of the current round (start of the next round). Otherwise, they will be blue. Second, as discussed, Properties (i) and (ii) will be useful to us as they will allow us to (approximately) keep track of the packets. Now, observe that the initial partition $(R^0, B^0)$ from Line 1 of Coordinate($\mathcal{K}$) does satisfy (i) and (ii). However, the reason we cannot just stop there is that we cannot create from $\mathcal{K}$ an activation sequence (permutation) that will produce the partition $(R^0, B^0)$. Thus, the remainder of the procedure Coordinate($\mathcal{K}$) is designed to construct a partition $(R^{q^*}, B^{q^*})$ for which we can construct a feasible activation sequence.

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

**Input:** A set of clear nodes $\mathcal{K}$.

**Output:** A partition $(R, B)$ of $V$.

Let $B^0 := \bigcup_{i \geq 1: V(C_i) \cap \mathcal{K} \neq \emptyset} V(F_i)$ be the set of nodes contained in an $F$-component whose first class cycle $C_i$ has a clear node, and let $R^0 = V - B^0$.
Initialise $q := 0$.
**repeat** {Outer Iteration}
  Initialise $\ell := 0$, $\mathcal{R}^q_0 = \{r\}$ and $U^q_0 := V - (B^q \cup \{r\}) = R^q - \{r\}$.
  **while** $\exists$ a node $u^q_\ell \in U^q_\ell$ that prefers some node $\tau^q_\ell$ in $\mathcal{R}^q_\ell$ to every node in $B^q \cup (U^q_\ell \cap \mathcal{K})$ **do** {Inner Iteration}
    $\mathcal{R}^q_{\ell+1} := \mathcal{R}^q_\ell \cup \{u^q_\ell\}$.
    $U^q_{\ell+1} := U^q_\ell - \{u^q_\ell\}$.
    Update $\ell := \ell + 1$.
  **end while**
  Define $\ell^q_* := \ell$.
  Let $B^{q+1} := B^q \cup U^q_{\ell^q_*}$ and $R^{q+1} := \mathcal{R}^q_{\ell^q_*}$.
  Update $q := q + 1$.
**until** $B^q = B^{q-1}$.
Define $q_* := q$;
**return** $(R^{q_*}, B^{q_*})$.

Second, observe that Coordinate($\mathcal{K}$) contains many loops. Thus, we essentially derive $(R^{q^*}, B^{q^*})$ via $q^*$ permutations. Of course, we wish to generate a single activation sequence $\pi$ from $(R^{q^*}, B^{q^*})$. We will show how such a permutation $\pi$ can be obtained in Section 4.2.4.

### 4.2.1 An Example Run.

An example run of the procedure Coordinate is shown in Figures 4 and 5.

Figure 4 illustrates a run of the inner iteration. In the figure, the green arcs are the arcs in the routing graph; the green nodes are are the nodes that were clear at the beginning of the round, that is, the nodes in $\mathcal{K}$. The solid arcs are first-choice arcs, and the dashed arcs are second-choice arcs. The inner loop starts by defining sets $B^0$ and $\mathcal{R}^0_0 = \{r\}$. Then the procedure grows $\mathcal{R}^0_1, \mathcal{R}^0_2, \mathcal{R}^0_3, \ldots, \mathcal{R}^0_{13}$, and the loop terminates with $\mathcal{R}^0_{13}$ since, at this point, no node prefers a node in $\mathcal{R}^0_{13}$ to every node in $B_0 \cup (\mathcal{K} \cap U^0_{13})$. Thus, the procedure stops and constructs $B^1$ and $R^1$.

Figure 5 illustrates the run of the outer iteration. The outer iteration constructs partitions $(B^0, R^0), (B^1, R^1), (B^2, R^2), \ldots$ where with each iteration the red set $R^q$ decreases its size. The loop then terminates when $R^3$ remains the same set as $R^2$. Observe that the outer iteration terminates with the partition $(B^3, R^3)$ of $V$.

### 4.2.2 Key Facts.

We begin by proving a collection of facts about the ordered-pairs $(R^1, B^1), (R^2, B^2), \ldots, (R^{q_*}, B^{q_*})$ constructed in the procedure Coordinate($\mathcal{K}$).

**Fact 6.** *$(B^q, U^q_\ell, \mathcal{R}^q_\ell)$ is a partition of $V$, for all $q = 0, 1, \ldots, q_*$ and all $\ell = 0, 1 \ldots, \ell^q_*$.*

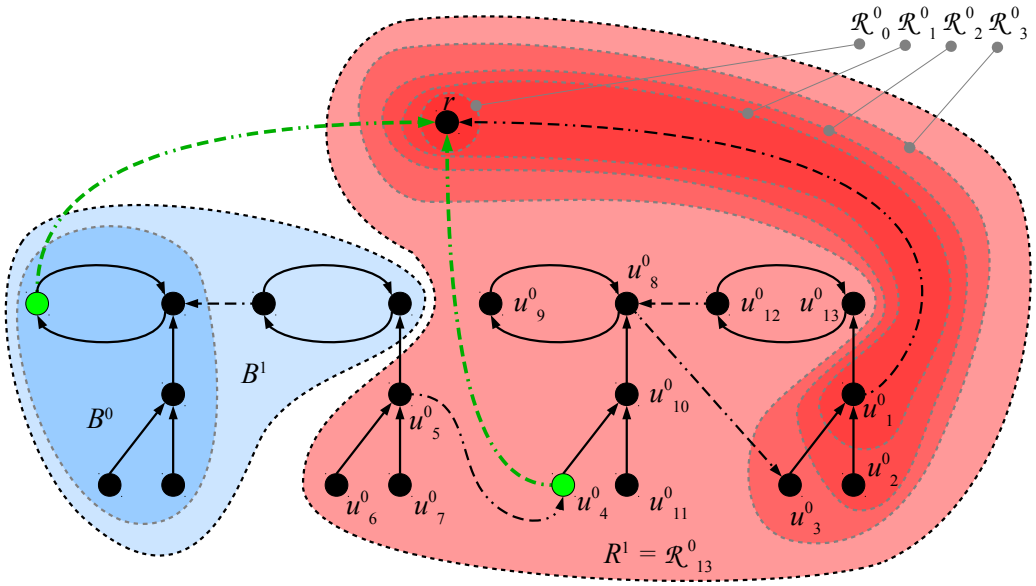


Figure 4: An example of a run of the inner iteration of Coordinate.

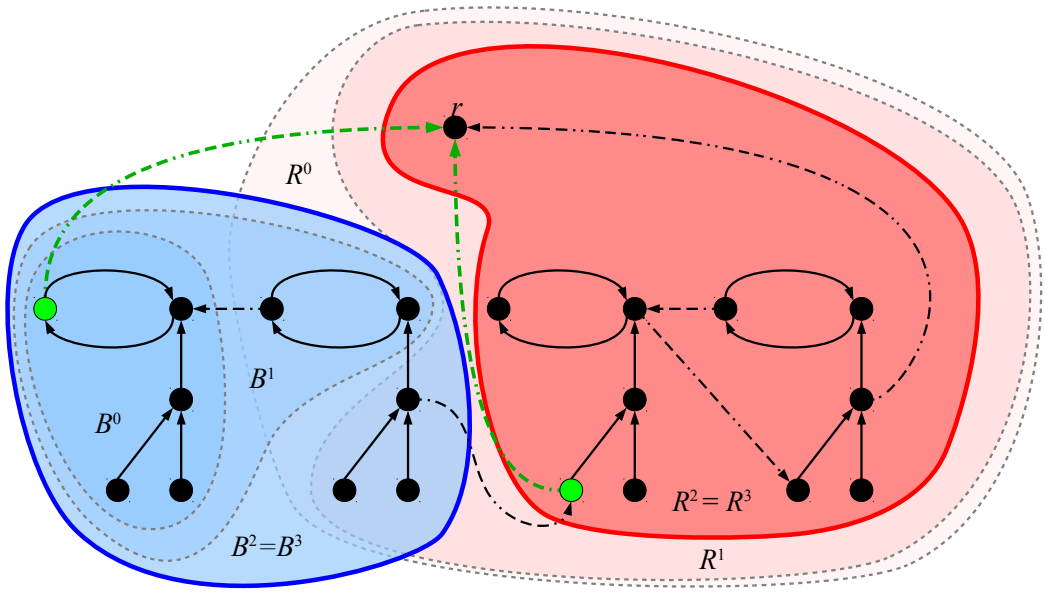


Figure 5: An example of a run of the outer iteration of Coordinate.

*Proof.* Observe that the procedure maintains the invariant $\mathcal{R}^q_\ell \cup U^q_\ell = V - B^q$ and maintains $\mathcal{R}^q_\ell \cap U^q_\ell = \emptyset$, for all $q, \ell \geq 0$. Therefore, any node must be exclusively in $B^q$ or $\mathcal{R}^q_\ell$ or $U^q_\ell$. □

**Fact 7.** *$(R^q, B^q)$ is a partition of $V$, for all $q = 0, 1, \ldots, q_*$.*

*Proof.* We have that $(B^0, R^0)$ is a partition of $V$ by definition. Now, from Line 11, we have $B^{q+1} = B^q \cup U^q_{\ell^q_*}$ and $R^{q+1} = \mathcal{R}^q_{\ell^q_*}$. Thus, $(B^{q+1}, R^{q+1})$ is a partition of $V$ by Fact 6. □

**Fact 8.** $B^0 \subset B^1 \subset \ldots \subset B^{q_*-1} = B^{q_*}$.

*Proof.* Since $B^{q+1} = B^q \cup U^q_{\ell^q_*}$, for $q = 0, 1, \ldots, q_* - 1$, we have by construction that $B^q \subseteq B^{q+1}$. The terminating condition of the outer iteration is $B^q = B^{q+1}$. This implies that these containments are strict for $q < q_* - 1$ and tight for $q = q_* - 1$. □

**Fact 9.** $R^0 \supset R^1 \supset \ldots \supset R^{q_*-1} = R^{q_*}$.

*Proof.* This is an immediate consequence of Fact 7 and Fact 8. □

**Fact 10.** *The graph $H^q = (R^{q+1}, E')$, where $E' = \{(u^q_\ell, \tau^q_\ell) : 0 \le \ell \le \ell^q_*\}$, is an arborescence rooted at $r$.*

*Proof.* Consider the outer iteration $q$ and the node $u^q_\ell \in R^{q+1} - \{r\}$ (where $0 \le \ell \le \ell^q_*$). Then $u^q_\ell \in R^{q+1} - \{r\}$ was added to $\mathcal{R}^q_{\ell+1}$ during the inner iteration $\ell$ because it preferred a node $\tau^q_\ell$ in $\mathcal{R}^q_\ell = \{r, u^q_0, \ldots, u^q_{\ell-1}\}$ to every node in $B^q \cup (U^q_\ell \cap \mathcal{K}) \supseteq B^q$. Thus, every node in $H^q - \{r\}$ has out-degree one and is connected by a directed path to the sink $r$. Therefore, $H^q$ is an arborescence on the node set $R^{q+1} = \mathcal{R}^q_{\ell^q_*}$. □

**Fact 11.** *$H^{q_*-1}$ is an arborescence rooted at $r$ and $u^{q_*-1}_\ell$ prefers $\tau^{q_*-1}_\ell$ to any node in $B$.*

*Proof.* That $H^{q_*-1}$ is an arborescence follows from Fact 10 applied to the final outer iteration $q_* - 1$. By construction, $u^{q_*-1}_\ell$ prefers $\tau^{q_*-1}_\ell$ to any node in $B^{q_*-1} \cup (U^{q_*-1}_\ell \cap \mathcal{K}) \supseteq B^{q_*-1} = B$. □

### 4.2.3 Structural Lemmas.

With these facts in hand, we prove the following three structural lemmas.

**Lemma 12** (Monotonicity Lemma)**.** *Let $(R, B)$ and $(\widetilde{R}, \widetilde{B})$ be generated by Coordinate($\mathcal{K}$) and Coordinate($\widetilde{\mathcal{K}}$), respectively. If $\widetilde{\mathcal{K}} \subseteq \mathcal{K}$, then $\widetilde{B} \subseteq B$.*

*Proof.* We show, by induction, that $\widetilde{B}^q \subseteq B^q$ for all $q \ge 0$. For the base case, we have

$$\widetilde{B}^0 := \bigcup_{i \ge 1 : V(C_i) \cap \widetilde{\mathcal{K}} \neq \emptyset} V(F_i) \subseteq \bigcup_{i \ge 1 : V(C_i) \cap \mathcal{K} \neq \emptyset} V(F_i) = B^0$$

Now assume $\widetilde{B}^q \subseteq B^q$ and consider the $q$th outer iteration of Coordinate($\mathcal{K}$). By Fact 10, $H^q$ is an arborescence on $R^{q+1}$ and each node $u^q_\ell$ prefers $\tau^q_\ell$ to every node in $B^q \cup (U^q_\ell \cap \mathcal{K})$. Let $u^q_\ell$ be the first node we add to $\mathcal{R}^q_\ell$ but not to $\widetilde{\mathcal{R}}^q_\ell$. Thus, at this iteration, we have $\mathcal{R}^q_\ell = \widetilde{\mathcal{R}}^q_\ell$. Now observe that

$$\begin{aligned}
B^q \cup (U^q_\ell \cap \mathcal{K}) &= (B^q \cup U^q_\ell) \cap (B^q \cup \mathcal{K}) \\
&= (V - \mathcal{R}^q_\ell) \cap (B^q \cup \mathcal{K}) \\
&\supseteq (V - \widetilde{\mathcal{R}}^q_\ell) \cap (\widetilde{B}^q \cup \widetilde{\mathcal{K}}) \\
&= (\widetilde{B}^q \cup \widetilde{U}^q_\ell) \cap (\widetilde{B}^q \cup \widetilde{\mathcal{K}}) \\
&= \widetilde{B}^q \cup (\widetilde{U}^q_\ell \cap \widetilde{\mathcal{K}})
\end{aligned}$$

Here the containment follows by induction, the assumption that $\widetilde{\mathcal{K}} \subseteq \mathcal{K}$, and the fact that $\mathcal{R}_\ell^q = \widetilde{\mathcal{R}}_\ell^q$. But, this implies that $u_\ell^q$ can be added to $\widetilde{\mathcal{R}}_k^q$, for any $k \geq \ell$. Thus, $u_\ell^q \in \widetilde{R}^{q+1}$ and so $R^{q+1} \subseteq \widetilde{R}^{q+1}$. Therefore, by Fact 7, we have that $\widetilde{B}^{q+1} \subseteq B^{q+1}$. □

**Lemma 13** (Coordination Lemma)**.** *The partition $(R, B)$ generated by Coordinate($\mathcal{K}$) is a coordinated colouring.*

*Proof.* We know $(R, B)$ is a partition by Fact 7. Thus, it suffices to show that each $F_i$ is either all red or all blue. If not, there is an $F_i$ containing both red and blue nodes. Then there are two alternatives.

(i) There is a red node $v \in F_i$ whose first choice is a blue node $w$. Observe that $w \in B^{q_*-1}$ because of the terminating condition $B^q = B^{q+1}$. But then, at the inner iteration $q_*$, we are not allowed to add $v$ to $\mathcal{R}_\ell^{q_*}$, a contradiction.

(ii) There is a blue node $v \in F_i$ whose first choice is a red node $w$. We may assume that $v \in B - B^0$ because $B^0$ consists only of first-class components that are monochromatic blue. So, $v$ must have been added to $B^{q+1}$ in some outer iteration $q \geq 0$. By Fact 9, $R \subseteq R^{q+1}$ and therefore $w \in R^{q+1}$. Thus, in the outer iteration $q$, there is an inner iteration $\ell$ in which $w = u_\ell^q$ is added to $\mathcal{R}_\ell^q$. But then as $v$ has its first choice in $\mathcal{R}_\ell^q$, it too will be added to $R^{q+1}$ rather than $B^{q+1}$, a contradiction. □

**Lemma 14** (Regeneration Lemma)**.** *If $(R, B)$ is generated by Coordinate($\mathcal{K}$), then it is also generated by Coordinate($B$).*

*Proof.* When we run Coordinate($\mathcal{K}$), we terminate with $B^{q_*-1} = B^{q_*} = B$. By Lemma 13, we know that $(R, B)$ is a coordinated colouring. Thus, $B$ consists of a collection of first-class components. So, if we run Coordinate($\widetilde{\mathcal{K}}$) with $\widetilde{\mathcal{K}} = B$, then we initiate

$$\widetilde{B}^0 := \bigcup_{i \geq 1 : V(C_i) \cap \widetilde{\mathcal{K}} \neq \emptyset} V(F_i) = \bigcup_{i \geq 1 : V(C_i) \cap B \neq \emptyset} V(F_i) = B$$

We now claim that, in the first outer iteration of Coordinate($\widetilde{\mathcal{K}} = B$), every node of $V - B$ will be added to $\widetilde{R}^1$. Thus, $\widetilde{B}^1 = \widetilde{B}^0$ and the procedure will terminate immediately with the partition $(R, B)$.

Consider the set $\widetilde{R}_\ell^1$ for some inner iteration $\ell$. We know that $\widetilde{U}_\ell^1 = V - \widetilde{R}_\ell^1 \subseteq R$ because $\widetilde{B}^0 = B$. If $\widetilde{U}_\ell^1 = \emptyset$, then we are done as $\widetilde{B}^1 = \widetilde{B}^0 = B$. So, we assume that $\widetilde{U}_\ell^1 \neq \emptyset$. By Fact 10, Coordinate($\mathcal{K}$) terminated with the property that the arcs $(u_\ell^{q*-1}, \tau_\ell^{q*-1})$ form an arborescence on $R$ rooted at the sink. Note that $r \notin \widetilde{U}_\ell^1$. So, $\widetilde{U}_\ell^1 \subset R$. Consequently, there must be one node $u_\ell^1 \in \widetilde{U}_\ell^1$ that has a neighbour $\tau_\ell^1 \in \widetilde{R}_\ell^1$ that $u_\ell^1$ prefers to every node in $\widetilde{B}^0$ (as $\widetilde{B}^0 = B$). This means that every node in $V - B$ is added to $\widetilde{R}^1$ as the inner iteration will not stop until $\widetilde{U}_\ell^1$ is empty. Therefore, Coordinate($B$) generates $(R, B)$ as required. □

#### 4.2.4 Constructing an Activation Sequence.

We now construct an activation sequence $\pi$ that will produce the coordinated partition $(R, B)$. The sequence will be defined by breaking the nodes into three groups.

The first group of nodes in the activation sequence $\pi$ will be the nodes in $B^0$. Recall $B^0 = \bigcup_{i \geq 1 : C_i \cap \mathcal{K} \neq \emptyset} V(F_i)$, the components $F_i$ whose first class cycles contain at least one clear node.

Within this group we order the nodes as follows. For each $F_i$ with $V(F_i) \subseteq B^0$, take a clear node $w_i \in C_i \cap \mathcal{K}$. Then activate the nodes of $F_i$ (except $w_i$) in increasing order of distance from $w_i$ in $F_i$ and, after that, activate $w_i$.

The second group of nodes in the activation sequence $\pi$ will be the nodes in $B - B^0$. We order the nodes of $B - B^0$ in a greedy manner. Specifically, suppose we have a partial activation ordering where $X$ is the subset of nodes in $B - B^0$ that have not yet been activated. Then the next node in the ordering is a node $x \in X$ which has a neighbour $\mu_x \in (B - X) \cup (\mathcal{K} \cap X)$ that $x$ prefers to every node in $R$; we prove in Lemma 15 below that such a node $x$ always exists.

Finally, the third group of nodes in the activation sequence $\pi$ will be the nodes in $R$. We activate the nodes in $R$ in the same order as they were added to $R = R^{q_*-1}$ in the outer iteration $q_* - 1$ of Coordinate.

As stated, to show that this activation sequence $\pi$ is well-defined we require the following lemma.

**Lemma 15.** *For any non-empty subset $X \subseteq B - B^0$, there is a node $x \in X$ that prefers some node $\mu_x \in (B - X) \cup (X \cap \mathcal{K})$ to every node in $R = V - B$.*

*Proof.* Let $X$ be a subset of $B - B^0$. Consider an outer iteration $q$, where $q$ is the largest number such that $B^q \subseteq V - X$; such a $q$ exists because $B^0 \subseteq V - X$. Therefore, $X \cap (B^{q+1} - B^q) = X \cap U^q_{\ell^q_*}$ is non-empty. By the terminating condition, Line 5, of the inner iteration, for each node $u \in U^q_{\ell^q_*}$ either (a) $u$ has a neighbour $\mu_u$ in $B^q \cup (U^q_{\ell^q_*} \cap \mathcal{K})$ that it prefers to every node in $\mathcal{R}^q_{\ell^q_*} = R^{q+1}$, or (b) $u$ has no neighbour in $\mathcal{R}^q_{\ell^q_*} = R^{q+1}$.

Suppose there is a node $x \in X \cap U_{\ell^q_*}$ of the former type (a). So, $x$ prefers $\mu_x$ to every node in $\mathcal{R}^q_{\ell^q_*} \supseteq R$. There are two possibilities. First, assume $\mu_x \in B^q \subset B$. Then, since $B^q \subseteq V - X$, we have $\mu_x \in B - X \subseteq (B - X) \cup (X \cap \mathcal{K})$, as desired. Second, assume $\mu_x \in U^q_{\ell^q_*} \cap \mathcal{K}$. Now $U^q_{\ell^q_*} = B^{q+1} - B^q \subseteq B$ and thus $\mu_x \in B \cap \mathcal{K} \subseteq (B - X) \cup (X \cap \mathcal{K})$, as desired.

On the other hand, assume that every node $v \in X \cap U^q_{\ell^q_*}$ is of the latter type (b). Such a node $v$ has no neighbour in $R^q_{\ell^q_*} = R^{q+1}$. Since $r \in R$ and every node has a path to $r$ in the all-choice graph, every path $P$ from $v$ to $r$ must contain a node of $B^{q+1} - X$. Hence, there must exist a node $x \in X \cap U^q_{\ell^q_*}$ that has a neighbour $\mu_x \in B^{q+1} - X \subseteq B - X$. Since $x$ has no neighbour in $R^{q+1} \supseteq R$, it certainly prefers $\mu_x$ to all nodes in $R$. □

Now, as desired, we obtain a feasible activation sequence $\pi$ that produces the coordinated partition $(R = R^{q^*}, B = B^{q^*})$ from $\mathcal{K}$.

**Lemma 16.** *Given a partition $(R, B)$ from Coordinate($\mathcal{K}$), the activation sequence $\pi$ associated with $(R, B)$ induces a sink-component on $R$ and non-sink-components on $B$.*

*Proof.* First, let's verify that each node in $R$ ends up in the sink-component under the activation sequence $\pi$. This follows immediately from Fact 11 since $u_\ell^{q^*-1} \in R$ will choose to connect to $\tau_\ell^{q^*-1}$.

Next, let's show that the nodes in $B_0$ do not end up in the sink-component. So, take a first class component $F_i$ with $V(F_i) \subseteq B_0$. Recall we activate the nodes of $F_i - \{w_i\}$ in increasing distance from $w_i$, where $w_i \in C_i$ is initially clear. Thus, each node in $F_i - w_i$ will have its first-choice available for selection (clear) when it is activated. Thus, each node in $F_i - w_i$ will select its first choice. Then so will $w_i$, when it is activated last. Consequently, $F_i$ will be a subgraph of the non-sink-components of the routing graph under the activation sequence $\pi$.

Finally, consider the nodes in $B - B^0$. Upon their activation under $\pi$ all the nodes in $B^0$ are activated and are currently clear. Let $X \subseteq B - B^0$ be a set of non-activated nodes as in the construction of the permutation $\pi$. If $X = B - B^0$, then by Lemma 15, there is a node $x \in X$ whose most preferred valid choice is $\mu_x$ in $B - X = B^0$. This node $\mu_x$ is a clear node in a non-sink-component and so $x$ attaches to this non-sink-component. Inductively, assume that all nodes in $B - X$ are clear. Then, Lemma 15 implies that there is a node $x \in X$ that has its most preferred valid choice in $B - X$, and $x$ must attach to a non-sink component. □

### 4.3 The Routing Phase.

Now, we apply the coordination algorithm three times to route all the packets. For each round $t = 1, 2, 3$, we denote by $\mathcal{K}_t$ and $(R[t], B[t])$ the set of clear nodes at the beginning of round $t$ and the red-blue partition generated by calling Coordinate($\mathcal{K}_t$). Thus, if we consider only the generation of partitions, then our algorithm can be described as

$$(R[t], B[t]) := \text{Coordinate}(\mathcal{K}_t), \quad \text{for } t = 1, 2, 3$$

Before proceeding to the main theorem, we remark that any for a cycle $C$ to form in the routing graph at round $t+1$ it must contain at least one clear node from $\mathcal{K}_t$. In fact, we will now show that if $(\mathcal{K}, V - \mathcal{K})$ is a coordinated colouring then the only cycles that can be formed are the first-class cycles. This fact implies that if a packet gets stuck, then it must be stuck in a first class cycle – this, in turn, will allow us to (approximately) keep track of where any lost packet must be.

Formally, we have the following property of each permutation $\pi$ generated by our algorithm.

**Lemma 17.** *Suppose the routing graph is coordinated, i.e., the set of clear nodes $\mathcal{K}$ at the beginning of the round forms a coordinated colouring $(\mathcal{K}, V - \mathcal{K})$. Then, if $(R, B)$ is generated by Coordinate($\mathcal{K}$), any cycle in the routing graph $\mathcal{N}$ (formed by the corresponding activation sequence $\pi$) is a first-class cycle in $\mathcal{K}$.*

*Proof.* Since $(\mathcal{K}, V - \mathcal{K})$ is coordinated, we initialise $B^0 = \mathcal{K}$ in Coordinate($\mathcal{K}$). By the proof of Lemma 16, under the activation sequence $\pi$ we have that each $F_i$ in $B^0$ forms a component in the routing graph $\mathcal{N}$. Thus, (i) every first class cycle $C_i$ contained in $\mathcal{K}$ must appear in the routing graph $\mathcal{N}$, and (ii) any other cycle in $\mathcal{N}$, if one exists, must be disjoint from $B^0$. But, Lemma 16 also tells us the red nodes $R$ form the sink-component $\mathcal{N}$. Thus, (iii) any other cycle in $\mathcal{N}$, if one exists, must also be disjoint from $R$.

So, suppose that $B - B^0$ induces a cycle $C$ in $\mathcal{N}$. By Lemma 16, the nodes of $B - B^0$ are in non-sink components. Note that, because $(\mathcal{K}, V - \mathcal{K})$ is a coordinated colouring, all nodes in $B - B^0$ are opaque at the beginning of the round. Now let $v$ be the first node of $C$ that is activated by $\pi$. So, at the time $v$ is activated, all the nodes of $C$ are opaque and thus cannot be chosen by $v$, a contradiction. □

To show every packet routes, it suffices to consider only the red-blue partitions generated from the coordinate algorithm. We will now prove the main theorem of this section.

**Theorem 18.** *In three rounds, every packet routes.*

*Proof.* Let's carefully examine where the packets are after each of the three rounds.
• **Round 1:** $(R[1], B[1]) := \text{Coordinate}(\mathcal{K}_1)$.

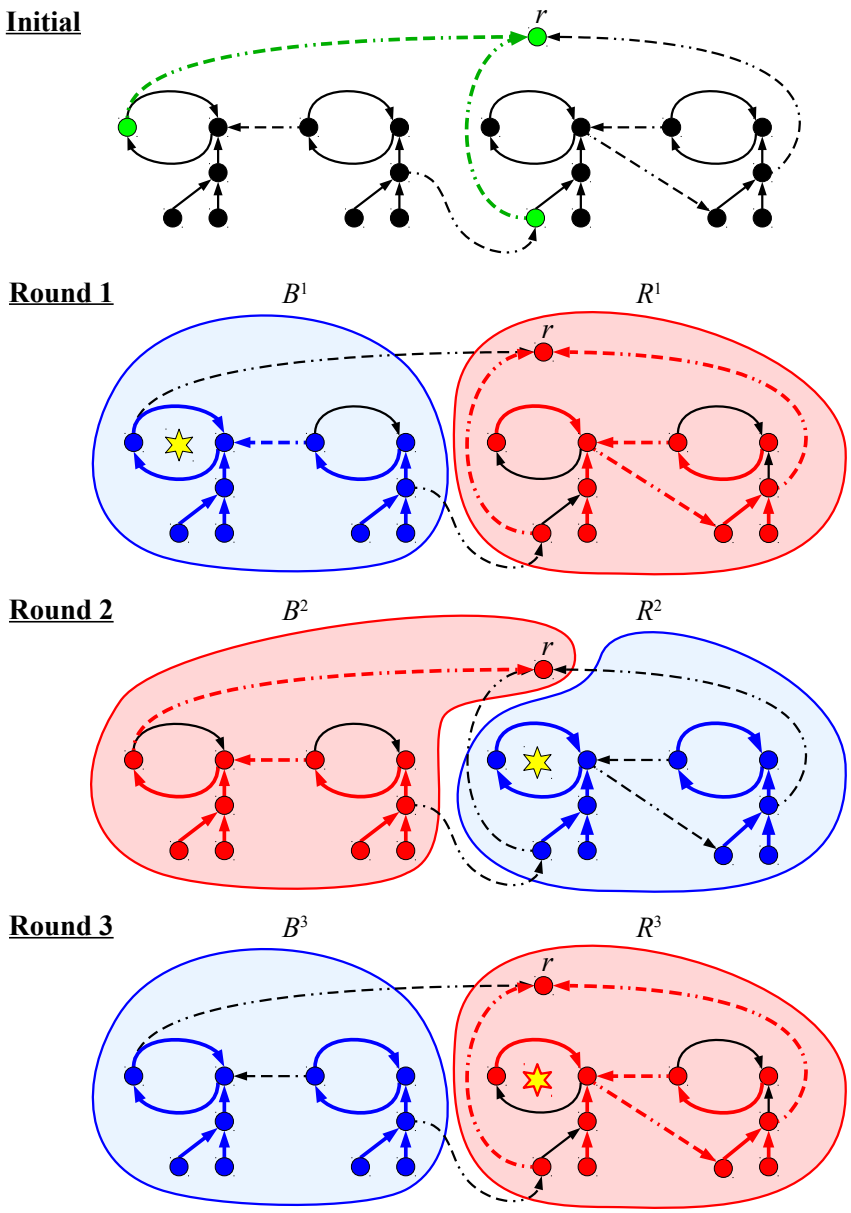


Figure 6: The illustration of the proof of Theorem 18. In the first two rounds, the packet ends up in a non-sink component. In the third round, it is in the sink-component.

The first round $t = 1$ is simply the coordination phase. Any packet that does not route is now stuck in $B[1]$ by Lemma 16.

- **Round 2:** $(R[2], B[2]) :=$ Coordinate$(R[1])$.

  By Lemma 16, $\mathcal{K}_2 = R[1]$. Now $(\mathcal{K}_2, V - \mathcal{K}_2) = (R[1], B[1])$ is coordinated by Lemma 13. Since $(\mathcal{K}_2, V - \mathcal{K}_2)$ forms a coordinated colouring, we initialise $B[2]^0 = \mathcal{K}^2 = R[1]$. Thus,

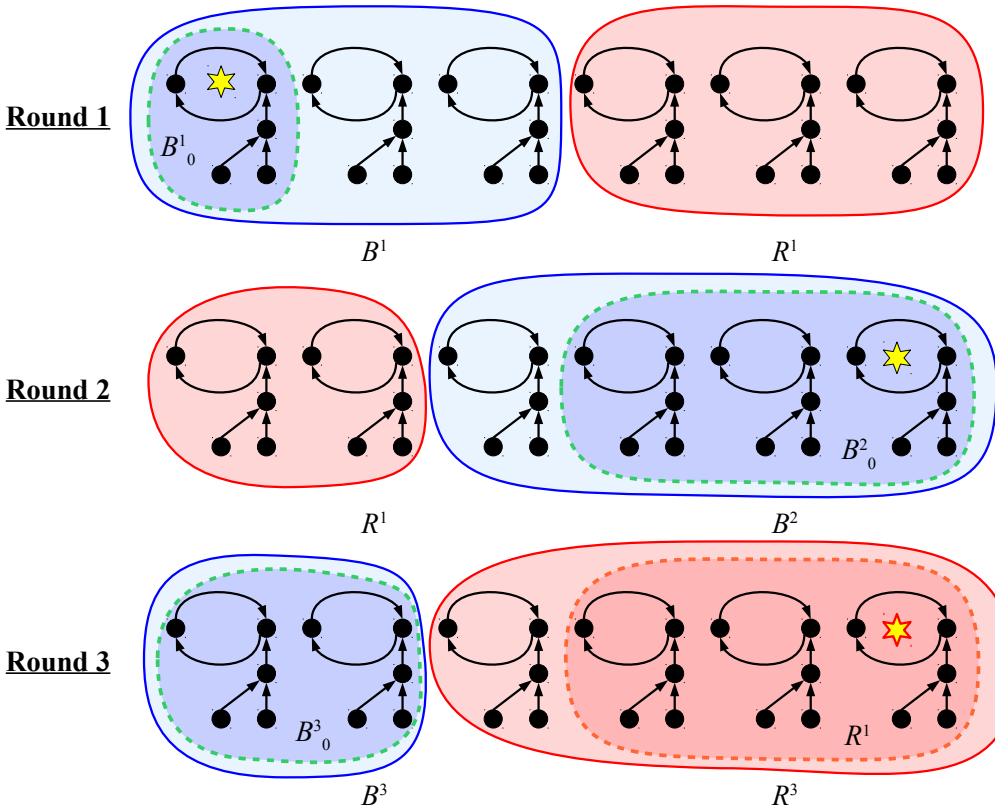


Figure 7:

because $B[2]^0 \subseteq B[2]$, we have $R[1] \subseteq B[2]$. Therefore, by Fact 7, $R[2] \subseteq B[1]$.

Next, since $(R[1], B[1])$ is coordinated, Lemma 17 implies that any packet that does not route is now stuck in $R[1]$.

- **Round 3:** $(R[3], B[3]) :=$ Coordinate$(R[2])$.

  By the Regeneration Lemma (Lemma 14), $(R[1], B[1])$ is generated both by Coordinate$(\mathcal{K}_1)$ and Coordinate$(B[1])$.

  By Lemma 16, $\mathcal{K}_3 = R[2]$. We have seen $R[2] \subseteq B[1]$. Therefore, by the Monotonicity Lemma (Lemma 12), Coordinate$(\mathcal{K}_3)$ generates a smaller (or equal) blue set than Coordinate$(B[1])$. The latter generates the blue set $B[1]$ itself. Thus, $B[3] \subseteq B[1]$ and, by Fact 7, $R[1] \subseteq R[3]$.

  Therefore, because every packet that does not route in Round 2 is in $R[1]$, any such packet is now in the sink-component $R[3]$ and is routed successfully.

This completes the proof. □

Figure 6 illustrates this proof for our running example. Note that, in general, we will have $R^1 \subset R^3$; an abstract illustration of that more standard type is shown in Figure 7.

# 5 Filtering: Not-Me!

So, even if no attempt to deter the formation of cycles is encoded into the preference lists every packet will route. In practice, the preference lists are designed to discourage cycles forming in the routing graph of a network. To achieve this, *loop-detection* is implemented in the BGP-4 protocol [17]. The "Not-Me!" filtering encodes loop-detection in the BGP-4 protocol simply by having a filtering list $\mathcal{D}(v) = \{v\}$, for every node $v$. For this class of preference function, we again show that every packet will route. Recall, this is in contrast to Theorem 5, which says that it is NP-hard to determine whether we can route every packet for general filtering lists of cardinality one.

Clearly, when filtering lists are non-empty, we have an additional difficulty: even if $w$ is the most preferred choice of $v$ and $w$ has a non-empty routing path $P$, $v$ still may not be able to choose $w$ because $P$ contains a node on $v$'s filter list (in this case, $v$ itself). This can cause the routing graph to evolve in ways that are very difficult to keep track of. Thus, the key idea is to design activation sequences that manipulate the routing graph in a precise and minor fashion in each round. To do this, we search for a spanning tree with a *Strong Stability Property*.

## 5.1 Strong Stability

Recall the definition of a stable spanning tree.

**Definition 19** (Stability)**.** *A (consistent) spanning tree $S$ is stable if and only if each node $v \in V$ prefers its parent $p(v)$ in $S$ to any non-descendant.*

Our goal here is to give a constructive way to obtain a stable spanning tree $S$ over $n$ rounds (interestingly, we will prove that all of the packets will actually route before stability is obtained). To show this, we will maintain a stronger stability property, and to define this strengthening we will need the following definitions. Given a graph $H$ and nodes $U \subseteq V$, let $H[U] = \{(u, v) : u, v \in U, (u, v) \in H\}$ be the subgraph of $H$ induced by $U$. We denote by $H^+[U] = \{(u, v) : u \in U, (u, v) \in R\}$, the subgraph of arcs of $H$ induced by $U$ plus all the arcs leaving $U$. Given a set of nodes $\Phi \subseteq V$, the $S[\Phi]$-*subtree* of $v$ is the maximal subtree rooted at $v$ of the forest $S[\Phi]$.

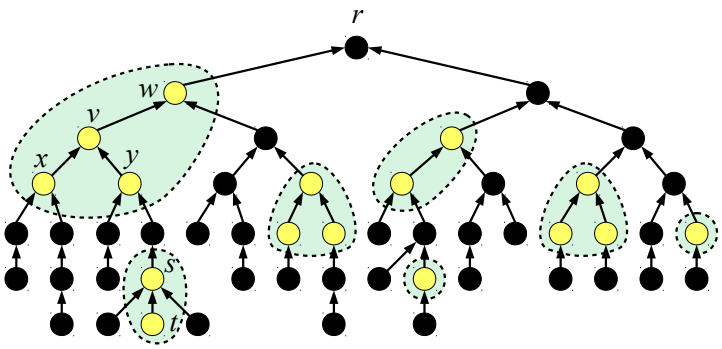


Figure 8: Maximal $S[\Phi]$-subtrees.

This concept is illustrated in Figure 8. There is a spanning tree $S$ and a subset of nodes $\Phi$ coloured in yellow. The dashed lines encompass maximal $S[\Phi]$-subtrees. An important observation

is that the roots of two disjoint maximal $S[\Phi]$-subtrees may form an ancestor-descendent pair. For example, for the maximal subtrees $\{w, v, x, y\}$ and $\{s, t\}$ we have that $w$ is an ancestor of $s$.

With this concept at hand, we may now define strong stability.

**Definition 20** (Strong Stability)**.** *A (consistent) spanning tree $S$ is strongly stable on $\Phi \subseteq V$ if and only if each node $v \in \Phi$ prefers its parent $p(v)$ in $S$ to every node outside the $S[\Phi]$-subtree rooted at $v$.*

To illustrate this concept, consider again Figure 8 and suppose that the spanning tree $S$ is strongly stable on $\Phi$. Then $w$ must prefer $r = p(w)$ to every node in $V - \{v, x, y\}$; similarly, $v$ must prefer $w = p(v)$ to every node in $V - \{x, y\}$.

Strong stability is indeed a strengthening of stability. To see this, note that if $S$ is strongly stable on $\Phi = V$ (or on $\Phi = V - \{r\}$), then $S$ is a stable tree. This observation leads to the basic approach used by our algorithm. A stable tree will route every packet and to find a stable tree it suffices to exhibit a set of fair activation sequences that allow for *strong stability* on a growing set of nodes $\Phi_1 \subset \Phi_2 \subset \cdots \subset \Phi_n = V$. (In fact, our analysis will actually show that every packet routes before stability is guaranteed.)

Before formally describing the algorithm, let's understand, at a high level, why strong stability is required during the intermediary stages rather than just the stability property on nodes in $\Phi_t$. The algorithm will target a specific stable spanning tree $S$ (it is easy to find such a tree in polynomial time). Specifically, it will find activation sequences $\pi_1, \pi_2, \ldots, \pi_n$ such that in round $t$ every node $v \in \Phi_t$ makes the same choice as in $S$. Furthermore, since we will have $\Phi_t \subset \Phi_{t+1}$, the choice of each node in $\Phi_t$ will become permanent after round $t$.

However, stability is insufficient to make this permanence property hold. The problem is that the stability property is with respect to the entire subtree rooted at a node $v$. Since such subtrees may contain nodes outside $\Phi_t$, we cannot ensure these subtrees grow with each round. Some descendant $w$ of $v$ in $S$ that lies outside $\Phi_t$ might become valid for $v$, and this then means that $v$ can now choose $w$ if $v$ prefers $w$ over its parent $p(v)$ in $S$. So, we lose the control over choices of nodes in $\Phi_t$. For a concrete example, consider the tree $S$ in Figure 9. If the tree is stable, then we

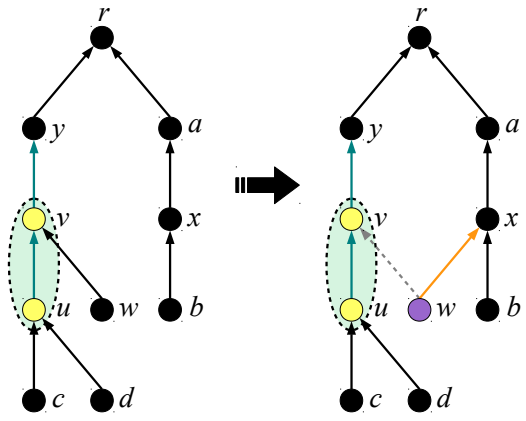


Figure 9: The figure illustrates a tree with strong stability on $\{v, u\}$. The node $v$ prefers $y$ to $a, b, c, d, w, x$ (but not $u$). So, even if $w$ changes its parent to $x$, $v$ still has $y$ as its best-choice.

know that $v$ prefers $y$ to $\{r, a, b, x\}$. Assume we are in round $t$, say, and consider the node $w \notin \Phi_t$. Since we only have stability on the nodes in $\Phi_t$, it is possible that $w$ will now change its parent from $v$ to $x$. But, if it does so, the subtree rooted at $v$ will then be reduced in size. Furthermore, $w$ is now a valid choice for $v$ as it lies outside of $v$'s subtree. Therefore, if $v$ prefers $w$ (or a descendent of $w$ if one exits) to $p(v) = y$, then the stability property for $v$ becomes violated. In contrast, suppose that $S$ is strongly stable on $\Phi_t$. Then the subtree of $\Phi_t$ rooted at $v$ is just $\{v, u\}$. Hence, we know that $v$ prefers $y$ to $\{r, a, b, c, d, w, x\}$. Thus, strong stability property for $v$ will hold even if $w$ changes its parent. Therefore, strong stability will ensure that the choices of nodes in $\Phi_t$ can be maintained and the maximal $S[\Phi_t]$-subtrees can only grow.

Since in each round we can make the strong stability property span at least one additional node, it follows that after $n$ rounds we have a stable tree and every packet will route. We remark that our procedure actually guarantees a stronger property: if in some round we cannot route all the packets, then the strong stability property spans at least two additional nodes. Thus, in $\frac{1}{2}n$ rounds, every packet will route, but we need $n$ rounds to obtain guarantee stability.

### 5.2 Finding a Strongly Stable Tree

We now formally implement the algorithm outlined above. Again, $\mathcal{O}_t$ and $\mathcal{K}_t$ denote the set of opaque and clear nodes, respectively, at the beginning of round $t$. It turns out to be fairly easy to obtain the strong stability property on the opaque nodes, that is, on the set $\mathcal{O}_t$. Hence, in each round, we grow $\Phi_t$ by ensuring the strong stability of $\mathcal{O}_t$ and adding $\mathcal{O}_t$ to $\Phi_{t+1}$. (In fact, we also add to $\Phi_{t+1}$ an additional node for which we can also guarantee the strong stability property.) The problem is that the resulting tree may *not* be a spanning tree. If it is spanning, then all the packets will route as this tree contains the sink.

Therefore, we need a method that allows us to expand a non-spanning tree to a spanning tree whilst maintaining strong stability. With this aim, we introduce a notion of a *skeleton*.

**Definition 21** (Skeleton)**.** *A spanning tree $S$ is a* skeleton *of a (non-spanning) tree $T$ on $\Phi$ if:*
- *$T \subseteq S$, and*
- *For every (maximal) subtree $F \subseteq S[\Phi]$, either $S^+[V(F)] \subseteq T$ or $V(T) \cap V(F) = \emptyset$.*

See Figure 10 for an example of a skeleton. We are now ready to present a subroutine for finding a spanning tree with the strong stability property on a set of nodes. Specifically, the input of this algorithm (see Procedure 4) is a sink-component $T_{in}$ and a spanning tree $S_{in}$ with the strong stability property on a given set of nodes $\Phi$. The algorithm expands the strong stability property to also hold on $\mathcal{O}$, the set of nodes not in the sink-component $T_{in}$.

Before proving the correctness of the procedure FindStable($T_{in}$, $S_{in}$, $\Phi$), we prove some basic facts.

**Lemma 22** (Union Lemma)**.** *Let $S$ be a spanning tree that is strongly stable on $A \subseteq V$ and also on $B \subseteq V$. Then $S$ is strongly stable on $Q = A \cup B$.*

*Proof.* Without loss of generality, take a node $v \in A \subseteq Q$. Let $F_A$ and $F_Q$ be the (maximal) $S[A]$-subtree and $S[Q]$-subtree of $v$, respectively. Then $V(F_A) \subseteq V(F_Q)$ because $A \subseteq Q$. By the strong stability property of $S$ on $A$, we have that $v$ prefers its parent $w$ in $S$ to any other node in $V - V(F_A)$. But, $V - V(F_Q) \subseteq V - V(F_A)$. It follows that $S$ is strongly stable on $Q = A \cup B$. □

The next lemma proves an important property of a skeleton of a tree $T$ on a set of nodes $\Phi$.

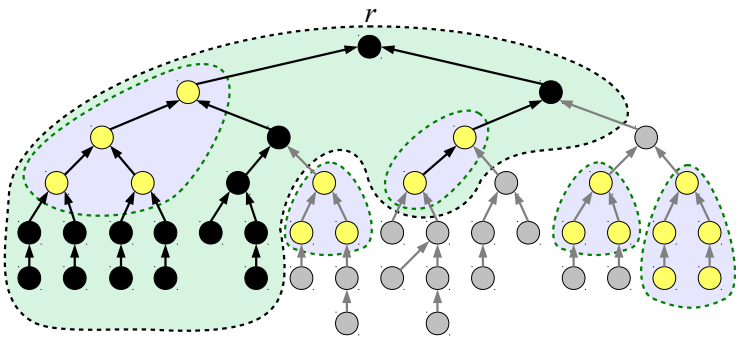


Figure 10: A skeleton $S$ of a (non-spanning tree) tree $T$ on $\Phi$. The black arcs denote arcs of the tree $T$ (which are also arcs of $S$); the grey arcs are arcs of $S$ that do not belong to $T$. The yellow nodes denote the set $\Phi$. Each subtree of $S[\Phi]$ is either contained in $T$ or is disjoint from $T$.

**Lemma 23** (Skeleton Lemma)**.** *Let $T$ be any tree. Let $S$ be a spanning tree that is strongly stable on a set of nodes $\Phi$ and is a skeleton of $T$ on $\Phi$. Let $\mathcal{O} = V - V(T)$. Then, for any spanning tree $T'$ such that $T \subseteq T'$, the tree $T'$ is strongly stable on $\Phi - \mathcal{O}$.*

*Proof.* Consider any node $v \in \Phi - \mathcal{O}$. Let $F_v$ be the $S[\Phi]$-subtree of $v$. By the definition of skeleton, for any (maximal) subtree $F \subseteq S[\Phi]$, either (i) $S^+[V(F)] \subseteq T$ or (ii) $V(F) \cap V(T) = \emptyset$. Since $v \in V(T) \cap \Phi$, it must be that $S^+[V(F_v)] \subseteq T \subseteq T'$. Thus, $V(F_v) \subseteq \Phi - \mathcal{O}$ as $V(T) = V - \mathcal{O}$. Therefore, $F_v$ is also the $S[\Phi - \mathcal{O}]$-subtree of $v$. By the strong stability property of $S$ on $\Phi$, we know that $v$ prefers its parent $w$ in $S$ to any node in $V - V(F_v)$. So, $S$ has the strong stability property on $\Phi - \mathcal{O}$. □

The next lemma shows the correctness of the procedure FindStable$(T_{in}, S_{in}, \Phi)$.

**Lemma 24.** *The procedure FindStable$(T_{in}, S_{in}, \Phi)$ outputs a spanning tree $S_{out}$ with the strong stability property on $\Phi \cup \mathcal{O}$ where $\mathcal{O} = V - V(T_{in})$.*

*Proof.* To begin, we show that $S_{out}$ is a spanning tree throughout the procedure. Initially $S_{out} = T_{in} \cup S_{in}^+[\mathcal{O}]$. Therefore, $S_{out}$ contains every node since $\mathcal{O} = V - V(T_{in})$. Let's see that $S_{out}$ is also connected. As $S_{in}$ is a spanning tree, we have that each component $F$ of $S_{in}^+[\mathcal{O}]$ is a tree (arborescence). Furthermore, as the sink $r$ is not in $\mathcal{O}$, there is exactly one arc in $S_{in}^+[\mathcal{O}]$ leaving $F$ (from its root) and entering $V(T_{in})$. Thus, $S_{out} = T_{in} \cup S_{in}^+[\mathcal{O}]$ (after Line 2 of FindStable) is a spanning tree.

Now consider how $S_{out}$ changes during the loop phase of the procedure. No node in $V - \mathcal{O}$ is considered during this phase, so $S_{out}$ never contains any arc leaving $V - \mathcal{O}$ and entering $\mathcal{O}$. As a result, the $S_{out}[\mathcal{O}]$-subtree of $v$ (chosen in Line 5 of FindStable) coincides exactly with the set of all descendants of $v$ in $S_{out}$. Hence, in Line 6 of FindStable, node $v$ never selects a descendant node to be $w$. So, we can safely replace the arc $(v, y) \in S_{out}$ by the arc $(v, w)$ without creating a cycle. This shows that $S_{out}$ is always a spanning tree.

Next, we show that $S_{out}$ has the strong stability property both on $\Phi - \mathcal{O}$ and on $\mathcal{O}$. By Lemma 22, this will imply that $S_{out}$ is strongly stable on $\Phi \cup \mathcal{O}$. Now, $S_{in}$ is strongly stable on $\Phi$ and is a skeleton of $T_{in}$ on $\Phi$. Furthermore, $T_{in} \subseteq S_{out}$ because, by construction, only outgoing

**Procedure 4** FindStable($T_{in}$, $S_{in}$, $\Phi$)

**Input:** A sink-component $T_{in}$ and a spanning tree $S_{in}$ such that
(1) The tree $S_{in}$ has the strong stability property on $\Phi$, and
(2) $S_{in}$ is a **skeleton** of $T_{in}$ on $\Phi$.

**Output:** A stable spanning tree $S_{out}$ with the strong stability property on $\Phi \cup \mathcal{O}$, where $\mathcal{O} = V - V(T_{in})$.

1: Let $\mathcal{O} = V - V(T_{in})$ be the set of nodes not in the sink component $T_{in}$.
2: Initialise $S_{out} := T_{in} \cup S^{+}_{in}[\mathcal{O}]$.
3: Initialise $\mathcal{C}_1 := S_{out}[\mathcal{O}]$.
4: **for** iteration $t := 1$ to $|\mathcal{O}|$ **do**
5: Pick an arbitrary leaf $v$ of $\mathcal{C}_t$.
6: Pick a node $w \in V(S_{out})$ such that (1) $w$ is not in a descendant of $v$ in $S_{out}$ and (2) $v$ prefers $w$ to any other node not its descendants in $S_{out}$.
7: Replace the arc $(v, y)$ in $S_{out}$ by the arc $(v, w)$.
8: Update $\mathcal{C}_{t+1} := \mathcal{C}_t - \{v\}$.
9: **end for**
10: **return** $S_{out}$.

arcs of $S_{out}[\mathcal{O}]$ are changed in the for-loop, and $\mathcal{O} = V - V(T_{in})$. Thus, applying Lemma 23, we have that $S_{out}$ is strongly stable on $\Phi - \mathcal{O}$. It only remains to show that $S_{out}$ is strongly stable on $\mathcal{O}$. To achieve this, we show by induction that $S_{out}$ is strongly stable on $\mathcal{L}_t = \mathcal{O} - V(C_t)$ in each iteration $t$. (Note that, on termination, $\mathcal{L}_t = \mathcal{O}$.) This is true for $t = 1$ as $\mathcal{L}_1 = \emptyset$. Now, consider iteration $t > 1$, and assume that strong stability holds on $\mathcal{L}_{t-1}$. Observe that no node $u \in \mathcal{O} - \mathcal{L}_{t-1}$ has a parent $x \in \mathcal{L}_{t-1}$; otherwise, $x$ would have not been added to $\mathcal{L}_{t-1}$. Since $v$ is a leaf of $C_t$, all the nodes in the $S_{out}[\mathcal{O}]$-subtree of $v$ must be in $\mathcal{L}_{t-1}$. Because nodes in $\mathcal{L}_{t-1}$ cannot change their parents after this time, every descendant of $v$ in $\mathcal{O}$ will remain a descendant of $v$. Consequently, $v$ prefers $w$ to any other non-descendant in $S_{out}$ throughout the rest of the procedure. Thus, $S_{out}$ is strongly stable on $\mathcal{L}_t$. □

### 5.3 Routing Every Packet in $n$ Rounds.

We are now ready to present an algorithm that routes every packet in $n$ rounds (recall that each round consists of a single fair-activation sequence). In addition to the procedure FindStable, two procedures (namely, Procedures 5 and 6) based upon a breath-first-search (BFS) algorithm are our basic building blocks for generating an activation sequence. Given a spanning tree $F$ and a set of nodes $U \subseteq V$, the procedure BFS($U, F$) activates the nodes of $U$ in *breadth-first-search (BFS) order*. That is, the nodes of $U$ are activated in increasing order of distance to the sink $r$ in $F$.

**Procedure 5** BFS($U, F$)

**Input:** A set $U \subseteq V$ and a spanning tree $F$.

1: Let $v_1, v_2, \ldots, v_q$ be nodes in $U - \{r\}$ sorted in increasing order of distance to the root $r$ of $F$.
2: **for** $i := 1$ to $q$ **do**
3: Activate $v_i$.
4: **end for**

Similarly, the procedure reverse-BFS$(U, F)$ activates the nodes of $U$ in breadth-first-search (BFS) reverse-order.

**Procedure 6** reverse-BFS$(U, F)$

**Input:** A set $U \subseteq V$ and a spanning tree $F$.

Let $v_1, v_2, \ldots, v_q$ be nodes in $U - \{r\}$ sorted in increasing order of distance to the root $r$ of $F$.
**for** $i := q$ to 1 **do**
    Activate $v_i$.
**end for**

Over the course of these $n$ rounds, the main algorithm (Procedure 7) utilises these three procedures on the following two classes of nodes.

(1) The set of nodes that have been clear in *every* round up to time $t$, denoted by $\Delta_t \subseteq \bigcap_{i=1}^{t} \mathcal{K}_i$.

(2) The complement of $\Delta_t$, which is the set of nodes that have been opaque at least once by time $t$ plus nodes on which the strong stability hold, denoted by $\Phi_t \supseteq \bigcup_{i=1}^{t} \mathcal{O}_i$.

We remark that, to be precise, in each round, we also move one node from $\Delta_t$ to $\Phi_t$, but this is only to make the routing graph become stable faster.

**Procedure 7** Fair-Stabilise()

Initialise $\widehat{S}_0 :=$ an arbitrary spanning tree, $\widehat{\Delta}_0 := V$ and $\widehat{\Phi}_0 := \emptyset$.
**for** round $t := 1$ to $n$ **do**
    Let $T_t$ be the sink-component at the beginning of round $t$.
    Let $\mathcal{O}_t = V - V(T_t)$ be the set of opaque nodes at the beginning of round $t$.
    Apply FindStable$(T_t, \widehat{S}_{t-1}, \widehat{\Phi}_{t-1})$ to compute a spanning tree $S_t$.
    Update $\Phi_t := \widehat{\Phi}_{t-1} \cup \mathcal{O}_t$ and $\Delta_t := V - \Phi_t$.
    Activate BFS$(\Phi_t, S_t)$.
    Denote the new routing graph by $\mathcal{N}_{\Phi_t}$.
    **if** $\Delta_t \neq \emptyset$ **then**
        Activate reverse-BFS$(\Delta_t, S_t)$.
        Denote the new routing graph by $\mathcal{N}_{\Delta_t}$.
        Pick a node $v_t$ that is the first node activated by reverse-BFS$(\Delta_t, S_t)$.
        Construct a graph $\widehat{S}_t$ from $S_t$ by replacing the arc $(v_t, p(v_t)$ in $S_t$ by $(v_t, w_t)$ the arc chosen by $v_t$ in the routing graph $\mathcal{N}_{\widehat{\Delta}_t}$.
        Update $\widehat{\Phi}_t := \Phi_t \cup \{v_t\}$ and $\widehat{\Delta}_t := \Delta_t - \{v_t\}$.
    **else**
        Exit the loop.
    **end if**
**end for**

Observe that Procedure 7 is clearly fair because $\Delta_t$ and $\Phi_t$ partition the set of nodes. The basic intuition behind the method is that if we can make our routing graph $T_{t+1}$ look like the spanning tree $S_t$, then every packet will route. Typically, any activation sequence that attempts to do this, though, will induce inconsistencies. This, in turn, will force nodes to go opaque. But, it turns out

that we can make those nodes in $\Phi_t$ choose arcs in accordance with $S_t$ via the use of BFS$(\Phi_t, S_t)$. (It is not at all obvious that this can be done because nodes in $\Phi_t$ may actually be clear, that is, they need not be in $\mathcal{O}_t$.)

Then the question becomes how do we keep track of the packets. The key point is that nodes in $\Phi_t$ choose arcs in accordance with the spanning tree $S_t$. Therefore, since $\Phi_1 \subset \Phi_2 \subset \Phi_3 \subset \cdots$ and the containments are strict, we will eventually have $\Phi_t = V$, and our routing graph will be a spanning tree. Thus, every packet routes! Moreover, the strong stability property of $S_t$ on $\Phi_t = V$ also implies that the final routing graph is a stable spanning tree.

The following lemmas present the key properties we need to prove all this.

**Lemma 25.** *In Fair-Stabilise, assuming that FindStable is called with a valid input, we have that $\mathcal{N}_{\Phi_t} = S_t$ and that every node is consistent in the network $\mathcal{N}_{\Phi_t}$.*

*Proof.* Line 5 of Fair-Stabilise calls the Procedure FindStable. Thus, we know that $S_t$ is a spanning tree that is strongly stable on $\Phi_t$ and that $S_t[V - \mathcal{O}_t] = T_t$.

Initially, the routing graph consists of the sink-component $T_t$ and a set $\mathcal{O}_t = V - V(T_t)$ of isolated nodes. To obtain $\mathcal{N}_{\Phi_t}$, we only activate the nodes in $\Phi_t$. It then suffices to show, by induction on the order of activation within BFS$(\Phi_t, S_t)$, that every node $v \in \Phi_t$ will (i) choose its parent $p(v)$ in $S_t$ as its next hop and (ii) become consistent (i.e., $v$ has a "real" path as its chosen route).

For the base case, consider the first node $v$ activated by BFS$(\Phi_t, S_t)$. We will show that $p(v)$ is the best valid choice of $x$.

First, suppose that $p(v) \in T_t$. Thus, $p(v)$ is clear and is available to be selected by $v$. By the strong stability property of $S_t$ on $\Phi_t$, we know that $v$ prefers $p(v)$ to any node outside the $S_t[\Phi_t]$-subtree of $v$, say $F_v$. Now, suppose that there is a node $w \in F_v$ that is a valid choice and is preferred by $v$ over $p(v)$. If $w$ is valid, then it is clear, that is, $w \in T_t$, and it does not contain $v$ in its chosen route $\mathcal{P}(w)$. If $w \in T_t$, then it is consistent, but then $v$ is an ancestor of $w$ in $T_t$. Thus, the consistency of $w$ implies that $\mathcal{P}(w)$ does contain $v$. So, $w$ is not a valid choice. Thus, $v$ does select $p(v)$. Moreover, after this selection, $v$ is also consistent because $T_t$ is consistent.

Second, suppose $p(v) \notin T_t$. Then $p_v \in \mathcal{O}_t$. But, $\mathcal{O}_t \subseteq \Phi_t$; this contradicts the fact that $v$ was the first node activated.

For the inductive step, assume that every node activated prior to $v \in \Phi_t$ chooses its parent in $S_t$ and becomes consistent. First, if $p(v) \in T_t$, then we apply the same argument as above. Second, suppose $p(v) \notin T_t$. Then $p(v) \in \mathcal{O}_t \subseteq \Phi_t$, so it has already been activated. Thus, $p(v)$ is now clear. Again, by the strong stability property of $S_t$ on $\Phi_t$, we know that $v$ prefers $p(v)$ to any node outside the $S_t[\Phi_t]$-subtree of $v$, say $F_v$. Because $p(v) \notin T_t$, we know that $v \notin T_t$. Therefore, no node of $F_v$ is in $T_t$. As a consequence, by the activation ordering, every node in $F_v$ is still opaque. Thus, $v$ does select $p(v)$ and becomes consistent. This proves that $\mathcal{N}_{\Phi_t} = S_t$ and that every node is consistent in $\mathcal{N}_{\Phi_t}$. □

**Lemma 26.** *At the end of round $t$ of Fair-Stabilise, we have*

*(a) The spanning tree $\widehat{S}_t$ is strongly stable on $\widehat{\Phi}_t$.*

*(b) $\widehat{S}_t^+[\widehat{\Phi}_t]$ is contained in the routing graph at the end of round.*

*Proof.* We prove the lemma by induction on $t$. The base case $t = 0$ is trivial because $\widehat{\Phi}_0 = \emptyset$. Now assume that the statements hold up to round $t - 1$ for some $t > 0$. Our proof is in two parts.

First, we prove (a) and (b) hold for the spanning tree $S_t$ with respect to $\Phi_t$. Second, we show both properties are maintained after $v_t$ changes its parent and is added to $\Phi_t$; that is, both (a) and (b) hold for $\widehat{S}_t$ with respect to $\widehat{\Phi}_t$.

• *The properties hold for $S_t$ with respect to $\Phi_t$.*

By Lemma 24, FindStable$(T_t, \widehat{S}_{t-1}, \widehat{\Phi}_{t-1})$ builds a spanning tree $S_t$ that is strongly stable on $\widehat{\Phi}_t$. Thus, Property (a) does hold *provided* the inputs to FindStable were valid. To be valid, we require that (i) $\widehat{S}_{t-1}$ is strongly stable on $\widehat{\Phi}_{t-1}$ and (ii) $\widehat{S}_{t-1}$ is a skeleton of $T_t$ on $\widehat{\Phi}_{t-1}$. The first fact (i) follows immediately from the induction hypothesis. The second fact (ii) also follows by induction as Property (b) is clearly a stronger property than the skeleton property. Thus, (a) holds.

Given the inputs were feasible, we obtain from Lemma 25 that $\mathcal{N}_{\Phi_t} = S_t$. After this, before the end of the round, the node activations are via reverse-BFS$(\Delta_t, S_t)$. Thus, the nodes in $\Phi_t = V - \Delta_t$ maintain their selection until the end of the round. So, $S_t^+[\Phi_t]$ is in the routing graph and (b) holds.

• *The properties hold for $\widehat{S}_t$ with respect to $\widehat{\Phi}_t$.*

To create $\widehat{S}_t$ from $S_t$, observe that we replace $(v_t, p(v_t))$ by $(v_t, w_t)$. By Lemma 25, the network $\mathcal{N}_{\Phi_t}$ was consistent. Thus, $w_t$ is not a descendant of $v_t$ in $S_t$ and, therefore, $\widehat{S}_t$ is indeed a spanning tree.

Now $\widehat{\Phi}_t = \Phi_t \cup \{v_t\}$. Since $v_t$ selects $w_t$, we have that $\widehat{S}_t^+[\widehat{\Phi}_t]$ is in the routing graph and (b) holds.

It remains to show (a). By the choice of $v_t$ (in Line 10) of Fair-Stabilise, we know that every descendant of $v_t$ in $\mathcal{N}_{\Phi_t}$ is in $\Phi_t$. Thus, strong stability for the node $v_t$ is achieved by the choice of $w_t$. So, let's verify that the strong stability condition holds for any other node $y$ in $\widehat{\Phi}_t$, that is, for each node in $\widehat{\Phi}_t - \{v_t\} = \Phi_t$. Now $v_t \in \Delta_t$ and, hence, it is not contained in the $\widehat{S}_t[\Phi_t]$-subtree for $y$. Thus, when we add $v_t$ to $\widehat{\Phi}_t$ the resulting $\widehat{S}_t[\widehat{\Phi}_t]$-subtree for $y$ cannot be smaller that the corresponding $\widehat{S}_t[\Phi_t]$-subtree. Thus, the strong stability is now easier to satisfy for $y$ and so it must still apply. Property (a) then holds. □

**Theorem 27.** *The algorithm produces a stable tree after $n$ rounds.* □

*Proof.* By the construction in Line 14 of Fair-Stabilise, we have $|\widehat{\Phi}_{t+1}| \geq |\widehat{\Phi}_t| + 1$, for $t \geq 0$. Lemma 26 then guarantees that the final routing network is stable. □

**Lemma 28.** *If at least one packet does not route in round $t$, then $|\widehat{\Phi}_{t+1}| \geq |\widehat{\Phi}_t| + 2$.*

*Proof.* If $\widehat{\Phi}_t = V - \{r\}$, then by Lemma 26, we have $\widehat{S}_t^+[\widehat{\Phi}_t] = \widehat{S}_t$. Thus, Lemma 26 implies that $\mathcal{N}_{\Delta_t} = \widehat{S}_t$ is a stable spanning tree and every packets will route. Consequently, if at least one packet does not route, then $\widehat{\Phi}_t \subset V - \{r\}$.

We want to show that in round $t+1$ at least one node $y$ is added to $\widehat{\Phi}_{t+1}$ in addition to $v_{t+1}$. It suffices to show that there is a node $y \in \mathcal{O}_{t+1} - \widehat{\Phi}_t$. Now $\mathcal{O}_{t+1}$ is non-empty, otherwise every packet routed. Thus, the routing graph $\mathcal{N}_{\Delta_t}$ (at the end of round $t$, before the learning phase) contains a cycle $C$. By Lemma 26, $\widehat{S}_t^+[\widehat{\Phi}_t]$ is contained in $\mathcal{N}_{\Delta_t}$. The subgraph $\mathcal{N}_{\Delta_t}^+[\widehat{\Phi}_t]$ is thus a forest because $\mathcal{N}_{\Delta_t}^+[\widehat{\Phi}_t] = \widehat{S}_t^+[\widehat{\Phi}_t]$ and $\widehat{S}_t$ is a tree. So, at least one node $y$ of $C$ is not in $\widehat{\Phi}_t$. □

It is immediate from Lemma 28 that we can route every packet in $\lfloor n/2 \rfloor$ rounds, and that the network becomes stable in $n$ rounds. Moreover, we can deduce a stronger failure guarantee. We say that round $t$ is an *imperfect round* if we cannot route every packet. Then there can be at most $\lfloor n/2 \rfloor$ imperfect rounds (note that these may not be consecutive rounds) even if the routing graph is not yet stable.

**Theorem 29.** *There is an activation sequence that routes every packet in $\lfloor n/2 \rfloor$ rounds, gives a stable spanning tree in n rounds, and guarantees that there are at most $\lfloor n/2 \rfloor$ imperfect rounds.*

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

## References

[1] Thomas Erlebach, Alexander Hall, Alessandro Panconesi, and Danica Vukadinovic. Cuts and disjoint paths in the valley-free model. *Internet Mathematics*, 3(3):333–359, 2007. 31

[2] Alex Fabrikant and Christos H. Papadimitriou. The complexity of game dynamics: BGP oscillations, sink equilibria, and beyond. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 844–853, 2008. 6

[3] Alex Fabrikant, Umar Syed, and Jennifer Rexford. There's something about MRAI: Timing diversity can exponentially worsen BGP convergence. In *Proceedings of the 30th IEEE International Conference on Computer Communications (INFOCOM)*, pages 2975–2983, 2011. 30

[4] Lixin Gao and Jennifer Rexford. Stable internet routing without global coordination. *IEEE/ACM Transactions on Networking*, 9(6):681–692, 2001. 30

[5] Timothy Griffin, F. Bruce Shepherd, and Gordon T. Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Transactions on Networking*, 10(2):232–243, 2002. 1, 6, 29, 30

[6] Timothy Griffin and Gordon T. Wilfong. An analysis of BGP convergence properties. In *Proceedings of the ACM SIGCOMM 1999 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 277–288, 1999. 6, 29

[7] Timothy G. Griffin and Brian J. Premore. An experimental analysis of BGP convergence time. In *Proceedings of the 9th International Conference on Network Protocols (ICNP)*, pages 53–61, 2001. 30

[8] John P. John, Ethan Katz-Bassett, Arvind Krishnamurthy, Thomas E. Anderson, and Arun Venkataramani. Consensus routing: The internet as a distributed system. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design & Implementation (NSDI)*, pages 351–364, 2008. 31

[9] Howard J. Karloff. On the convergence time of a path-vector protocol. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 605–614, 2004. 30

[10] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972. 7

[11] Nate Kushman, Srikanth Kandula, Dina Katabi, and Bruce M. Maggs. R-BGP: Staying connected in a connected world. In *Proceedings of the 4th Symposium on Networked Systems Design and Implementation (NSDI)*, 2007. 30

[12] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. Delayed internet routing convergence. *IEEE/ACM Transactions on Networking*, 9(3):293–306, 2001. 30

[13] Hagay Levin, Michael Schapira, and Aviv Zohar. Interdomain routing and games. *SIAM Journal on Computing*, 40(6):1892–1912, 2011. 31

[14] Zhuoqing Morley Mao, Jennifer Rexford, Jia Wang, and Randy H. Katz. Towards an accurate AS-level traceroute tool. In *Proceedings of the ACM SIGCOMM 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 365–378, 2003. 31

[15] Dan Pei, Matt Azuma, Daniel Massey, and Lixia Zhang. BGP-RCN: improving BGP convergence through root cause notification. *Computer Networks*, 48(2):175–194, 2005. 30

[16] Michael Schapira, Yaping Zhu, and Jennifer Rexford. Putting BGP on the right path: a case for next-hop routing. In *Proceedings of the 9th ACM Workshop on Hot Topics in Networks (HotNets)*, page 3, 2010. 30, 31

[17] John W. Stewart, III. *BGP4: Inter-Domain Routing in the Internet*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998. 20, 29

[18] Kannan Varadhan, Ramesh Govindan, and Deborah Estrin. Persistent route oscillations in inter-domain routing. *Computer Networks*, 32(1):1–16, 2000. 29

[19] Stefano Vissicchio, Luca Cittadini, Laurent Vanbever, and Olivier Bonaventure. iBGP deceptions: More sessions, fewer routes. In *Proceedings of the 31st IEEE International Conference on Computer Communications (INFOCOM)*, pages 2122–2130, 2012. 2

## Appendix: Interdomain Routing and Model Technicalities.

The Internet is a union of subnetworks called *domains* or Autonomous Systems (ASes). The interdomain routing protocol used in the Internet today is called the Border Gateway Protocol (BGP), and it works as follows [17]. For destination $r$ and router $v$, each neighbouring router of $v$ announces to $v$ the route to $r$ that it has chosen and from amongst these announced routes, $v$ chooses the route $\mathcal{P}(v)$ that it ranks highest. The router $v$ then in turn announces to its neighbouring routers its routing path $\mathcal{P}(v)$. This process continues until an *equilibrium* is reached in which each router has chosen a route and for each router $v$, no neighbour of $v$ announces a route that $v$ would rank higher than its current routing path. The ranking of routes at a router depends on a number of route attributes such as which neighbour announced the route, how long the route is, and which domains the route traverses. In fact, the ranking of routes at $v$ is a function of $v$'s traffic engineering goals as well as the Service Level Agreements (SLAs), that is, the economic contracts $v$ has made with its neighbours.

It is well known that BGP can be thought of as a game [5] and that BGP as a game may have no Nash equilibrium [18, 6]. There is now a vast literature studying the conditions under

which BGP will or will not have an equilibrium (for example [5, 4]). It has been shown that in a BGP instance, the absence of a structure known as a *dispute wheel* implies that the BGP instance will have a unique equilibrium [5]. There have been a number of papers analysing the worst-case convergence time of BGP instances having no dispute wheel [9, 3, 16]. There have also been many experimental papers measuring BGP convergence times [12, 7] and papers offering modifications to BGP with the goal of speeding up convergence [15, 11].

However, BGP convergence is only a step towards the ultimate goal of successfully delivering packets to the destination. In fact, routers perform operations simultaneously on two basic levels: (1) on the *control plane* (i.e., where BGP exchanges routing information with other routers as described above) and (2) on the *forwarding plane* where routers use the routing information from BGP to forward packets to neighbouring routers towards the packets' ultimate destinations. That is, packets are being forwarded during the time that the control plane is attempting to settle on an equilibrium.

Recall our model in Section 2. We base our idealised routing protocols on BGP and two particularly important attributes that a router uses to rank its available routes. Firstly, a router might not trust certain domains to handle its packets securely or in a timely fashion, so it may reject routes traversing such unreliable domains. This motivates a *(no-go) filtering*, which will filter out any route that goes through an undesirable domain (i.e., a domain on the router's no-go filtering list). Secondly, it has been argued that perhaps the most important attribute in how a router $v$ ranks routes is the neighbour of $v$ announcing the route to $v$ [16]. That is, one can think of each router ordering its neighbours and ranking any route from a lower ordered router over any route from a higher ordered router. This is called *next-hop routing*. Thus, in our protocols, a node ranks routes by first filtering out any route that goes through nodes on its filtering list and then choosing from amongst the remaining routes the one announced by the lowest ordered neighbour (*next-hop preference with filtering*).

As discussed, to analyse stability, it suffices to consider only the control plane. But, to understand packet routing, we need to understand the interaction between the forwarding and control planes. Thus, we need to incorporate the actions of the forwarding plane into the standard model of the control plane [5]. To do so, some assumptions must be made, particularly concerning the synchronisation between the planes. In setting up a model for a practical problem, it is important to examine how the modelling assumptions relate to reality. So, here we briefly address some technical aspects:

- **Synchronisation of the Planes.** Observe that, in our model, the control plane and the forwarding plane operate at a similar speed. Given fair activation sequences (see below), this assumption is the worst case in that it maximises the rate at which inconsistencies are produced between the nodes routing paths. In practice, updates in the control plane are much slower than the rate of packet transfer.

- **Packet Cycling.** When a packet gets stuck in a cycle, we will assume that, at the start of the next round, an adversary can position the packet at whichever node in the cycle they wish.

- **Fair Activation Sequences.** We insist that activation sequences in the control plane are *fair* in that all nodes update their routes at a similar rate. Clearly, the use of permutations ensures fairness. From the theoretical point of view, fairness is important as it avoids artificially

routing packets by the use of unnatural and pathological activation sequences. For example, it prohibits the use of activation sequences that are biased towards nodes in regions where disconnectivities arise and attempts to fix this by "freezing" other nodes until consistency is obtained. Moreover, in practice, routers timings in the control plane are similar.

- **Routing in Rounds.** The use of rounds (defined by permutations) for routing is not vital and is used for clarity of exposition and to emphasise fairness. Also, packet forwarding is clearly not delayed until the end of a "round" in practice but, again, this is also not needed for the model. The assumption is made as it clarifies the arguments needed in the analyses. For example, forwarding at the end of a round can be shown to be equivalent to forwarding continuously throughout the round with the planes in sync; that is, packets are forwarded immediately and, within a round, the routing path at a node is updated just before the first packet a node sees is about to leave it.

- **Route-Verification.** Route-verification at the end of the round is our one non-worst case assumption and is not a standard aspect of BGP, albeit one that can be incorporated in a fairly simple fashion by tools such as traceroute or an AS-level traceroute tool such as that described by Mao et al. [14]. Route-verification is the focus of the influential paper of John et al. [8] on consensus routing. It is also used in the theory literature on incentives under BGP [13]. Due to the manipulative power provided by unfair activation sequences, it is not hard to simplify our algorithms and omit the route-verification step given the use of unfair activation sequences; see also [16]. It remains an interesting open problem to obtain consistency using fair sequences without route-verification.

- **Filtering.** In this paper, we assume that each node can apply what is known as *import filtering* – that is, not accepting certain routes from its neighbours. This implicitly assumes that each node announces its routing path to all of its neighbours. In reality, each node may choose to apply *export filtering* – that is, it may announce any particular route to only a subset of its neighbours (e.g., in order to assure "valley-free routing" [1]).

  Export filtering can be incorporated into our model by allowing for neighbour specific import filtering rules, where a node $v$ can have a filtering list $\mathcal{D}(v, w)$ for each neighbour $w$. Of course, our lower bounds would still hold for this more general model, but it would allow for more special cases to explore.