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.

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; the extended version of this work [12] shows 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 III.

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 V). 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], [13], 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 [14].

Roadmap

In Section II we introduce our DPVP model. We then show several examples of oscillations that would have been stable without spurious updates. Section III shows examples of 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 IV, and the necessary and sufficient conditions in Section V. Section VI presents an algorithm for checking DPVP safety in polynomial time for practical BGP policies.

II. 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, which we review in Section II-A. Section II-B informally explains how we model spurious updates. Then,

Section II-C defines DPVP dynamics by specifying how a node exchanges routing information and selects a preferred route. A few examples of configurations that oscillate due to spurious announcements are in Section II-D. Many more examples can be found in the long version of this paper [12].

A. SPP Review

The Stable Paths Problem (SPP) due to Griffin et al. [4] consists of a graph $G = (V, E)$ where each node represents a single BGP speaker, and a fixed node 0 which all other nodes try to reach. Each node $v \in V$ has its own set $\mathcal{P}^v$ of *permitted paths* to the origin, and a *ranking function* λ^v. If for two permitted paths $\lambda^v(P) < \lambda^v(Q)$, then path P is *preferred* to Q by v. The preferences are strict, i.e., if $\lambda^v(P) = \lambda^v(Q)$ then $P = Q$ or the first hop of P and Q is the same. The empty path ϵ is always permitted and is the lowest ranked. *Paths* are represented as sequences of nodes $(v_k\ v_{k-1}...v_1 0)$, PQ is a *concatenation* of the two paths, and $P[v_i]$ denotes a subpath from node v_i to the origin 0.

A solution of the SPP is a *path assignment* π that maps each node v to a path in $\mathcal{P}^v$, that is stable. The assignment π is *stable* if $\pi(v) = \mathsf{best}(\mathsf{choices}(\pi, v), v)$, where:
$$\mathsf{choices}(\pi, v) = \begin{cases} \{(v\ u)\pi(u)|(v, u) \in E\} \cap \mathcal{P}^v & v \neq 0 \\ \{\epsilon\} & \text{o.w.}, \end{cases}$$

and if $W \subset \mathcal{P}^v$ consists of paths with distinct next hops:
$$\mathsf{best}(W, v) = \begin{cases} P \in W \text{ with maximal } \lambda^v(P) & W \neq \emptyset \\ \epsilon & \text{o.w.} \end{cases}$$

As DPVP is grounded in SPP, we refer the reader to [4] or [12] for a systematic treatment.

B. 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. Any future design decision that violates this model, i.e., potentially sends spurious updates indefinitely, would surely be rejected by the designers.

---

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

C. 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 rib-in$(v \Leftarrow w)$ maintained by node v contains the most recently processed information received from node w. The set 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 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 $t \geq$ 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 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 rib-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 recentRts(v). Newly available routes are added, and if a route becomes unavailable at time t, it is scheduled for removal from 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 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.

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 $P \in \{\text{recentRts}(w) \cup \epsilon\}$
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.

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).

The **BGP message queues** in the classical SPVP model [4] are not explicitly modeled by DPVP. However, the DPVP model captures any behavior that a queue could produce. E.g., if a node in the SPVP model processes a message from its queue that is outdated by another update from the same sender, DPVP models this as a spurious update. On the other hand, DPVP intentionally allows more varied behavior than per-edge queues do. While features like route flap damping may be modeled by particular patterns of dropped messages, some patterns of spurious updates valid in DPVP will not correspond to any possible queue behavior. We note that *an edge in DPVP does not necessarily model a single BGP session*, but rather a connection between two potentially large, complex, distributed routers, which may share multiple physical links for redundancy, or might even have several BGP sessions. Since the implementation details of intra-router architectures are proprietary, we intentionally limit our assumptions about DPVP node interactions. We only assume what we expect all routers to obey: the router acts on some reasonable timescale consistently like a monolithic BGP speaker, independently of its internal complexities.

If a DPVP instance models AS-level behavior, there are many more possible sources of spurious updates from inter-router interaction. Those spurious events would not be well represented by queues assigned to entire ASes.

D. Examples of Oscillations

Here we provide two simple examples of oscillations due to spurious updates in DPVP. The spurious updates in the first example are caused by route flap damping, and in the second by features of a specific router architecture. Many more examples can be found in the long version of the paper [12].

Example of a network configuration is depicted in Figure 2. The network contains three nodes which attempt to
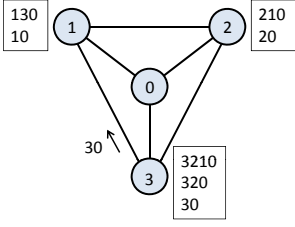
Fig. 2. Example DPVP oscillation caused by spurious announcement of route 30 by node 3.
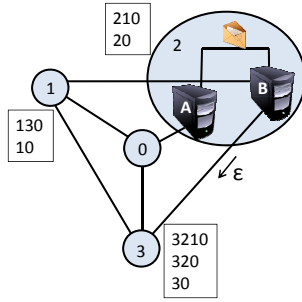

Fig. 3. Temporary lack of visibility of route 20 by processor blade B causes permanent oscillation.

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 130 over 10. It is easy to verify that if no router sends spurious updates, the configuration is safe, and every fair activation sequence leads to the state $(10, 210, 3210)$, i.e, nodes 1, 2, and 3 use paths 10, 210, and 3210, respectively. To demonstrate that the configuration is unsafe in DPVP, 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 transition notation describes the permanent oscillation:

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

The nodes or edges activated in each step are listed above the arrow. For example, the path assignment $(10, 210, 320)$ is reached from the initial state by activating node 2. If a spurious announcement is made, this is also described above the arrow. For example, $\xrightarrow{1;(1 \Leftarrow 3:30)}$ represents an activation of node 1 where node 1 learns about route 30 from its neighbor 3. The spurious announcement 30 is allowed by DPVP because recentRts(3) contains route 30 and the path assignment of node 3 keeps changing during the outlined oscillation.

Route flap damping (RFD) [10] can cause the spurious announcement 30, and the oscillation in Figure 2. When a router receives frequent updates from a neighbor, all routes from that neighbor may be temporarily suppressed in the hope of improving stability. Upon activating for the first time, node 3 processes the update from node 2, triggering the RFD mechanism which suppresses all routes from 2. Since route 3210 is suppressed, node 3 spuriously announces 30 instead. Once 3210 is no longer suppressed, node 3 stops making spurious announcements and the system eventually enters state $(10, 20, 320)$, the same state it started in[2].

Distributed cluster-based router architecture is responsible for oscillations in our second example. This increasingly

---

[2]Here RFD causes a *persistent* oscillation, in contrast to previous work illustrating how RFD can lead to *slower* convergence to a stable state [15].

popular architecture parallelizes functionality across multiple cores or server blades within each router [16], [17]. Each control processor blade handles a subset of the BGP sessions, 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, leading to spurious announcements.

The oscillation relies on a similar configuration as before, but now node 2 is implemented as a cluster-based router, as depicted in Figure 3. The following oscillation may occur:

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

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 learning about route 20 from the other blade, blade B sent a spurious withdrawal to node 3. Once again, because without spurious announcements the example is stable, we have shown that spurious announcements may be responsible for unexpected oscillations.

III. 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 III-A that an exponential slowdown in convergence time may occur. Furthermore, in Section III-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 convergence time [18], [19] has been a central question in the BGP literature. While the lower bound on convergence is in general exponential [20], 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 routes learned from peers or providers.

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 [3]. *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. We
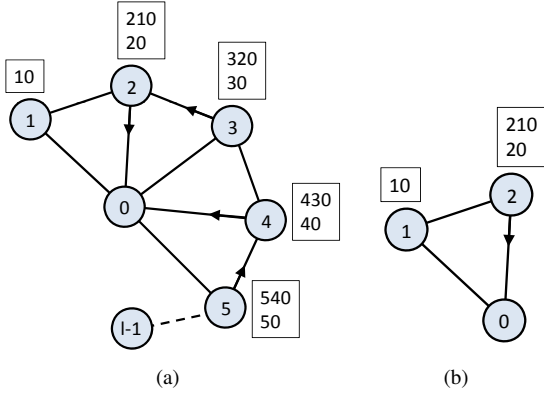
Fig. 4. Slowly converging network configurations.

Fig. 5. The graph does not contain a reel. However, spurious updates may cause oscillations.

show an example where, if spurious updates 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. Our example shows that *spurious updates may cause an exponential slowdown of convergence* even in the Gao-Rexford setting.

Figure 4(a), which originally appeared in [3], depicts a network with l nodes where 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 $2 \le i \le l-1$. The longest directed customer-provider chain has length l. The arrows in the figure describe the initial routing choices.

Let us first show that the smaller topology in Figure 4(b) converges in $(2k+2)$ phases. Spurious updates are only used when we explicitly mention them, and by assumption both nodes activate simultaneously. The initial state $(\epsilon, 20)$ becomes $(10, 20)$ after the first phase, and $(10, 210)$ after the second phase. Node 1 spuriously withdraws the route from node 2 in the third phase, bringing the system to state $(10, 20)$. This sequence is repeated. 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, but the route of node 2 changed $2k+1$ times.

Now it is easy to see that the example in Figure 4(a) converges in $(2k+1)^{l-2}$ phases. Node i only makes a spurious announcement if all nodes with higher node number already emitted k such announcements after their last route change. If the route chosen by node i changes, the route chosen by node $i+1$ changes $2k+1$ times, and thus the route chosen by node $l-1$ changes $(2k+1)^{l-2}$ times.

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

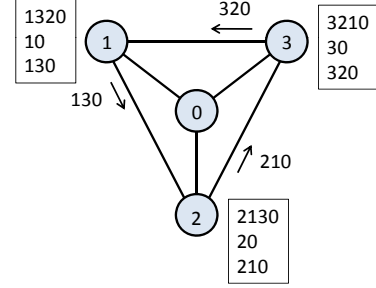[3]Dispute wheel and dispute reel are formally specified in Definitions IV.1 and IV.2.

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 5. 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 IV.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.

IV. 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 IV-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 III-B showed that the absence of dispute reels is
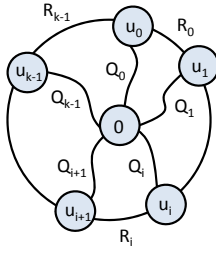
Fig. 6. A dispute wheel of size k.

not sufficient for safety under filtering with spurious updates, Section IV-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 6, formally defined as follows:

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

Theorem IV.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.

Theorem IV.1 can be derived as a corollary of the stronger result in Section IV-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 IV.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 III-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 IV.2. *BGP, as modeled by DPVP, is safe under filtering if and only if the network has no "two-third reel":*

Definition IV.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. A full proof of Theorem IV.2 appears in the long version of this paper [12].

V. 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 [14]. Most remarkably, our results in Section VI 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 V.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 V.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 V.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 V.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 nonempty 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 V.1 separately: Lemma V.2 and Lemma V.4 establish that a CoyOTE is sufficient and necessary, respectively, for oscillations in DPVP.

Lemma V.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 V.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 V.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 V.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 V.3 covers condition v. ∎

VI. 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 greedily builds up the candidate stable set, starting with just the destination[4]:

---

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"

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

The proof of Theorem VI.1 appears in the long version [12].

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 *efficient policies* that allow DeCoy to run efficiently, and then, in Section VI-B, discuss the wide range of realistic policies that fit into this framework.

Definition VI.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 VI.2. *If all nodes implement optimizable policy, DeCoy runs in polynomial time.*

Proof sketch: 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 a non-stable path is preferred over $\mathsf{bestStable}$ for 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. ∎

Effectively, this states that a node's policy should let 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

The BGP policies actually used by ASes are varied and complex, but fairly well understood [21]. Generally, a router filters some policies before importing, selects the best importable route under its local ranking, and exports it to some subset of its neighbors.

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 involving a lower ranked route (modeled with the λ local ranking function); and (2) **filtering** policies definitively remove a route from consideration, 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, 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.

Here, we argue about policies by example, but prefer that the reader remains agnostic about which kinds of policies should be modeled as filters, and which as preferences. Given a policy constraint, the decision of whether to allow spurious violations of it 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. Changing a filtering policy to a preference policy can only allow more spurious update options, yielding a more robust "safety" notion, which we think natural to seek. Notably, in all the cases we consider below, moving policies we consider as filtering policies to the preference stage would keep all the policies optimizable.

---

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

We now outline some examples of a broad range of common policies that turn out to be optimizable. A filtering policy requires a router to not route at all rather than use a particular route, presumably due to a strong incentive, such as an economic or security constraint. We admit as **"typical filtering policies"** any filter of paths that are:

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

Rule (a) is quite general; (b) is a special case of (a).

We allow a very broad variety of **"typical preference policies"**. For instance, we could allow policies as rich as:

1) penalize paths **passing through country** X by p_x, and through countries Y and Z, by p_{yz}; then
2) prefer **customer routes**; 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**.

We can also allow any subset of such rules, most rearrangements of such rules, and a variety of rules not mentioned here. Formally, we admit any preference policy that splits paths into several categories based on edge and node sets they can use, and, within each category, allows arbitrary optimization isomorphic to "stratified shortest paths" (the latter itself a very general policy class, discussed in [22]). The formal definition and proofs for this wide class of policies is deferred to the full version of the paper [12]. To the best of our knowledge, the only vaguely realistic preference policies outside this class require the router to "care" about fine-grained distinctions among a superpolynomial number of path categories.

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

Proof: Proof sketch The algorithm rests on three points:

- the "typical" policy classes above can be optimized by an augmented Bellman-Ford algorithm, executed separately for each "category" in the preference policy;
- the next- and previous-hop filtering requires maintaining multiple Bellman-Ford "wavefronts" at each node, for each relevant neighbor pair
- With relevant integrality and range constraints, a constant number of "levels" of lexicographically combined shortest-path-like preferences can be combined into a single objective function.

The formal policy class definitions, algorithms, and proof are in the full version [12]. ∎

VII. 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. Development 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] M. Suchara, A. Fabrikant, and J. Rexford, "BGP safety with spurious updates," CS Dept, Princeton, Tech. Rep. TR-881-10, July 2010.

[13] J. L. Sobrinho, "An algebraic theory of dynamic network routing," *IEEE/ACM Trans. Netw.*, vol. 13, no. 5, pp. 1160–1173, 2005.

[14] 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.

[15] Z. M. Mao, R. Govindan, G. Varghese, and R. H. Katz, "Route flap damping exacerbates Internet routing convergence," in *Proc. of ACM SIGCOMM*, 2002, pp. 221–233.

[16] 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.

[17] "Router reliability research (R3)," 2010, http://r3.cis.upenn.edu/.

[18] T. G. Griffin and G. Wilfong, "An analysis of BGP convergence properties," in *Proc. of ACM SIGCOMM*, 1999, pp. 277–288.

[19] D. Obradovic, "Real-time model and convergence time of BGP," in *Proc. of INFOCOM*, 2002, pp. 893–901.

[20] A. Fabrikant, U. Syed, and J. Rexford, "There's something about MRAI: Timing diversity exponentially worsens BGP convergence," 2010, in submission.

[21] M. Caesar and J. Rexford, "BGP routing policies in ISP networks," *IEEE Network Magazine*, vol. 19, no. 6, pp. 5–11, 2005.

[22] T. G. Griffin, "The stratified shortest-paths problem," in *Proc. COMSNETS*, 2010, pp. 1–10.