BGP Safety with Spurious Updates

Martin Suchara, Alex Fabrikant, and Jennifer Rexford
Computer Science Department, Princeton University
Email: {msuchara, afabrika, jrex}@cs.princeton.edu

***Abstract*—We explore BGP safety, the question of whether a BGP system converges to a stable routing, in light of several BGP implementation features that have not been fully included in the previous theoretical analyses. We show that Route Flap Damping, MRAI timers, and other intra-router features can cause a router to briefly send "spurious" announcements of less-preferred routes. We demonstrate that, even in simple configurations, this short-term spurious behavior may cause long-term divergence in global routing. We then present DPVP, a general model that unifies these sources of spurious announcements in order to examine their impact on BGP safety. In this new, more robust model of BGP behavior, we derive a necessary and sufficient condition for safety, which furthermore admits an efficient algorithm for checking BGP safety in most practical circumstances — two complementary results that have been elusive in the past decade's worth of classical studies of BGP convergence in more simple models. We also consider the implications of spurious updates for well-known results on dispute wheels and safety under filtering.**

I. Introduction

The Border Gateway Protocol (BGP) [1], the de facto interdomain routing protocol in the Internet, offers autonomous systems (ASes) the flexibility to specify their custom routing policies. Unfortunately, this flexibility may result in policy choices that cause persistent oscillations. Such oscillations unnecessarily increase the number of BGP updates and negatively impact network traffic. Over the past decade, researchers have developed a good understanding of which combinations of routing policies lead to oscillations [2]–[8]. Most of these results were based on an abstract model of the interdomain routing system — namely the Simple Path Vector Protocol (SPVP) [9] — that captures how each node selects the highest-ranked path consistent with its neighbors' decisions.

This paper shows that local engineering decisions such as BGP timers and internal router structures can produce short-term artifacts that lead to protocol oscillations not well modeled by SPVP. To capture how these local phenomena affect global convergence, we introduce an extension of SPVP called the Dynamic Path Vector Protocol (DPVP). Although DPVP is seemingly more complicated than SPVP, it actually yields to analysis more easily: we show that DPVP admits a *necessary and sufficient condition* of convergence. Furthermore, we give an algorithm that for most realistic settings efficiently determines whether a DPVP instance is safe, i.e., whether the BGP system as modeled by DPVP converges.

A. Spurious Selection of Lower-Ranked Routes

Earlier studies of interdomain routing assume that routers select and announce the most-preferred available route. However, routers in practice may temporarily announce other recently-available routes, or even withdraw a route when the destination appears reachable. We call such unexpected announcements and withdrawals *spurious updates*. These spurious updates can be caused by several router-level mechanisms that delay the propagation of update messages (to reduce overhead and improve stability) or limit visibility into the alternate routes (to improve scalability), including:

- **Route flap damping** [10]: Route flap damping temporarily suppresses a route if it appears unstable. As a result, a router may temporarily select a less-preferred route.
- **MRAI timers** [11]: The Minimum Route Advertisement Interval (MRAI) timer paces BGP update messages. Delaying message delivery can cause a router to temporarily select a lower-ranked alternate route.
- **Router queuing mechanisms**: The BGP message queues between routers delay the delivery of update messages. These queues, coupled with optimizations that stop generating new messages when the queue grows large, can lead to delays in selecting the highest-ranked route.
- **Cluster routers**: Large routers are distributed,with BGP sessions terminating on different processor blades. To improve scalability, these blades do not exchange full information with each other, which may lead to a temporary selection of a less-preferred route.
- **Proposed router extensions:** Extensions to the BGP route-selection process were proposed to improve router reliability [12] or to reduce convergence time [13]. This changes the timing of routing decisions.

All spurious updates share two common properties: (i) a router can only send spurious updates for a short time after receiving information changing its most preferred route, and (ii) spurious updates are based on routes that have been recently available (including spurious withdrawals because "no route" is always available). DPVP allows any spurious update with these properties. We argue that such model is general enough to capture all spurious updates, but at the same time we show that the model is not overly broad.

Just as local routing policies can affect global convergence [9], these local engineering decisions have global consequences—by triggering oscillations and slowing convergence exponentially. Eliminating all sources of spurious updates would require major changes to router design and the BGP protocol. Some of these mechanisms are important for reducing protocol overhead and improving scalability, making it unappealing to eliminate them entirely. Protocol designers, router designers, and network operators could strive to reduce

the frequency and duration of spurious updates. However, it is not clear that such a quest is warranted or plausible. Rather than advocating for a world free of spurious updates, we argue for a better understanding of their consequences.

B. DPVP Convergence

While allowing spurious updates shrinks the set of BGP configurations that are safe from oscillations, we establish that *most* of the well-studied situations deemed safe under SPVP remain safe even under DPVP. In particular, we strengthen the SPVP-based results of [9] to show that even DPVP is safe in a network without a "dispute wheel" structure. Thus, spurious updates do not affect the large body of research on *safety in dispute-wheel-free settings*. In contrast, BGP safety in more general settings, as well as convergence time, *can* be adversely affected by spurious updates, as illustrated in Section VI.

Our main positive result on convergence is a *combinatorial necessary and sufficient characterization of safe DPVP instances, which is tractable under most typical settings*. We show that a DPVP instance is unsafe if and only if it admits a certain combinatorial structure we call a "CoyOTE" (explained in Section VIII). Although DPVP adds the "complexity" of spurious updates over SPVP, this characterization is surprisingly nice in several aspects that have been elusive for SPVP:

- **Bijectivity**: The absence of CoyOTEs is necessary *and* sufficient. Prior work has only yielded sufficient but not necessary [2], [5]–[7], [14], or necessary but not sufficient [3], [8] conditions of convergence.
- **Tractability in most common cases**: Checking whether a network admits a CoyOTE under general routing policies is NP-complete, just like the weaker question of checking for the sufficient-only condition of No-Dispute-Wheel [9]. Luckily, we were able to find a polynomial time algorithm that verifies safety of BGP configurations for virtually any policy used by network operators in practice.
- **Verifiability**: Given a CoyOTE structure, one can easily *verify* its validity as a proof that a network is unsafe. On a more theoretical note, this also places the formal problem of DPVP safety in complexity class CoNP, relatively much easier than the PSPACE-complete problem of checking safety in a comparable SPVP setting [15].

Roadmap

In Section II we review the Stable Path Problem (SPP), a general theoretical framework for describing interdomain routing. In Section III we formally introduce our DPVP model of BGP built on top of SPP that captures the effects of spurious updates on worst-case BGP convergence. To demonstrate DPVP's versatility and applicability, Section IV describes a variety of real and proposed router behaviors that could cause temporary announcements of lower-ranked routes, and demonstrates global oscillations caused by these behaviors, yet not predicted by the classical SPVP model of BGP. Section V, conversely, establishes that DPVP is not over-broad: we show that any sequence of events allowed by DPVP might indeed occur from combinations of the above causes. Section VI shows examples of theoretical results in the literature that, while correct under the SPVP model, no longer hold in the presence of spurious updates. We show the No-Dispute-Wheel DPVP safety condition in Section VII, and the necessary and sufficient conditions for safety in Section VIII. Section IX presents an algorithm for checking DPVP safety in polynomial time for all "realistic" BGP policies. Finally, Section X shows that this "realistic policy" constraint is necessary — allowing truly arbitrary policies makes it NP-complete to verify safety.

II. The Stable Paths Problem (SPP)

Here we review the Stable Paths Problem (SPP) due to Griffin et al. [4]. The reader familiar with the SPP framework may proceed directly to Section III.

The SPP [4] consists of a graph where each node represents a single BGP speaker, and a fixed node which all other nodes try to reach. Each node has its own set of permitted paths to the origin, and a ranking function that ranks the permitted paths in the order of preference. A solution of the SPP is a global assignment of nodes to permitted paths such that each node is assigned the highest ranked path that can be constructed based on the paths assigned at neighboring nodes. The formal definition of SPP follows.

The simple undirected graph $G = (V, E)$ with nodes $V = \{0, 1, 2, ..., n\}$ represents the network topology. Node 0 is the address *origin* and all other nodes try to establish a path to the origin. Let $\mathsf{neighbors}(v)$ denote the neighbors of node v.

Paths are represented as a sequence of nodes $(v_k\ v_{k-1}...v_1v_0)$ where for each $k \geq i > 0$ we have $(v_i, v_{i-1}) \in E$. An empty path is denoted ϵ. If two paths P and Q are not empty, and the last node in P is the same as the first node in Q, the *concatenation* of the two paths is denoted PQ. A *subpath* of the original path $P = (v_kv_{k-1}...v_1v_0)$ from node v_i to v_j for some $i > j$ is $P[v_i, v_j]$, and $P[v_i]$ denotes a subpath from v_i to the origin. We use $v \in P$ to denote that node v appears in path P.

The permitted paths to the origin are explicitly specified for each node. The set of *permitted paths* for each $v \in V$ is $\mathcal{P}^v$. Any path P that appears in the set is *permitted* at the node v. $\mathcal{P}^0$ is well defined and contains the only valid path to the origin, i.e., the empty path ϵ. Let the collection of all permitted paths be $\mathcal{P} = \{\mathcal{P}^v | v \in V\}$.

Route preference of each node $v \in V$ is captured by its *ranking function* λ^v. If two paths $P_1, P_2 \in \mathcal{P}^v$ and $\lambda^v(P_1) < \lambda^v(P_2)$ then P_2 is *preferred* to P_1 by v. Let the collection of all ranking functions be $\Lambda = \{\lambda^v | v \in V\}$.

Additional requirements pertain to the permitted paths and their ranking. For each λ^v and $\mathcal{P}^v$ we require:

(i) **Paths are simple:** every non-empty path in $\mathcal{P}^v$ is a simple path from v to the origin.
(ii) **Empty path permitted:** $\mathcal{P}^v$ contains the empty path ϵ.
(iii) **Empty path lowest ranked:** $\lambda^v(\epsilon) < \lambda^v(P)$ for all $P \in \mathcal{P}^v$.
(iv) **Strictness:** if $\lambda^v(P_1) = \lambda^v(P_2)$ then either $P_1 = P_2$ or the first edge of the two paths is the same.

A path assignment is a function π that maps each node v to a path in $\mathcal{P}^v$. $\pi(v) = \epsilon$ denotes that node v is not assigned a path to the origin. We write a path assignment as a vector $(P_1, P_2, ..., P_n)$ where $\pi(v) = P_v$, and the path of the origin to itself is omitted. Let $\mathsf{choices}(\pi, v)$ be the set of all possible permitted paths at v that extend the paths assigned to their neighbors:

$$\mathsf{choices}(\pi, v) = \begin{cases} \{(v\ u)\pi(u) | (v,u) \in E\} \cap \mathcal{P}^v & v \neq 0 \\ \{\epsilon\} & \text{o.w.} \end{cases}$$

Let W be a subset of permitted paths $\mathcal{P}^v$ such that each path has a distinct next hop. The best path in W is:

$$\mathsf{best}(W, v) = \begin{cases} P \in W \text{ with maximal } \lambda^v(P) & W \neq \emptyset \\ \epsilon & \text{o.w.} \end{cases}$$

The path assignment π is *stable* at node v if $\pi(v) = \mathsf{best}(\mathsf{choices}(\pi, v), v)$.

The SPP specification is a triple $S = (G, \mathcal{P}, \Lambda)$ consisting of the graph, permitted paths, and ranking functions. The specification S is *solvable* if there exists a stable path assignment π for S, otherwise it is *unsolvable*.

III. DPVP: BGP Model with Spurious Updates

To study the dynamic properties of BGP, we introduce the Dynamic Path Vector Protocol (DPVP), a formal model that allows transmission of stale information in spurious route updates. The DPVP model specifies the dynamics of routing information exchange between routers in the SPP framework. Section III-A informally explains how we model spurious updates. Then, Section III-B defines DPVP dynamics by specifying how a node exchanges routing information and selects a preferred route. A convenient shorthand notation that provides a compact description of a dynamic evolution of the DPVP model is introduced in Section III-C.

A. Modeling the Spurious Updates

For a short period after receiving information that changes the best path, a router may temporarily transmit stale information in the form of *spurious* route announcements or withdrawals. An upper bound on the duration of the spurious behavior is required to prevent propagation of arbitrarily old information. The DPVP model introduces a universal fixed constant τ[1] that serves two purposes. First, it limits the interval after a route change at node v during which stale information may propagate from that node. Second, any stale information that propagates from node v at time t must have been available at node v at some point in the time interval $[t - \tau, t]$.

Specifically, the constant τ serves as an upper bound on the communication delay caused by queuing delays, the MRAI timer, the suppression period of route flap damping, and any other source of spurious behaviors, current or future. Indeed, we deliberately do *not* model the specific sources of spurious updates, so as to not limit our model to the sources thus far observed. Surely other sources may be buried deep inside current router designs, or may arise in the future, and we assert that modeling all of them with a generic finite cutoff is the right approach. That is, we expect that any future design decision that violates this model (i.e., potentially sends spurious updates indefinitely in an otherwise-stable system) would not be accepted by the network operator community.

[1]Stability of an SPP instance in the DPVP model is independent of the actual numerical value of τ.

B. Dynamic Path Vector Protocol (DPVP)

The current time of a global clock is denoted by t.

The internal state maintained by each node v consists of the following. The assigned path $\pi(v)$ represents the most preferred route that is consistent with the information received by the node at the present time. The structure $\mathsf{rib\text{-}in}(v \Leftarrow w)$ maintained by node v contains the most recently processed information received from node w. The set $\mathsf{recentRts}(v)$ contains all routes that node v has had recently available. This set includes any route that is available at the present time t according to the information in the rib-in structure, as well as any route that was available in the time interval $[t - \tau, t]$. The state also includes variable $\mathsf{stableTime}(v)$ which encodes information about stability of the node as defined below.

The stability of a node determines the properties of the information transfer from that node. The node v is stable if $t \geq \mathsf{stableTime}(v)$ and it is not stable otherwise. If a node v is *stable*, any information transfer in the system concerning the assigned path $\pi(v)$ must be accurate, i.e., the neighbors of node v learn the correct most recent route $\pi(v)$. However, if a node is *not stable*, then the neighbors may receive stale information. The stale information received from node v may include any route from the set $\mathsf{recentRts}(v)$.

The dynamic route information exchange is facilitated by *edge activations*. Simultaneous edge activations are allowed. When the edge (w, v) activates, the process shown in Figure 1 is executed. The "if" branch on lines 2–3 is executed if at the time of activation the node w is stable. The $\mathsf{rib\text{-}in}(v \Leftarrow w)$ variable is updated with the most recent information from node w. If node w is not stable, then lines 4–6 are executed, and node v learns information that is potentially stale. Stale information either contains a route withdrawal, or announcement of some recently available route at node w. The commands on lines 7–9 update the list of the recently available routes $\mathsf{recentRts}(v)$. Newly available routes are added, and if a route becomes unavailable at time t, it is scheduled for removal from $\mathsf{recentRts}(v)$ at time $t + \tau$. Finally, the if statement on lines 10–12 determines whether the best route available to node v consistent with the information received thus far changes. If the route changes then $\pi(v)$ is updated accordingly and the node is marked as unstable for a time period τ.

An edge activation sequence σ of sets $(E_0, E_1, \ldots)$ has E_t containing the edges that are activated at time t. An activation sequence is fair if each edge $e \in E$ appears in the sequence infinitely often, i.e., all node pairs continue exchanging routing information indefinitely.

A vertex activation sequence ρ of sets $(V_0, V_1, \ldots)$ has V_t containing the vertices that are activated at time t. A vertex

activate$(v \Leftarrow w)$

1: old-rib-in $=$rib-in$(v \Leftarrow w)$
2: **if** $t \geq$ stableTime(w) **then**
3: rib-in$(v \Leftarrow w) = (vw)\pi(w)$
4: **else**
5: pick some $P \in \{$recentRts$(w) \cup \epsilon\}$
6: rib-in$(v \Leftarrow w) = (vw)P$
7: **if** rib-in$(v \Leftarrow w) \neq$ old-rib-in **then**
8: add rib-in$(v \Leftarrow w)$ to recentRts(v)
9: remove old-rib-in from recentRts(v) at time $t + \tau$
10: **if** $\pi(v) \neq$ best(rib-in, v) **then**
11: $\pi(v) =$best(rib-in, v)
12: stableTime$(v) = t + \tau$

Fig. 1. The DPVP model for router v responding to the activation of edge $v \Leftarrow w$, i.e. v processing information from w.

v activates when all its adjacent edges $(w, v) \in E$ activate simultaneously. We introduce vertex activations merely for convenience to allow more compact notation.

DPVP is stable at time t if the path assignment π is stable and it has not changed in the time interval $[t - \tau, t]$. Note that if DPVP is stable, it is impossible for nodes to exchange stale information, and the state cannot change at any later time.

DPVP is safe if any fair activation sequence, from any starting state, always converges to a stable state. We also define safety under filtering in the same way as [6], [8] do. DPVP is **safe under filtering** if it remains safe under removal of arbitrary subsets of paths from an arbitrary subset of the $\mathcal{P}^v$s (this generalizes the removal of arbitrary nodes and edges).

C. A Shorthand Notation

We introduce a shorthand state transition notation that concisely describes allowed oscillations caused by spurious updates in the DPVP model. A systematic treatment of the causes and consequences of spurious updates follows in Section IV.

An unsafe example of a network configuration is depicted in Figure 2. The network contains three nodes which attempt to obtain a route to node 0. Each node is annotated with its permitted paths, and these paths are listed in the order of decreasing preference. For example, node 1 prefers the path 1230 over 10. To demonstrate that the configuration is unsafe, we must find an oscillation, i.e, an initial path assignment, an activation sequence that activates every edge, and possible spurious announcements that cause a cyclical change of the path assignment. As long as the same activation sequence and spurious announcements are repeated, the oscillation persists.

The shorthand state transition notation that captures a possible oscillation in Figure 2 is as follows:

$$(10, 20, 30) \xrightarrow{2,3} (10, 210, 30) \xrightarrow{1;(1\Leftarrow 2:230)} (1230, 210, 30) \xrightarrow{2} (1230, 20, 30) \xrightarrow{1} (10, 20, 30).$$

The initial path assignment is $(10, 20, 30)$, nodes 1, 2, and 3 have paths 10, 20, and 30 respectively. The nodes or edges activated in each step are listed above the arrow. For example, the path assignment $(10, 210, 30)$ is reached from the initial state by activating nodes 2 and 3. If a spurious announcement is made, this is also described above the arrow. For example, $\xrightarrow{1;(1\Leftarrow 2:230)}$ represents an activation of node 1 where node 1 learns about route 230 from its neighbor 2. The spurious announcement of route 230 is allowed by the DPVP model because recentRts(2) contains route 230 and the path assignment of node 2 keeps changing during the outlined oscillation.

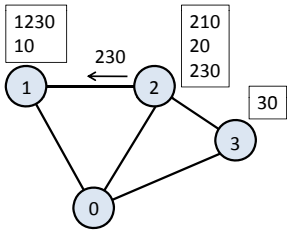


Fig. 2. Example of an oscillation in DPVP. Node 2 exports a low ranked route 230.

It is important to realize that not every shorthand notation that can be written down corresponds to a valid evolution in the DPVP model. Consider for example the following:

$$(10, 210, 30) \xrightarrow{1,2,3;(1\Leftarrow 2:230)} (1230, 210, 30) \xrightarrow{1} (10, 210, 30).$$

This notation is invalid because node 2, which is a stable node with a fixed path assignment 210, cannot spuriously announce the low ranked route 230 in DPVP.

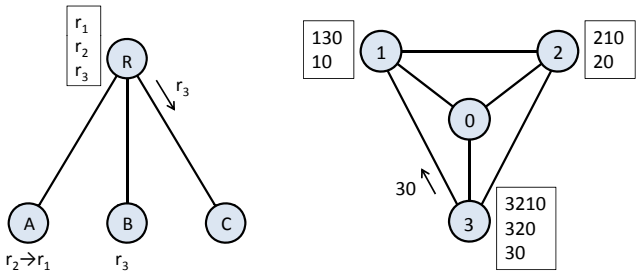
IV. Expressiveness of DPVP

Having specified DPVP formally, we need to establish that it is a realistic model of BGP. We discuss several key sources of spurious updates in BGP, and demonstrate how the DPVP model captures them. These sources of spurious updates include (i) **mechanisms that introduce delays** to improve stability and reduce overhead, e.g. route flap damping and MRAI timers, and (ii) **mechanisms that limit route visibility** due to scalability requirements, e.g. specifics of router implementations. While this list is necessarily non-exhaustive, DPVP is an *abstract* model, and hence it is able to capture other sources of spurious updates as well.

We show specific examples of network configurations where the sources of spurious updates, such as route flap damping or router architectures, cause persistent BGP oscillations. These oscillations are correctly captured by the DPVP model, but the earlier established models of BGP are usually not able to model them.

A. Route Flap Damping

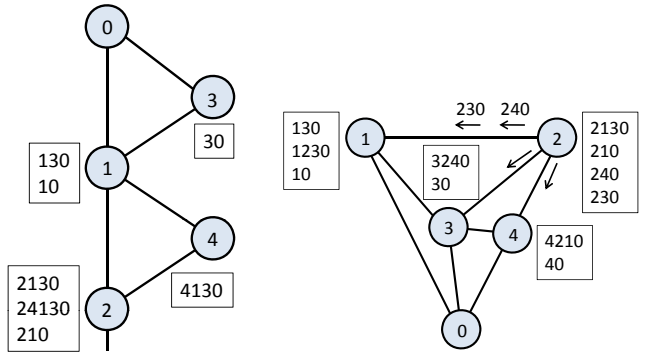
The route flap damping mechanism is used to limit the propagation of unstable routes [10]. When it is enabled, a BGP router maintains a penalty associated with *every* prefix announced by *each* BGP neighbor. Upon receiving a route update from a neighbor, the router increases the penalty. If



(a) After the update $r_2 \rightarrow r_1$, the less preferred route r_3 is temporarily selected.

(b) If node 3 suppresses routes from node 2, it must announce the route 30. This may cause oscillations.

Fig. 3. The effects of route flap damping.



(a) Node 2 temporarily selects route 24130.

(b) MRAI timer in node 2 may cause permanent oscillations.

Fig. 4. The effects of MRAI timers.

the penalty exceeds a given suppression threshold, the route is tagged when it is inserted into the RIB. Tagged routes are not used in the route selection process, and a route with a different next hop will be used. The penalties decay exponentially in time, and if a route doesn't change, its tag is eventually cleared and the route may be used. Next, we show how route flap damping may cause spurious updates, which may in turn lead to permanent oscillations.

A spurious route announcement occurs when the route flap damping mechanism temporarily suppresses a route that would otherwise be preferred. Consider Figure 3(a) where the router R initially learns routes r_2 and r_3 from its neighbors A and B. The router announces route r_2 to router C. If the route r_2 is updated to route r_1, and the penalty associated with BGP speaker A exceed the suppression threshold, router R temporarily suppresses route r_1 and selects and exports route r_3. After the penalty decreases, route r_1 is selected and exported. This appears as a spurious announcement to router C. Route flap damping may also cause a spurious withdrawal. For example, if router R was only connected to routers A and C, the same route update would lead to a route withdrawal from router C. These spurious updates are allowed in the DPVP model.

A permanent oscillation caused by route flap damping may occur in Figure 3(b). First, we will convince ourselves that the configuration is safe in the absence of spurious updates: node 2 must choose either 210 or 20, and thus node 3 will choose either 3210 or 320. Therefore node 1 must choose 10 and the stable state is $(10, 210, 3210)$. However, the following oscillation is possible in DPVP:

$$(10, 20, 320) \xrightarrow{2} (10, 210, 320) \xrightarrow{3} (10, 210, 3210) \xrightarrow{1;(1 \Leftarrow 3:30)} (130, 210, 3210) \xrightarrow{1,2} (10, 20, 3210) \xrightarrow{3} (10, 20, 320).$$

Indeed, this oscillation may occur due to route flap damping. Initially, node 2 activates and changes its route from 20 to 210. When node 3 activates, it processes the route update from node 2, which triggers the route flap damping mechanism. Although node 3 enters the state 3210 in DPVP, in the real BGP system the route 3210 is suppressed and the route 30 is used instead. This explains why in the next activation the spurious announcement 30 is made, and the system enters state $(130, 210, 3210)$. Assuming that the damping penalty decreases, the subsequent activations do not contain any spurious updates, and the system eventually enters the state it started in. This example demonstrates that route flap damping may cause unexpected oscillations that are not predicted by the earlier models of BGP.

B. MRAI Timer

The MRAI timer [11], [16] may also exhibit unexpected spurious updates. When a new route is announced to a peer, subsequent route updates are postponed until the MRAI timer expires[2]. The MRAI timer is applied to route announcements and, depending on the implementation, may [11] or may not [16] be applied to withdrawals. Some previous models of routing do not capture the asynchrony caused by MRAI timers. One such example is a variant of the Simple Path Vector Protocol (SPVP) with vertex activations [9], [17]. We show that when MRAI timers are used, an AS may unexpectedly lose connectivity or select a route which would not be selected in the SPVP model of routing. This may in turn lead to unexpected oscillations.

An unexpected spurious announcement caused by the MRAI timer is illustrated in Figure 4(a). The simplified variant of the SPVP model with node activations does not allow node 2 to select the route 24130. This can be explained as follows. Node 2 can only learn route 24130 after node 1 learns route 130, but then node 2 should select route 2130. However, the behavior of real BGP with MRAI timers differs. Let's assume that node 1 learns route 10 and exports it to node 2. Then it learns route 130, but cannot export it to node 2 because the corresponding MRAI timer has not expired yet. Then node 2 may select the route 24130 learned from node 4, but cannot select route 2130 until the timer in node 1 expires. The announcement of route 24130 by node 2 is possible in our DPVP model as a spurious announcement. The SPVP model with *edge* activations also allows this announcement.

[2]The default value is 30 seconds in eBGP and 5 second in iBGP. However, the values used in practice range between 0 and 30 seconds.

A permanent oscillation caused by MRAI timers may occur in Figure 4(b). This gadget originally appeared in [18] as Figure 3. They show that this gadget is safe in the SPVP model with node activations, but it may oscillate in SPVP with edge activations. We show that the gadget may also oscillate as a result of node 2 using the MRAI timer. Indeed, our DPVP model allows the following oscillation:

$$(1230, 240, 30, 4210) \xrightarrow{2} (1230, 230, 30, 4210) \xrightarrow{3;(3\Leftarrow 2:240)} (1230, 230, 3240, 4210) \xrightarrow{1;(1\Leftarrow 2:240)} (10, 230, 3240, 4210) \xrightarrow{2} (10, 210, 3240, 4210) \xrightarrow{1,3,4;(\{1,3,4\}\Leftarrow 2:230)} (1230, 210, 30, 40) \xrightarrow{2,4} (1230, 240, 30, 4210).$$

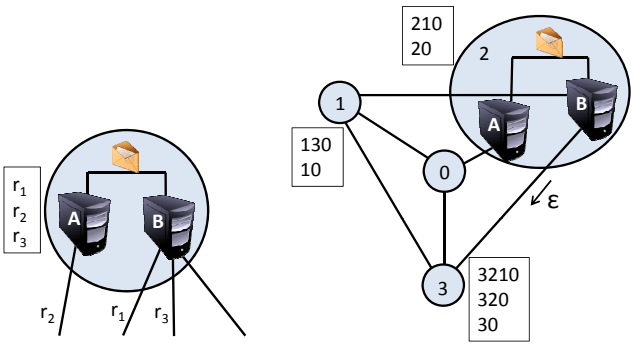
Initially, node 2 activates and changes its route from 240 to 230. This route change prevents the node from immediately announcing the new route 230 to its neighbors due to the MRAI timer. Therefore, when nodes 3 and 1 activate in the next two rounds, they receive the stale route 240. The MRAI timer is also invoked during the second to last round of activations when nodes 1, 3, and 4 receive the stale route 230. In conclusion, MRAI timers may be responsible for spurious announcements that cause permanent oscillations.

C. Lack of Route Visibility

Oscillations can be also caused by spurious updates resulting from a temporary loss of route visibility. We describe such losses of visibility due to the increasingly popular cluster-based router architectures.

Distributed cluster-based routers parallelize functionality across multiple cores and across multiple server blades within each router [19], [20]. These architectures are becoming more common due to the need to scale to larger port densities and traffic demands at a reasonable cost. A router consists of multiple control processor blades, each handling a subset of the BGP sessions. Each blade runs its own software and exchanges reachability information with other blades. While the details of this information exchange differ from one implementation to another, scalability requires each processor blade to usually only announce the currently used route (the best route) to the other blades. Due to asynchrony, a blade may be temporarily unable to see a more preferred route learned by some other blade. This may lead to spurious updates being sent.

A spurious route announcement due to loss of visibility may occur, for example, after the most preferred route is withdrawn, and the second best route is not visible. Consider the example in Figure 5(a) which consists of two communicating processor blades. The cluster-based router prefers route r_1 to r_2 which is still more preferred than r_3. When all three routes are available, blade B selects route r_1 and announces it to the other blade A. If route r_1 is withdrawn, blade B must temporarily select route r_3 while it waits to learn about route r_2 from the other blade. Therefore, blade B spuriously announces route r_3 to other external BGP speakers. A spurious withdrawal can be caused if both routes r_1 and r_3 are withdrawn simultaneously. In such a case blade B must withdraw the route r_1 from its external BGP neighbor until it learns a valid route from the other blade.



(a) After route r_1 is withdrawn, blade B temporarily announces r_3.

(b) The temporary lack of visibility of route 20 by processor blade B causes permanent oscillation.

Fig. 5. The internal architecture of routers (or ASes) is a cause of spurious updates.

A permanent oscillation due to temporary lack of route visibility may occur in Figure 5(b). First, we observe that if no router sends spurious updates, the configuration is safe. This is the same configuration as in Figure 3(b) and hence the stable state must be $(10, 210, 3210)$. However, the following oscillation is allowed in DPVP:

$$(130, 210, 3210) \xrightarrow{2} (130, 20, 3210) \xrightarrow{1,3;(3\Leftarrow 2:\epsilon)} (10, 20, 30) \xrightarrow{2} (10, 210, 30) \xrightarrow{1,3} (130, 210, 3210).$$

In the second round of activations, node 3 receives a spurious withdrawal from node 2. This is explained as follows. Initially, node 2 was in state 210 and blade B used and exported the route 210. However, after node 1 switched to state 130, the route 210 was implicitly withdrawn, and blade B was temporarily left without a route. Before blade B learned about route 20 from the other blade, it sent a spurious withdrawal to node 3. Once again, this example demonstrates that spurious updates may cause unexpected oscillations.

This paper focuses on applying the DPVP model to model the router-level structure of the Internet. However, *DPVP may also be used to model entire ASes as nodes*, if the policies of routers inside an AS are consistent. This model is much coarser, and abstracts away many relevant intra-AS details, but is often reasonable since, from an external viewpoint, there is often very limited information about intra-AS router structure. In such a scenario, **the internal structure of an AS** may cause spurious updates, as well. The situation is analogous to the one with cluster based routers, individual routers correspond to processor blades and the communication of these routers is facilitated by route reflectors [21]. A subset of routers is assigned to each route reflector, which exchanges routing information with these routers and with other reflectors. Each router only learns one best route from each of its route reflectors, hence causing similar loss of visibility as we observed with cluster-based routers. This loss

of visibility can be modeled by earlier BGP models, such as SPVP, *if* the internal structure of each AS is known, and each router and route reflector is represented as a separate node; on the other hand, DPVP allows us to consider questions of safety while remaining agnostic about the ASes' internal structures.

D. Router Queues

Unlike the classical SPVP model [4], DPVP does not explicitly model queues of BGP updates at each edge endpoint. However, the DPVP model captures any behavior that a queue could produce. If a node in the SPVP model processes a particular message from its queue that is older than the most recent update from the same sender, it may be modeled in DPVP as a spurious update. If a message is dropped from the queue, this may be modeled as a DPVP spurious withdrawal. On the other hand, DPVP intentionally allows a more varied behavior than per-edge queues do. While features like route flap damping may be modeled by particular patterns of dropped messages in a queue, some patterns of spurious updates valid in DPVP will not correspond to any possible queue behavior. It is worth noting that *an edge in DPVP does not necessarily model a single BGP session*, but rather models all BGP messages that may be exchanged between two potentially large, complex, distributed modern routers, which may share multiple physical links for redundancy, or might even have several BGP sessions. Since much is unknown in the public domain about the details of bleeding-edge intra-router architectures that vendors use today or may use in the near future, we intentionally limit our assumptions about exactly how two DPVP nodes will interact. We only assume the bare minimum restriction that we expect all routers to obey: the router acts on some reasonable timescale consistently like a monolithic BGP speaker, independently of its internal complexities.

Of course, if a DPVP instance is used to model AS-level behavior, there are many more possible sources of spurious updates from inter-router interaction. Those spurious events would not be well represented at all by any model involving a queue assigned to a node representing a whole AS.

V. All Spurious Updates can be Realized

We demonstrate that any spurious behavior that is allowed in the DPVP model can be realized in the real interdomain routing system. Since we focus on the most general cases, our examples necessarily rely on complex configurations. We note, however, that the examples in Section IV show that many spurious updates in the DPVP model are caused by much simpler configurations.

In general, we need to show that when the most preferred path available to a router changes from r_1 to r_2, the router may spuriously announce an arbitrary sequence of route updates $r_3, r_4, r_5, ...$ to some neighbors before making the final announcement r_2. For full generality, we need to separately consider the two possible causes of the change of the most preferred route from r_1 to r_2. In the first, an update with route r_2 implicitly withdraws route r_1. In the second, route r_2 becomes available in addition to route r_1. We analyze the two cases separately.

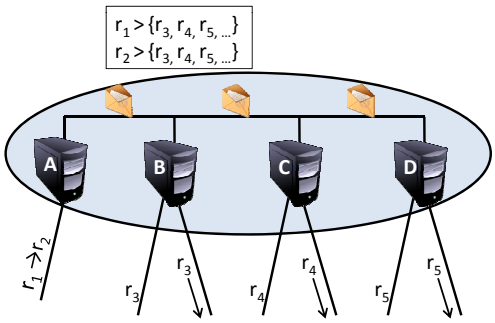


Fig. 6. If blade A damps route r_2 after the update $r_1 \rightarrow r_2$, blades B, C and D announce routes r_3, r_4, and r_5 after they learn about the loss of route r_1.

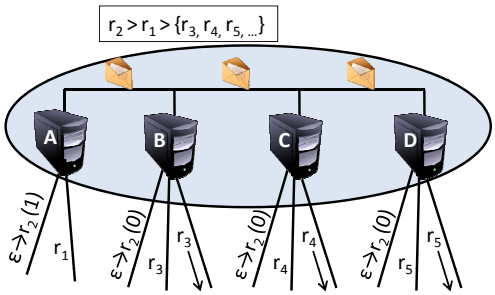


Fig. 7. Route r_2 becomes available (numbers in parentheses are MEDs). If a small asynchrony causes damping in blades B, C, and D only, then these blades announce routes r_3, r_4, and r_5 respectively.

The first case is illustrated in Figure 6 that depicts a cluster-based router consisting of four processor blades. The blade A receives an update with route r_2, which implicitly withdraws route r_1. Let us assume that this update triggers the route flap damping mechanism [10], and route r_2 is temporarily suppressed. Therefore, the router blade A announces to the other blades that no route is available to it. This information first reaches blade B, which identifies route r_3 as the currently best route that is available to it, and announces it to its external BGP neighbors. Similarly, blades C and D spuriously announce routes r_4 and r_5 after they hear from blade A. Assuming that the route flap damping ceases, and the internal state of the cluster-based router becomes consistent before further announcements are made, the final route r_2 is announced next. We conclude our analysis by noting that some of the routes $r_3, r_4, r_5, ...$ can be the empty routes ϵ, and hence spurious withdrawals can be made.

Figure 7 illustrates the remaining case where a new route r_2 becomes available in addition to a less preferred route r_1 which remains available as well. This example relies on the use of the Multi Exit Discriminator (MED). If one or more routers in an AS learn multiple routes from the same BGP speaker, these routes may be tagged with MED values. Routes tagged with smaller MED values are strictly preferred over routes with higher MED values. In our example in Figure 7,

route r_2 has its MED values shown in parentheses. Route r_2 is announced to multiple blades on multiple interfaces perhaps because multiple parallel links connect the cluster-based router to the next hop on route r_2. This is a common occurrence in practice because high speed connections often consist of many parallel lower speed links. The link serviced by blade A has a higher MED value than the other ones, perhaps because it is a backup link, or because the network operator decided that the link needs to be taken down for maintenance in the near future, and traffic needs to be diverted.

The following dynamics in the system in Figure 7 leads to the desired spurious behavior. Initially, route r_1 is announced to the outside by each blade. Because each blade is running its own copy of the routing software, the blades must independently decide if an externally learned route should be damped. The timing of message processing may cause a slight asynchrony where one blade allows a route, whereas it is damped by another blade. Here we will assume that blades B, C, and D damp the newly learned route r_2 while blade A allows it. Therefore, blades B, C, and D will learn that blade A would like to use route r_2 (which should not be used due to its MED), and therefore they will export the spurious routes r_3, r_4, and r_5 similarly as in the previous example. After the internal state of the cluster-based router becomes consistent, the final route r_2 is announced.

VI. Impact on Convergence

After having established that spurious updates may cause permanent oscillations in configurations that are otherwise stable, it is natural to ask if any existing results concerning convergence of BGP change with the introduction of spurious updates. We show in Section VI-A that an exponential slowdown in convergence time may occur. Furthermore, in Section VI-B we show an example of safety conditions in the literature that ensure safety in the absence of spurious updates, but that no longer hold in their presence.

A. Slower than Expected Convergence

Understanding and improving the convergence time [17], [22] has been a central question in the BGP literature. It has been established that while the lower bound on convergence is in general exponential [23], a more favorable bound can be obtained in a Gao-Rexford [2] model of routing. In the Gao-Rexford model, every pair of neighboring ASes has a customer-provider relationship or a peering relationship, and no AS can become an indirect provider of itself. Furthermore, every AS prefers customer routes over peer or provider routes.

A recent paper of Sami et al. [3] shows that in the Gao-Rexford setting, the convergence time of BGP is linear in the depth of the customer-provider hierarchy, or more precisely it is at most $2l + 2$ phases where l is the length of the longest directed customer-provider chain in the AS graph. We define the term *phase* below. We show an example of a network that, when spurious messages are allowed, the convergence takes $(2k{+}1)^{l-2}$ phases where k is the number of spurious messages that a node is allowed to announce after each route change.

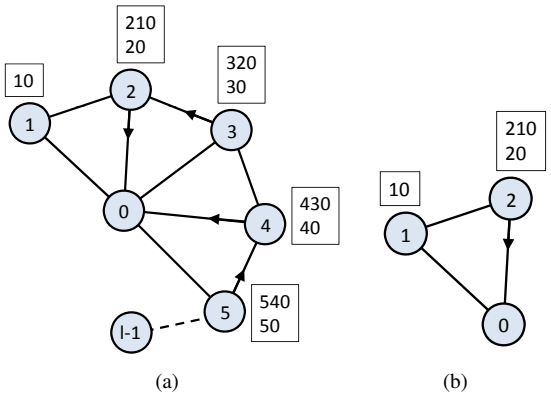


Fig. 8. Slowly converging network configurations.

Our example, which is based on a topology that appears in the original work of Sami et al. [3], shows that *spurious updates may cause an exponential slowdown of convergence* even in the Gao-Rexford setting.

In their work a *phase* is defined as a period of time in which all nodes get at least one update message from each neighboring node, and all nodes are activated at least once after receiving updates from their neighbors. When a node activates, it processes the messages it previously received from all of its neighbors. The example which we describe next demonstrates that because the model of Sami et al. does not capture spurious route updates, it may lead to overly optimistic conclusions.

Figure 8(a) depicts a network with l nodes. Node 1 prefers route 10, and every other node i prefers route $i(i-1)0$ over the direct route $i0$. This set of path preferences is compatible with the Gao-Rexford constraints if node 0 is a customer of every other node and node $i-1$ is a customer of node i for $2 \leq i \leq l-1$. The length of the longest directed customer-provider chain $0, 1, 2, ..., l-1$ is l. The arrows in the figure describe the initial routing choice of each node. Node 1 chooses the empty route, the even numbered nodes route directly to the origin 0, and all odd numbered nodes except node 1 route through the counter-clockwise neighbor. With the exception of node 1 this state is stable.

To show that the convergence in Figure 8(a) may take at least $(2k{+}1)^{l-2}$ phases in a model where each node is allowed to make at most k spurious updates after each route change, we first analyze the convergence of the smaller topology in Figure 8(b). The smaller topology is a special case of the larger one with $l = 3$, and we will show convergence in $(2k + 2)$ phases. In each phase, each node must activate at least once. Spurious updates are only used when we explicitly mention them. The initial state is $(\epsilon, 20)$. We assume that the two nodes activate simultaneously. After the first phase the state is $(10, 20)$, and after the second phase $(10, 210)$. In the third phase node 1 spuriously withdraws the route from node 2. The system reaches state $(10, 20)$ after the third phase, and state $(10, 210)$ after the fourth. This sequence is repeated, node 1 sends spurious updates in every odd phase for a total of k spurious updates, and the state flips back and forth between

$(10, 20)$ and $(10, 210)$. The kth spurious update is made in phase $2k + 1$ and the final state is reached in phase $2k + 2$. Note that the route of node 1 changed once from ϵ to 10, but the route of node 2 changed $2k + 1$ times.

A slow convergence in Figure 8(a) is achieved with the following timing of spurious updates. When the route of node i where $1 \leq i \leq l - 1$ changes, this node makes a spurious announcement of the previously chosen route after each of the nodes with a higher node number sent k spurious updates and these nodes reached a stable state. Using the fact that in Figure 8(b) the state of node 2 changes $2k + 1$ times, we conclude that whenever the route chosen by node i changes, the route chosen by node $i+1$ changes $2k+1$ times. Therefore, the route chosen by node $l - 1$ will change $(2k + 1)^{l-2}$ times and the number of phases required for convergence is $(2k + 1)^{l-2}$.

B. BGP Without a Reel Unsafe

Although BGP convergence without spurious updates has been studied extensively, prior work either concerns *sufficient or necessary* conditions for safety. A classical result shows that the absence of a structure called a *dispute wheel*[3] is sufficient for safety [4]. Safety is also guaranteed when routing policies satisfy the conditions of Gao and Rexford [2].

The strongest result concerning BGP safety prior to the publication of our work has been obtained by Cittadini et al. [6]. They provide the necessary and sufficient conditions for safety *with route filtering*. Filtering allows each node to remove an arbitrary subset of paths from the list of permitted paths. They prove that instances that do not contain a dispute reel[3] are safe under any filtering, and if an instance contains a dispute reel, then there exists a filtering that allows oscillations. Note that these conditions become *sufficient* conditions for safety in the general setting without filtering. The result of Cittadini et al. no longer holds *if we allow spurious updates*.

Consider the example in Figure 9. This is the same topology that appears as Figure 4 in the original work of Cittadini [6] as an example of a safe topology without a reel (the dispute wheel with pivot vertices 1, 2, and 3 is not a reel because each pivot vertex appears in three rim paths, violating Definition VII.2). However, the following oscillation may occur:

$$(10, 20, 30) \xrightarrow{1,2,3;(2\Leftarrow 1:130),(3\Leftarrow 2:210),(1\Leftarrow 3:320)} (1320, 2130, 3210) \xrightarrow{1,2,3} (10, 20, 30).$$

This is a valid oscillation in the DPVP model where the spurious announcements may be caused, e.g., by the details of cluster-based hardware implementation of routers 1, 2, and 3. To make a spurious announcement, router 1 needs to have one router blade responsible for the BGP session with router 0, and another blade responsible for the other two BGP sessions. Routers 2 and 3 could use similar hardware architecture.

[3]Dispute wheel and dispute reel are formally specified in Definitions VII.1 and VII.2.

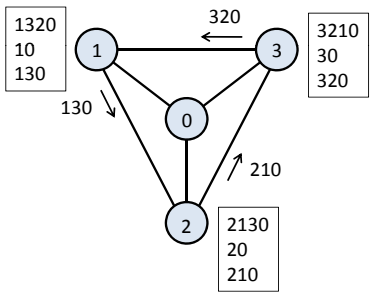


Fig. 9. The graph does not contain a reel. However, spurious updates may cause oscillations.

VII. BGP Safety with Spurious Updates

The unexpected oscillations due to spurious updates beg the question of whether previous results on BGP safety continue to hold under the DPVP model. Fortunately, in Section VII-A we are able to extend the well-studied No-Dispute-Wheel condition [4], sufficient for BGP safety in the SPVP model, to show that it still applies with spurious updates. This result implies that the class of systems that oscillate due to spurious updates is relatively small, and most importantly, earlier results that use the absence of dispute wheel as a condition of safety hold even in the presence of spurious updates. While Section VI-B showed that the absence of dispute reels is *not* sufficient for safety under filtering with spurious updates, Section VII-B introduces a modified structure, a *two-third reel*, which we show to be necessary and sufficient.

A. No Dispute Wheel Implies Safety

The classical result by Griffin et al. [4] shows that BGP is safe in the SPVP model in the absence of *dispute wheels* like the one in Figure 10, formally defined as follows:

Definition VII.1. [4] A *dispute wheel* $W = (U, \mathcal{Q}, \mathcal{R})$ of size k is a set of nodes $U = \{u_0, u_1, ..., u_{k-1}\}$ and sets of paths $\mathcal{Q} = \{Q_0, Q_1, ..., Q_{k-1}\}$ and $\mathcal{R} = \{R_0, R_1, ..., R_{k-1}\}$ such that the following conditions hold. For each $0 \leq i \leq k - 1$, when all subscripts are interpreted modulo k:

(i) Q_i is a path from u_i to the origin.
(ii) R_i is a path from u_i to u_{i+1}.
(iii) $Q_i \in \mathcal{P}^{u_i}$ and $R_i Q_{i+1} \in \mathcal{P}^{u_i}$.
(iv) $\lambda^{u_i}(Q_i) \leq \lambda^{u_i}(R_i Q_{i+1})$.

We strengthen Griffin et al.'s result to show that even with spurious updates, modeled by DPVP, BGP is *still* safe if there is no dispute wheel. This automatically strengthens the

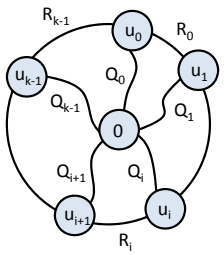


Fig. 10. A dispute wheel of size k.

applicability of the large body of BGP literature that relies on the original No-Dispute-Wheel result under SPVP. We show:

Theorem VII.1. *DPVP instance with no dispute wheel is safe.*

We note that although the absence of a dispute wheel is sufficient for safety, it is *not necessary* in both DPVP and SPVP. That is, dispute wheels *can* occur in safe instances of the routing problem.

Rather than prove Theorem VII.1 separately, we actually derive it as a corollary of the stronger result in Section VII-B.

B. Safety with Filtering

Safety under filtering [8] studies convergence where an arbitrary subset of routes may be removed from the set of permitted paths $\mathcal{P}^v$. In [6] it was shown that the necessary and sufficient condition for safety under filtering in the classical SPVP model is the absence of a particular type of dispute wheel called a *dispute reel*:

Definition VII.2. [6] A *dispute reel* is a dispute wheel which satisfies the following conditions:

(i) **Pivot vertices appear in exactly three paths:** for each $u_i \in U$, u_i only appears in paths Q_i, R_i and R_{i-1}.
(ii) **Spoke and rim paths do not intersect:** for each $u \notin U$, if $u \in Q_i$ for some i, then no j exists such that $u \in R_j$.
(iii) **Spoke paths form a tree:** for each distinct $Q_i, Q_j \in \mathcal{Q}$, if $v \in Q_i \cap Q_j$, then $Q_i[v] = Q_j[v]$.

Section VI-B showed that this result does *not* hold when we account for spurious updates. We define a generalized version of the dispute reel structure, which we prove to be exactly what is needed to identify the systems that are unsafe, but only due to spurious updates. That is, we prove:

Theorem VII.2. *BGP, as modeled by DPVP, is safe under filtering if and only if the network has no "two-third reel":*

Definition VII.3. A *two-third reel* is a dispute wheel which satisfies the second and third condition of dispute reel:

(i) **Spoke and rim paths do not intersect:** for each $u \notin U$, if $u \in Q_i$ for some i, then no j exists such that $u \in R_j$.
(ii) **Spoke paths form a tree:** for each distinct $Q_i, Q_j \in \mathcal{Q}$, if $v \in Q_i \cap Q_j$, then $Q_i[v] = Q_j[v]$.

The intuition for removing the first condition in the dispute reel definition is that spurious behavior of DPVP effectively allows us to "mangle" the rim of the reel, with pivots appearing in multiple rim paths. This would prevent divergence in SPVP, since a pivot would have to stick to one of its options in between its activations, preventing its participation in other pivots' rim paths. However, with spurious announcements, the "multi-tasking" pivot's internal structure may allow it to keep spuriously announcing some of its other available routes in such a pattern as to keep the oscillation going.

PROOF OF THEOREM VII.2: The theorem combines the two implication directions treated separately by Lemma VII.1 and Lemma VII.2. ■

Lemma VII.1. *If a DPVP instance S is unsafe under filtering, it has a two-third reel.*

Lemma VII.2. *If a DPVP instance S contains a two-third reel, it is not safe under filtering.*

We should first formalize the concept of a **DPVP evaluation cycle**, which, given that the system is finite, will arise in any unsafe instance. An evaluation cycle $C = (\Pi, M, \sigma)$ consists of a path assignment cycle $\Pi = \pi_1 \xrightarrow{\sigma_1(M_1)} \pi_2 \xrightarrow{\sigma_2(M_2)} \ldots \xrightarrow{\sigma_k(M_k)} \pi_{k+1}$, with $\pi_1 = \pi_{k+1}$, with a matching edge activation sequence σ and spurious message sequence M. A cycle is well-formed if all the paths can evolve as specified given the message sequence, and the message sequence can continue to be sent indefinitely. That is, no spurious announcements depend on any events that predate the cycle; and every node sending messages changes its chosen path at least once in the cycle, allowing it to keep sending spurious updates, as long as the cycle loops within τ time.

Let the set $\mathsf{values}(C, u)$ be the paths that node u adopts at some point in C. Let F be the set of **fixed** nodes: those that have a fixed path assignment throughout C; and let G be the other, oscillating, nodes. We will need this technical lemma:

Lemma VII.3. *Suppose $P \in \mathcal{P}^v$ is adopted by node $v \in G$ in the cycle C. Then we can write $P = QR$ where the first node on path R is in G and all other nodes further on in R are in F.*

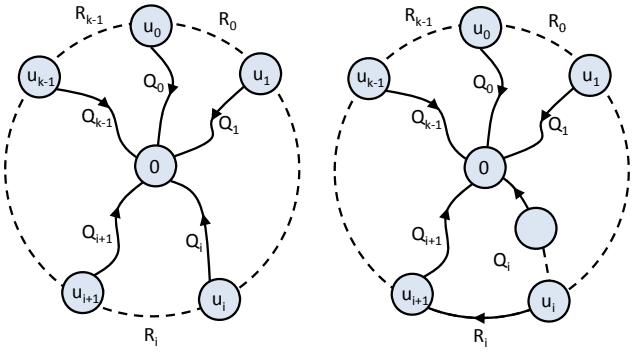
Proof: $v \in G$, and the destination 0 is in F. ■

PROOF OF LEMMA VII.1: We use proof techniques similar to the ones in the SPVP safety proof [4] to show that an unsafe instance has a two-thirds reel.

Let $C = (\Pi, M, \sigma)$ be a well-formed non-trivial cycle. Let $U \subseteq G$ be the nodes that sometimes adopt a path that consists of fixed nodes. U is nonempty, since there are oscillating nodes, and applying Lemma VII.3 to one of them gives a node in U.

We now construct a dispute wheel. Let u_0 be a node in U. Let $Q_0 = (u_0, w_0)Q_0'$ be the path of u_0 such that $w_0 \in F$. Since any such w_0 doesn't send spurious updates in the cycle, one can verify that there is only one such Q_0, and it is the lowest ranked path in $\mathsf{values}(C, u_0)$. Let $H_0 \in \mathsf{values}(C, u_0)$ be the highest-rank path u_0 ever adopts. We have $\lambda^{u_0}(H_0) > \lambda^{u_0}(Q_0)$. Lemma VII.3 lets us decompose $H_0 = R_0 Q_1$, with $Q_1 = (u_1, w_1)Q_1'$, $u_1 \in U$, and R_0 being non-empty since Q_0 is unique at u_0. We repeat this inductively to yield a sequence (u_i), which loops back to u_0 since U is finite. The nodes u_i, spokes Q_i and rims R_i form the dispute wheel.

Finally, we prove that the dispute wheel also satisfies the two-third reel conditions of Definition VII.3. Suppose that condition (i) is not satisfied and there exists a node $u \in Q_i \cap R_j$. Since $u \in Q_i$, u and the rest of $Q_i[u]$ lie in F. Fixed nodes can't send spurious updates, so any repeating announcement of a path via u will end with $Q[u]$ (by induction back from u to the sender of the announcement). Thus, $R_j[u]$ must be a prefix of $Q_i[u]$, requiring $R_j[u] \subseteq F$, but this contradicts $R_j[u]$



(a) All-spoke path assignment $\bar{\pi}$. (b) One-rim path assignment $\bar{\pi}^i$.

Fig. 11. Special path assignments of a two-third reel.

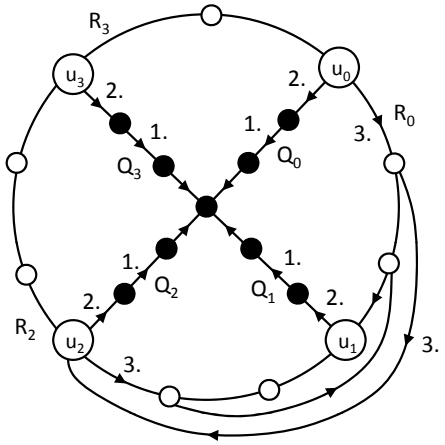


Fig. 12. Sequence of activations that results in permanent oscillations.

ending in $u_{j+1} \notin F$. Suppose condition (ii) is violated, and there exist spoke paths $Q_i, Q_j \in \mathcal{Q}$ and a node $v \in Q_i \cap Q_j$ such that $Q_i[v] \neq Q_j[v]$. Then v cannot be a stable node, a contradiction.

Any instance that is unsafe under filtering will thus have a two thirds reel after some routes are filtered. But if that subinstance has a two thirds reel, "unfiltering" those routes can't remove the reel, guaranteeing a two thirds reel in the original instance as well. ■

PROOF OF LEMMA VII.2: Given a two-thirds reel, we first find the right parts of the system to filter out to cause the oscillation. We then define the path assignments that comprise the oscillation, and finally we show the activation sequence that allows infinite transitions between these path assignments.

Given an SPP instance S containing a dispute wheel W, the *supporting instance* $S[W]$ is the minimal instance which contains the vertices, edges and paths of W. That is, the supporting instance is obtained by filtering all paths except the spoke and rim paths in the dispute wheel, and removing all edges and vertices outside of the dispute wheel.

The system oscillates by alternating between "all-spoke" and "one-rim" path assignments, like those in Figure 11. Formally, an *all-spoke* path assignment is a path assignment $\bar{\pi}$ such that $\bar{\pi}(u) = Q_i[u]$ if $u \in Q_i$, and $\bar{\pi}(u) = \epsilon$ otherwise. A *one-rim* path assignment is a path assignment $\bar{\pi}^i$ such that:

$$\bar{\pi}^i(u) = \begin{cases} Q_j[u] & \text{if } u \in Q_j, u \neq u_i \\ R_i[u]Q_{i+1} & \text{if } u \in R_i \\ \epsilon & \text{otherwise.} \end{cases}$$

Let S be the routing problem instance containing the two-third reel $R = (U, \mathcal{Q}, \mathcal{R})$ of size k. We consider the supporting instance $S[R]$ and construct a fair activation sequence with a sequence of possibly spurious messages that cause oscillations. The main idea is to alternate between the all-path assignment $\bar{\pi}$, and one-rim path assignment $\bar{\pi}^i$. Once a one-rim path assignment $\bar{\pi}^i$ is reached, the all-paths assignment is reached next, followed by the one-path assignment $\bar{\pi}^{i+1}$. This infinite sequence repeats itself, with the index $i+1$ calculated modulo k. The paths of the nodes on the rim changes, and hence these nodes may send spurious updates.

The proof is illustrated in Figure 12. The black nodes are nodes that have a fixed path throughout the evaluation cycle. The paths of the white nodes change and we assume that the convergence time τ is long enough to allow these nodes to make spurious updates. Note that we *design* the oscillation with the property that every rim node oscillates; the system may also have other oscillations without this property.

We start with the empty state, $\pi(v) = \epsilon$ for all v. First, we activate the edges of each spoke path Q_i in outbound order from 0. Any non-pivot spoke node is not on the rim, and the pivots are all on separate spokes, so the filtering causes the spokes to build up the all-spoke path assignment $\bar{\pi}$. In the remainder, we will assume that, unless stated otherwise, each node $u \in R_j$ for each j always sends a spurious withdrawal (ϵ) to every neighbor. All other nodes make non-spurious announcements. Next we activate the edges on path $R_0 = (x_0 = u_0, x_1, \ldots, x_l = u_1)$ in reverse order, i.e., starting from node u_1. The node x_i spuriously announces route $R_0[x_i]Q_1$ to node x_{i-1}. Therefore, the path assignment $\bar{\pi}^0$ is reached. If all edges of each rim node are activated making the spurious withdrawal as specified before, the all-spoke path assignment is reached again. Continuing inductively, we reach assignment $\bar{\pi}^1$, and so on, yielding the infinite activation sequence. ■

PROOF OF THEOREM VII.1 is now a direct corollary of Theorem VII.2. If there is no dispute wheel, there is no two-thirds reel, a special type of dispute wheel, and the resulting safety under filtering implies safety. ■

As should be evident from the ease with which we prove the sufficient conditions for safety in the DPVP model, spurious updates give us extra flexibility when finding oscillations, which makes proofs significantly *easier*. In the remainder of the paper we solve the long-standing open problem of finding the necessary and sufficient conditions for safety.

VIII. THE NECESSARY AND SUFFICIENT CONDITIONS

We now formulate a sufficient and necessary condition for safety in the DPVP model. We show that a system is safe if and only if the configuration allows the existence of a particular combinatorial structure in the network. Since such a

structure can serve as an easy-to-check proof of the system's DPVP safety, the problem of deciding DPVP safety is in NP, which contrasts the intractability results that show SPVP safety checking is PSPACE-complete [15]. Most remarkably, our results in Section IX show that we can check a DPVP system for safety in polynomial time in most practical settings.

Informally, the network will be unsafe if and only if it admits a mapping of each node to a "selected path" and a partition of the nodes into two non-empty sets:

1) **Stable nodes** that select paths which (i) use only other stable nodes, (ii) comprise a consistent routing tree to the destination when taken together, and (iii) are the most preferred such paths for their respective nodes, and
2) **Coy nodes** that are "coy" about joining that stable tree: they select a path that starts with another coy node as a next hop, preferring it over any paths that go straight to the stable tree.

We will shortly formally define this structure, which we dub a **CoyOTE** (for "Coy Optimum at Tree Edges"). We will show that the stable nodes are guaranteed to reach a stable state under any activation sequence, while the coy nodes are capable of triggering DPVP oscillations.

Theorem VIII.1. *An instance of DPVP oscillates if and only if it has a CoyOTE.*

To formally define the CoyOTE structure, we'll need some intermediate technical definitions:

We use $C \subset V$ to denote the set of coy nodes, and always define $S = V \setminus C$ to denote the complementary set of stable nodes. Given a path assignment Π and a coy set C, we define the set $\mathsf{stableChoices}(v, C, \Pi)$ of all paths available to v that go directly to the stable set: all $P \in \mathcal{P}^v$ such that $P = (v, u)P'$ where $u \in S$ and $P' = \pi^u$. If the set $\mathsf{stableChoices}(v, C, \Pi)$ is nonempty, let $\mathsf{bestStable}(v, C, \Pi)$ be the best such path: the unique $P \in \mathsf{stableChoices}(v, C, \Pi)$ for which $\lambda^v(P)$ is maximal. Otherwise let $\mathsf{bestStable}(v, C, \Pi)$ be ϵ.

Definition VIII.1. The pair (Π, C) of a path assignment Π and a **coy set** $C \subset V$ (with the **stable set** defined as $S = V \setminus C$) is a CoyOTE if the following conditions hold:

(i) **Stable origin**: The origin is stable, $0 \in S$.
(ii) **Coy existence**: There are coy nodes, $C \neq \emptyset$.
(iii) **Best stable path at stable node**: A stable node selects its best stable path — for all $v \in S$, $\pi^v = \mathsf{bestStable}(v, C, \Pi)$.
(iv) **Coy node prefers a coy path**: a coy node selects *a* path learned from a coy node — for all $v \in C$ we have $\pi^v = (v, v_{\text{next}}, \ldots, 0)$ with $v_{\text{next}} \in C$. The selected route must be higher ranked than v's most preferred stable route, $\lambda^v(\pi^v) > \lambda^v(\mathsf{bestStable}(v, C, \Pi))$.
(v) **Consistency with stable suffixes**: Every node's selected path is "suffix-consistent" with (S, Π), as defined below.

Definition VIII.2. A path π^v is **suffix-consistent** with (S, Π) if every suffix of π^v that starts with a node $s \in S$ is consistent with π^s: if $\pi^v = (v, \ldots, s, P_s, 0)$ and $s \in S$ (P_s is an arbitrary subpath), then $\pi^s = (s, P_s, 0)$.

We will need the following easy corollary of the definition:

Lemma VIII.1. *In a CoyOTE:*

(i) If v is stable (and $\pi^v \neq \epsilon$), then $\pi^v = (v, P_v^S, 0)$, where P_v^S is a possibly-ϵ subpath containing only stable nodes.
(ii) If v is coy, then $\pi^v = (v, P_v^C, P_v^S, 0)$, where P_v^C is a non-empty subpath consisting of only coy nodes, and P_v^S is a possibly-empty subpath consisting of only stable nodes.

Proof: By induction on the length of paths assigned by Π. Assuming a node chooses a path where a coy node occurs after a stable node, by condition v there is a suffix $(s, c, \ldots, 0)$ with $s \in S$, $c \in C$, but then π^s violates condition iii. With condition iv, the rest follows inductively. ∎

We now tackle the two directions of Theorem VIII.1 separately: Lemma VIII.2 and Lemma VIII.4 establish that a CoyOTE is sufficient and necessary, respectively, for oscillations in DPVP.

Lemma VIII.2. *If a DPVP instance has a CoyOTE, then the DPVP instance can oscillate.*

Proof: Given a CoyOTE (C, Π), it suffices to find an infinite fair activation sequence in which the state of the system continues changing.

Consider starting with an empty path assigned to each node (e.g., as if the destination just went online). The activation sequence starts by every stable node $v \in S$ with $\pi^v \neq \epsilon$ activating in the order of breadth-first search on the stable node tree, from the destination outward, so that each such node gets its path in π^v immediately, and never changes again.

After that, all the coy nodes $v \in C$ activate in a loop so that each one chooses the path π^v at some point. Pick some order of coy nodes, $c_{1\ldots k}$. For each c_i in order, we run 2 rounds of activations. In round 1, activate all the coy nodes on the coy prefix of the path π^{c_i} (nodes $P_{c_i}^C$ in Lemma VIII.1), starting with the node at the edge of the stable tree and moving toward c_i, and have them announce, perhaps spuriously, their suffix of π^{c_i}. By c_i's turn, π^{c_i} will be available, so it or another coy-next-hop path will get selected. Round 2 activates all the same nodes in the same order, and have them send spurious withdrawals. With no coy-next-hop paths available after that, c_i will change its path selection. Repeat this sequence for all the c_is in a loop. Each coy node will keep changing its route, allowing it to keep sending spurious updates.

By condition iii, any stable node v with $\pi^v = \epsilon$ cannot have any paths in $\mathcal{P}^v$ available from its stable neighbors. Thus, it may only receive announcements of allowed paths from coy nodes. By having these remaining stable nodes activate after the second round of each c_i step above, they will never have any allowed paths available to them. ∎

For the other direction, we need another technical lemma:

Lemma VIII.3. *Let an oscillating node $c \in V$ in an instance of DPVP select path P at some point during the oscillation. If the path P contains node $s \in V$ and the node s does not oscillate, then node s must permanently select path $P[s, 0]$.*

Proof: Oscillating nodes only announce recently available

paths. Since s will not be announcing spurious updates for longer than τ after it stops oscillating, any stale path containing node s and a suffix other than $P[s,0]$ will not be announced anywhere for longer than $i\tau$ time after s converges and enough fair activation phases pass. ∎

Lemma VIII.4. *If a DPVP instance oscillates, it has a CoyOTE.*

Proof: Given the oscillating instance I and the corresponding activation sequence, we construct a CoyOTE (C, Π).

Let S_I be the non-oscillating nodes and C_I the oscillating nodes in the instance I. Any non-oscillating node $v \in S_I$ which permanently chooses path $(v,u)P$ learned from an oscillating node $u \in C_I$ can be made to oscillate by changing the activation sequence. Add an activation of the edge (u,v) with a spurious withdrawal, and a second activation of the edge with a (possibly spurious) announcement of path P. We keep adding nodes to C_I until no more nodes can be made to oscillate. We can set Π so that (C_I, Π) is a CoyOTE. For non-oscillating nodes $v \in S_I$, we set π^v to be the path permanently chosen. For oscillating nodes $v \in C_I$, we set π^v be the highest ranked of the path(s) that v chooses in the oscillation.

(C_I, Π) can be seen to be a CoyOTE: The origin is stable and C_I is non-empty, yielding conditions i and ii. After the above iteration, every node s remaining in S_I has a path π^s that is either ϵ or was learned from another node in S_I. Any path from s to a next hop also in S_I is continually available, so s chooses the stable path $\mathsf{bestStable}(s, C, \Pi)$ which must be available, yielding condition iii. If, for $v \in C_I$, the highest ranked path chosen infinitely often in the oscillation has a stable next hop, then, after the next hop stabilizes and τ time passes, that path must indeed become permanently available, which contradicts v continuing to oscillate, yielding condition iv. Lastly, Lemma VIII.3 covers condition v. ∎

IX. Practical Algorithm for Safety Verification

We now consider the algorithmic question of checking safety under DPVP. We give an algorithm, "DeCoy", that greedily shrinks the candidate coy set, and reaches an empty coy set if and only if there is no CoyOTE. This algorithm is always correct, but its runtime depends on the local policies of the nodes. After analyzing the algorithm in general, we explore the large class of policies which enable it to run efficiently, which turns out to cover most BGP policies used in practice.

A. The "DeCoy" safety verification algorithm

The DeCoy algorithm in Figure 13 greedily builds up the candidate stable set, starting with just the destination[4]:

Theorem IX.1. *A DPVP instance is safe if and only if DeCoy terminates with all nodes in the stable set S.*

Proof: **DeCoy terminates with $S = V \Rightarrow$ safety**:

We only need to show that if DeCoy adds vertex $v \in V$ to the stable set S and assigns it path π^v, then for any infinite fair activation sequence in DPVP, node v must permanently choose path π^v. We do so by induction. The claim is true for nodes added to the stable set at beginning, since these nodes in the DPVP model always permanently select the empty path ϵ. Next, let's assume that the claim holds for all nodes in set S after a particular pass through the while loop.

Consider node v that is added to S and assigned path $\pi^v = \mathsf{bestStable}(v, C, \Pi)$ on the next iteration of the while loop, outside the for loop. By contradiction, assume that there exists some infinite fair activation sequence where for any step in the sequence t there exists step $T > t$ in which node v chooses path $P \neq \pi^v$. The case of $\lambda^v(P) > \lambda^v(\pi^v)$ is precluded by the conditions 2(a-b) of the while loop, since, from the induction over the while loop, all nodes already in S will stabilize, and thus any announcements using paths not suffix-consistent with (S, Π) will disappear from the system within $n\tau$ time after that. We also cannot have $\lambda^v(P) < \lambda^v(\pi^v)$: since π^v will always be available to v after τ time after the next hop stabilizes, a less preferred path cannot be chosen after that. Lastly, since the next hop of π^v, more than τ time after stabilization, won't announce any other paths, P must have a different next hop, which precludes the possibility of $\lambda^v(P) = \lambda^v(\pi^v)$, by the strictness constraint of SPP.

Similarly, the claim holds for any node v assigned to S with an empty π^v in the for loop. We know that $n\tau$ time after S stabilizes, no paths in $\mathcal{P}^v$ will be announced anywhere, and thus v will be forced to remain without a route after that.

DeCoy terminates with non-empty $C \Rightarrow$ a CoyOTE exists:

Observe that after DeCoy finishes, the π^v for every node $v \in C$ is left unspecified. We complete the specification of the path assignment Π by setting π^v for every $v \in C$ to the highest ranked path in $\mathcal{P}^v$ that is suffix-consistent with (S, Π). This must be a non-empty path, ranked strictly higher than $\mathsf{bestStable}(v, C, \Pi)$. Otherwise, if $\mathsf{bestStable}(v, C, \Pi)$ is non-empty and there is no such path, v would satisfy the while loop's condition and would be added to the stable set. Alternatively, if the resulting π^v is ϵ — or, equivalently, if $\mathsf{bestStable}(v, C, \Pi)$ is empty and there is no allowed suffix-consistent path higher-ranked than ϵ — then there is no allowed suffix-consistent path at all. In that case, v would

initialize $S = \{0\} \cup \{\text{nodes not connected to } 0\}$; $C = V \setminus S$
while there exists a $v \in C$ such that:
1) v has a neighbor in S, **and**
2) there is **no** path $P \in \mathcal{P}^v$ which is both:
   a) preferred by v over $\mathsf{bestStable}(v, C, \Pi)$, **and**
   b) suffix-consistent with (S, Π)

do
  move v from C to S; **set** $\pi^v = \mathsf{bestStable}(v, C, \Pi)$
  for any $v \in C$ with no paths in $\mathcal{P}^v$ that are suffix-consistent with (S, Π) **do**
    move v from C to S; **leave** $\pi^v = \epsilon$
if C is empty **then return** "safe"
else return "unsafe: (C, Π) is a CoyOTE"

Fig. 13. The DeCoy algorithm.

[4]And possibly some nodes that are permanently disconnected from the destination, which are stable and retain the empty route.

have been stabilized inside the for loop after its last allowed path stopped being suffix-consistent. This insures CoyOTE condition (iv).

CoyOTE conditions (i), (ii), (v) are immediate from the construction if $C \neq \emptyset$.

Note that throughout the algorithm, the set S grows, and path assignments are never changed after being set for the first time. As more nodes are stabilized and the number of constraints required for suffix-consistency increases, the set of candidate paths that are suffix-consistent with (S, Π) monotonically shrinks.

For a stable node s with a non-empty path, consider the stable next hop s' of the path $\mathsf{bestStable}(s, C, \Pi)$, using the *final* values of C, S, and Π. Suppose s' was added to the stable set after s, so it would have been still in C when s was being added. Since $\pi^{s'}$ was suffix-consistent when it was assigned to s', by monotonicity, it was suffix-consistent earlier, too. But, since that path became $\mathsf{bestStable}(s, C, \Pi)$ by the end of the algorithm, it must have ranked higher than whatever path s was assigned as its current $\mathsf{bestStable}(s, C, \Pi)$, using the running value of C and Π when s was stabilized (when s' wasn't stable yet). Thus, s''s path would have negated condition (2) of the while loop, preventing s from being stabilized. Thus, s' was stabilized before s, and, by the time s was stabilized, we had $\mathsf{bestStable}(s, C, \Pi) = (s, \pi^{s'})$, confirming CoyOTE condition (iii).

For a stable node s with an empty path, by the monotonic shrinking of the set of suffix-consistent paths, if there were no allowed suffix-consistent paths when π^s was assigned ϵ, none would have appeared since, guaranteeing that $\mathsf{bestStable}(s, C, \Pi)$ would be ϵ at termination, too, and confirming condition (iii) here, too. ∎

The efficiency of the DeCoy algorithm turns out to be intricately linked to the tractability of *optimizing over policies*, including both router preferences (embodying, e.g., BGP local preference settings), and allowed path sets (embodying, e.g., filtering rules). We first define the fully general requirements on *optimizable policies* that allow DeCoy to run efficiently, and then, in Section IX-B, discuss the wide range of realistic policies that fit into this framework.

Definition IX.1. We say that a node v implements *optimizable policy* if v's permitted path set $\mathcal{P}^v$ and ranking function λ^v allow a polynomial time algorithm which, given a suffix-consistent stable tree (S, Π), can find the highest-ranked suffix-consistent path $P \in \mathcal{P}^v$.

Theorem IX.2. *If all nodes implement optimizable policy, DeCoy runs in polynomial time.*

Proof: We can search through the $O(n)$ paths in $\mathsf{stableChoices}(v, C, \Pi)$ by brute force, to find the $\mathsf{bestStable}$ path. The optimizable policy constraint then lets us check whether the most preferred suffix-consistent allowed path is stable — which is needed to evaluate the second condition of the while loop. The for loop condition is checked by seeing if the optimization returns the empty path as the best suffix-consistent allowed path. The other steps are always efficient.

Nodes are stabilized at most once, so there are a total of at most $|V|$ iterations involved in both loops. Thus, even a brute-force implementation will finish in $O(|V|f(n)^2)$ time, where the efficient policy evaluation runs in time $f(n)$ as a function of the input. With most specific policies, we expect the best runtime to be substantially faster. ∎

DeCoy is thus polynomial-time as long as a node's policy lets it efficiently figure out "what route do I most prefer, given what the rest of the world has permitted me?" We assert that a policy which does *not* allow a tractable answer to this question is somewhat strange. In a heuristic sense, that would entail the router "not knowing what is best for it", even when shown the full network structure and available paths, and instead awaiting whatever the notoriously unpredictable network dynamics give it, with no easily-computable goal in mind.

B. Verifying Safety in Practice

We now present a broad range of realistic policies that turn out to be optimizable. The BGP policies actually used by ASes are varied and complex, but fairly well understood [24]. Typically, a router implements a decision process that first applies import policies to filter out some routes received, then applies a local ranking to select the most preferred available route, and then applies export policies to determine whom to announce it to.

For the purposes of DPVP analysis, we distinguish two levels of policies: (1) **preference** policies prefer some routes less than others, but don't preclude spurious announcements of such routes (modeled with the λ local ranking function); and (2) **filtering** policies which remove a route at such an early point, or at such a low level of the router, precluding even spurious updates involving that route (modeled with the permitted paths set $\mathcal{P}^v$). BGP import/export constraints will often be filtering policies. But BGP policies often thought of as filters, such as "avoidance policies" (like "avoid paths through country X") , are often actually just a depreferencing step in the preference function, without any clear mechanism forbidding spurious announcements of such a route.

Below, we present examples of common policies classified as either filters or preferences in a particular way. But we prefer that the reader remains agnostic about what kinds of policies to file under which of these two categories. This decision depends on the details of the mechanisms generating spurious updates *or* on the algorithm executor's decision to seek a more conservative/robust notion of safety: safety under a wider range of possible (mis)behaviors. Moving a policy from "filtering" status to "preference" status can only allow more spurious update possibilities, yielding a more robust notion of "safety", which we think is natural to seek. Remarkably, in all the cases we consider below, moving policies we consider as filtering policies to the preference stage would keep all the policies efficient.

A filtering policy requires that a path not be used even in absence of any other options, so we expect that each such black-and-white policy has a strong incentive behind

it, typically produced by economic or security constraints. In most typical scenarios, a non-stub AS is only likely to filter based on some combination of the next and previous hop on the path in question (as needed, e.g., for Gao-Rexford economic constraints [2]), and the destination of the packet (as needed for blackholing, motivated by security or politics).

In particular, we admit as **"typical filtering policies"** any disjunction of polynomially many filtering rules that filter paths that are:

(a) learned from neighbor u and announced to neighbor v, or
(b) learned from neighbor u, or
(c) announced to neighbor v, or
(d) violating Gao-Rexford constraints, or
(e) leading to a blacklisted destination.

Rule (a) is quite general and rules (b) through (d) can be implemented by applying rule (a) polynomially many times.

This notably excludes filtering paths that go *through* another transit provider somewhere further in the path, but not on the next hop. The latter type of filtering, for security or for censorship, is unlikely to be effective, due to path diversity in the Internet.

Most **preference policies** observed in practice are covered by the following (in an *arbitrary* order):

(i) split the space of paths into some polynomial number of "categories", each of which disallows the use of some subset of nodes and/or edges, and prefer "categories" in a particular order, or penalize each category differently.
(ii) within each category, prefer routes based on a *stratified shortest path* metric with a logarithmic number of strata, and polynomially many preference levels within each stratum.

The inner category of stratified shortest path metrics, introduced by Griffin [25], allows a rich space of preferences based on a general algebraic semi-ring structure that allows us to "lexicographically stack" several precedence levels of preference constraints that are algebraically isomorphic to a shortest path problem. This allows many common policies beyond just basic shortest-hop-length. For instance, such a policy can be as complicated as:

1) prefer **customer routes**; then
2) penalize paths **passing through country** X by p_x, and through countries Y and Z, by p_{yz}; then
3) pick a route using the **shortest hop path**;
4) of those routes, pick a route that that minimizes some **linear combination** of:
   a) **number of "undesirable" nodes it passes** (e.g. low-performing nodes)
   b) **average data-plane packet loss rate over the path**;
5) then pick the one w/the **lexicographically 1st next hop**.

The number of strata in this unusually complicated example is 4, and no more than a small constant number of strata should ever be needed for realistic policies. Indeed, Griffin's work on such policies in [25] restricts the consideration to $\leq$ 3-stratum policies.

The outer splitting of policies into categories with disallowed nodes/edges allows a yet wider range of policies that don't fit neatly into the stratified shortest path framework. E.g., it allows a fixed penalty for path visiting some set of nodes (rather than counting how many nodes are visited), or it allows any route going through some next-hops (e.g. Gao-Rexford customers) to be strictly preferred over other next-hops (Gao-Rexford providers). This in particular allows the full implementation of any Gao-Rexford business policies, in addition to complex stratified shortest path policies within each business relationship class. The only meaningful restriction that we know of is that the number of categories be polynomial. If there is a list of more than a logarithmic number of "bonuses", each assigning an additive penalty to a path with a particular non-stratified-shortest-path-based feature, that would generate a superpolynomial number of categories. While such a policy *could* be implemented with using common router policy description languages, it is not clear that operators would do so, since it creates an extremely fine-grained distinction between superpolynomially many *combinations of bonuses*, which seems unnecessary in any context we know of. Together, these two levels of preference policy correspond closely to the decision processes described in [24].

Theorem IX.3. *Any combination of "typical" filtering and preference policy classes above constitutes optimizable policy.*

Proof: We need to demonstrate a polynomial-time algorithm to find a node's most preferred suffix-consistent path that would not be filtered by the nodes on it.

We start by modifying the graph to ensure suffix-consistency and compliance with filtering policies, and then run a modified version of the Bellman-Ford algorithm, applied separately to each of the polynomially many categories of the preference policy.

To ensure suffix-consistency with stable set S, convert into one-way edges (1) all edges from $V \setminus S$ to S, in that direction; and (2) the edges of the routing tree, formed by the (suffix-consistent) path assignments of stable nodes with non-empty paths toward d. Then remove edges between two stable nodes that are not used in the routing tree. Any permitted path that is consistent with these changes is suffix-consistent, since once it hits a stable node, it can do nothing but follow the stable tree to the destination.

Then remove any nodes that filter paths to the destination in question.

Then, separately for each category of the preference function, remove the disallowed edges or nodes from the network graph.

We then set a function λ^v by transforming each stratum of the stratified-shortest-path preferences into the isomorphic shortest path problem, and combining all the strata into a single preference value that follows the lexicographic ordering. For example, a 2-stratum policy with top stratum ranging from 1 to 9, and a second stratum ranging from 1 to 99 would be combined into a λ^v ranging from 101 to 999, with the first digit corresponding to the first stratum, and the last two digits

corresponding to the second stratum. The combined λ^v still has a polynomial range, due to the constraints on the strata.

We then run a Bellman-Ford variant to compute paths to destination minimizing λ^v. The only additional constraint is that we must account for node filtering policies, which we do at each node as the Bellman-Ford wavefront propagates. Specifically, we maintain not just the shortest path at each node, but rather each node tracks for each next-hop-and-previous-hop pair (u, v) separately the shortest path that can be imported from u and exported to v.

The correctness argument follows from Bellman-Ford. The runtime scales linearly with the complexity of the preference policy evaluation, and increases only polynomially from maintaining shortest path options for each (u, v). ■

X. Hardness of Some Safety Verification Formulations

We will prove that in the most general settings it is NP complete to verify whether a DPVP instance is safe. The intractability relies on route preferences that are typically not used in practice, such as preference for longer routes over shorter ones, preference for routes of a particular fixed length, or filtering policies applied to transit traffic.

BGP permits the use of regular expressions to specify routing policies. In order to show that verifying the safety of BGP becomes NP complete when regular expressions are used to make the problem inputs (which include routing policies) more compact, we define regular expressions that are allowed to contain the "." metacharacter representing any one autonomous system. For the purpose of our proof, any other definition of regular expressions that allows the "." metacharacter is equivalent.

A *regular expression* is a sequence of metacharacters "." and node numbers. The metacharacter "." represents any node number. For example, the regular expression $3..0$ matches both paths $3, 7, 4, 0$ and $3, 1, 9, 0$. When regular expressions are allowed, ranking functions Λ become *compact ranking functions*: they provide a ranking of regular expressions rather than individual paths. Path P_1 is preferred to path P_2 if the highest ranked regular expression that matches path P_1 is preferred to the highest ranked regular expression that matches path P_2. If two paths match the same regular expression then let the lexicographically smaller path be preferred. If a path does not match any regular expression it is not permitted.

Theorem X.1. *The problem SAFE-REGEXP-DPVP of determining the safety of an instance of DPVP with compact ranking functions is NP-complete.*

Proof: The problem is in NP because if we are given a CoyOTE structure (Π, C), we can check its validity by verifying that the conditions in Definition VIII.1 are satisfied. The path ranking function λ^v is polynomial-time computable and $\lambda^v(P)$ can be evaluated in polynomial time for any v and P. Hence we can also evaluate $\mathsf{bestStable}(v, C, \Pi)$ in polynomial time, and check each condition of Definition VIII.1.

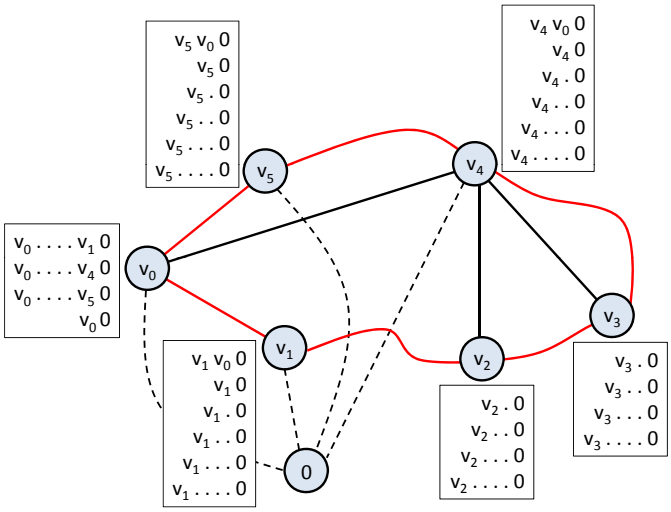


Fig. 14. Example of construction. Solid lines represent edges in the original graph.

The rest of the proof uses a reduction from the Hamiltonian cycle problem, which is one of Karp's 21 NP-complete problems [26]. An instance of the Hamiltonian cycle problem is an undirected graph $G = (V, E)$. The problem asks if there exists a Hamiltonian cycle (a closed loop) that visits each distinct node exactly once.

Suppose we are given an instance I of the Hamiltonian cycle problem. We now construct an instance of the SAFE-REGEXP-DPVP problem D that is not safe if and only if I has a Hamiltonian cycle.

We construct the graph and route rankings of the SAFE-REGEXP-DPVP problem as illustrated in Figure 14. Let $G = (V, E)$ be the graph in the original instance I, let $n = |V|$, and let $T = \{v_i : (v_0, v_i) \in E\}$ denote the set of neighbors of node $v_0 \in V$. We modify the graph G by adding vertex 0 representing the origin. We also add edge $(v_i, 0)$ for each $v_i \in \{T \cup v_0\}$. Finally, we specify the compact ranking functions. Node v_0 allows route $v_0 ... v_i 0$ for each $v_i \in T$ where the number of the dots is such that the length of the route is n. Node v_0 also allows $v_0 0$. Each node $v_i \in T$ allows route $v_i v_0 0$, and routes $v_i ... 0$ where the number of dots is such that the length of the route is between 1 and $n-1$, and shorter routes are strictly preferred to longer ones. Finally, each node $v_i \in V - \{T \cup 0\}$ allows routes $v_i ... 0$ where the number of dots is such that the length of the route is between 2 and $n-1$, and shorter routes are strictly preferred to longer ones.

We now show that if I contains a Hamiltonian cycle then D contains a CoyOTE. Let the Hamiltonian cycle in I be $(v_0, v_1, ..., v_{n-1}, v_0)$. We construct a CoyOTE (Π, C). Let $C = \{v_0, v_1, v_2, ..., v_{n-1}\}$. Let $\pi^{v_0} = v_0 v_1 v_2 ... v_{n-1} 0$, for each $v_i \in T$ let $\pi^{v_i} = v_i v_0 0$, and for all $v_j \neq 0$, $v_j \notin T$ let $\pi^{v_j} = v_j v_{j+1} ... v_{n-1} 0$. It is easy to verify that the conditions of Definition VIII.1 are satisfied and hence (Π, C) is a CoyOTE.

It remains to show that if D contains some CoyOTE (Π, C), then I contains a Hamiltonian cycle. First we show

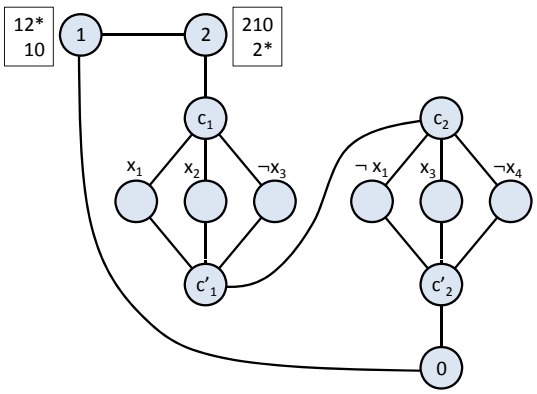


Fig. 15. Example of construction for a 3-CNF formula $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4)$. Nodes labeled x_i avoid nodes labeled $\neg x_i$.

by contradiction that $v_0 \in C$. Assume $v_0 \notin C$. There are two cases, either $\pi^{v_0} = v_0 ... v_i 0$ or $\pi^{v_0} = v_0 0$. If $\pi^{v_0} = v_0 ... v_i 0$ then by Lemma VIII.1 each $v \in \pi^{v_0}$ is stable, and hence each $v \in V$ is stable. This contradicts the fact that (Π, C) is a CoyOTE because condition ii of Definition VIII.1 is not satisfied. If $\pi^{v_0} = v_0 0$ we reach a contradiction as follows. Nodes $v_i \in T$ are stable because their highest ranked path $v_i v_0 0$ is permanently available. It is easy to verify that the remaining nodes $v_i \in V - \{T \cup 0\}$ are also stable because they strictly prefer shorter paths over longer ones, reaching the same contradiction as before.

Since $v_0 \in C$ we must have $\pi^{v_0} \neq v_0 0$ because $0 \notin C$. Then $\pi^{v_0} = v_0 ... v_i 0$ and we know that π^{v_0} is a valid loop free path in the graph $G = (V, E)$ of instance I. $v_i \in T$ and hence $(v_i, v_0) \in E$. Therefore $v_0 ... v_i v_0$ is the sought Hamiltonian cycle in the original graph G. ∎

Next we show that if we allow each node $v \in V$ to filter routes that contain a node belonging to an arbitrary subset of nodes A^v, safety verification becomes NP-complete. We extend the route filtering and route preference rules of Section IX-B to include this filtering step and show NP-completeness.

Theorem X.2. *The problem SAFE-DECISION-DPVP of determining the safety of an instance of DPVP where node $v \in V$ filters routes containing any node $a \in A^v$ and otherwise implements the filtering and preference rules of Section IX-B is NP-complete.*

Proof: The problem is in NP by similar observation as in the proof of Theorem X.1. To show NP completeness we reduce from 3-SAT [26].

We transform a 3-CNF formula of 3-SAT into an instance of SAFE-DECISION-DPVP that is safe if and only if the formula is not satisfiable. This transformation is depicted in Figure 15. For each clause the graph contains the following gadget. If the ith clause in the formula is $(x \vee y \vee z)$, the graph contains nodes c_i and c'_i connected through three intermediate nodes labeled x, y and z. The graph also contains nodes 0, 1 and 2. Node 2 is connected to c_1, c'_i is connected to c_{i+1}, and the last node c'_i is connected to 0.

Node 1 prefers routes learned from node 2 and node 2 prefers routes learned from node 1. Nodes labeled with a variable x avoid every other node labeled with its negation $\neg x$. Node c_1 also avoids node 2. It is easy to see that the original formula can be satisfied if and only if node 2 can use a path to the origin 0 passing through the nodes corresponding to the variables that satisfy each clause. If and only if this path exists nodes 1 and 2 are in dispute and the DPVP instance is not safe. ∎

XI. Conclusion

In this paper we explored BGP safety. We observed that a number of BGP implementation features cause spurious announcements — temporary announcements of routes that are not the most preferred ones. In order to study the properties of BGP in the presence of these spurious announcements, we introduced DPVP, a new dynamic model of BGP. This model proved to be a powerful tool with favorable properties that allowed us to prove the necessary and sufficient conditions of convergence in that model, a question that remained elusive with earlier models of BGP. We also introduced the DeCoy algorithm, an efficient polynomial time algorithm that verifies BGP safety for a rich group of policies that are representative of the configurations used in practice. This also resolves an important question that is of practical interest to both researchers and network operators who need to verify the correctness of their configurations. Design of a distributed privacy-preserving version of the DeCoy algorithm that does not require autonomous systems to reveal their routing policies is the subject of our ongoing investigation.

References

[1] J. W. Stewart, III, *BGP4: Inter-Domain Routing in the Internet*. Addison-Wesley Longman Publishing Co., Inc., 1998.
[2] L. Gao and J. Rexford, "Stable Internet routing without global coordination," *IEEE/ACM Trans. Netw.*, vol. 9, no. 6, pp. 681–692, 2001.
[3] R. Sami, M. Schapira, and A. Zohar, "Searching for stability in interdomain routing," in *Proc. of INFOCOM*, 2009, pp. 549–557.
[4] T. G. Griffin, F. B. Shepherd, and G. Wilfong, "The stable paths problem and interdomain routing," *IEEE/ACM Trans. Netw.*, vol. 10, no. 2, pp. 232–243, 2002.
[5] L. Gao, T. Griffin, and J. Rexford, "Inherently safe backup routing with BGP," in *Proc. of INFOCOM*, 2001, pp. 547–556.
[6] L. Cittadini, G. D. Battista, M. Rimondini, and S. Vissicchio, "Wheel + ring = reel: The impact of route filtering on the stability of policy routing," in *Proc. of ICNP*, 2009, pp. 274–283.
[7] T. G. Griffin, A. D. Jaggard, and V. Ramachandran, "Design principles of policy languages for path vector protocols," in *Proc. of ACM SIGCOMM*, 2003, pp. 61–72.
[8] N. Feamster, R. Johari, and H. Balakrishnan, "Implications of autonomy for the expressiveness of policy routing," in *SIGCOMM 2005*, pp. 25–36.
[9] T. G. Griffin, F. B. Shepherd, and G. Wilfong, "Policy disputes in path-vector protocols," in *Proc. of ICNP*, 1999, pp. 21–30.
[10] C. Villamizar, R. Chandra, and R. Govindan, "BGP route flap damping," 1998, IETF RFC 2349.
[11] Y. Rekhter, T. Li, and S. Hares, "A border gateway protocol 4 (BGP-4)," 2006, IETF RFC 4271.
[12] E. Keller, M. Yu, M. Caesar, and J. Rexford, "Virtually eliminating router bugs," in *Proc. of CoNEXT*, 2009, pp. 13–24.
[13] W. Sun, Z. Mao, and K. Shin, "Differentiated BGP update processing for improved routing convergence," in *Proc. of ICNP*, 2006, pp. 280–289.
[14] J. L. Sobrinho, "An algebraic theory of dynamic network routing," *IEEE/ACM Trans. Netw.*, vol. 13, no. 5, pp. 1160–1173, 2005.

[15] A. Fabrikant and C. H. Papadimitriou, "The complexity of game dynamics: BGP oscillations, sink equilibria, and beyond," in *Proc. of SODA*, 2008, pp. 844–853.
[16] Y. Rekhter and T. Li, "A border gateway protocol 4 (BGP-4)," 1995, IETF RFC 1771 (obsoleted by RFC 4271).
[17] T. G. Griffin and G. Wilfong, "An analysis of BGP convergence properties," in *Proc. of ACM SIGCOMM*, 1999, pp. 277–288.
[18] L. Cittadini, G. D. Battista, and M. Rimondini, "How stable is stable interdomain routing: Efficiently detectable oscillation-free configurations," Dept. of CS&Automation, Roma Tre Univ., Tech. Rep. DIA-132-2008, July 2008.
[19] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "RouteBricks: Exploiting parallelism to scale software routers," in *Proc. of SOSP*, 2009.
[20] "Router reliability research (R3)," 2010, http://r3.cis.upenn.edu/.
[21] T. Bates, R. Chandra, and E. Chen, "BGP route reflection - an alternative to full mesh iBGP," 2000, IETF RFC 2796.
[22] D. Obradovic, "Real-time model and convergence time of BGP," in *Proc. of INFOCOM*, 2002, pp. 893–901.
[23] A. Fabrikant, U. Syed, and J. Rexford, "There's something about MRAI: Timing diversity exponentially worsens BGP convergence," 2010, in submission.
[24] M. Caesar and J. Rexford, "BGP routing policies in ISP networks," *IEEE Network Magazine*, vol. 19, no. 6, pp. 5–11, 2005.
[25] T. G. Griffin, "The stratified shortest-paths problem," in *Proc. COMSNETS*, 2010, pp. 1–10.
[26] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. Plenum Press, 1972, pp. 85–103.