

On the Detection of Policy Conflicts in Interdomain Routing

KARIM MATTAR SAMUEL EPSTEIN IBRAHIM MATTA
Computer Science Department
Boston University, Boston, MA 02215
{kmattar, samepst, matta}@cs.bu.edu

Technical Report BUCS-TR-2010-009
Revises Technical Report BUCS-TR-2009-001

April 16, 2010

Abstract—The *dynamic policy routing model* (DPR) was recently introduced to explicitly model the dynamics of policy routing. DPR extends the formalism of the stable paths problem with discrete synchronous time to capture the propagation of path changes in any dynamic network using a structure called the causation chain.

In this work, we extend DPR by introducing several novel structures, namely, causation fences and policy digraphs that provide further insight into how the dynamics of policy routing manifest in the network. Using our extensions to DPR, we solve a fundamental problem: policy conflict detection. We show how the root cause of any cycle of routing update messages, under any routing policy configuration, can be precisely inferred as either a transient route flap or a policy conflict. We also develop SAFETYPULSE, a token-based distributed algorithm to detect policy conflicts in any dynamic network. SAFETYPULSE has several novel characteristics, namely, it is privacy preserving, computationally efficient and provably correct.

I. INTRODUCTION

The Internet consists of thousands of autonomous systems (ASes). Each AS represents an Internet Service Provider, company or university, that is managed independently. Today, BGP is the routing protocol of choice for connecting these ASes while allowing them to set their routing policies independently. Routing policies determine how a path to a particular destination is chosen out of a candidate set of paths.

This flexibility in configuring routing policies comes at the cost of stability. BGP has been known to suffer from slow convergence time, where ASes continually advertise new routing updates for extended periods of time before reaching a stable routing configuration. Experimental measurements show that interdomain routers may take *tens of minutes* to reach a consistent view of the network after a fault [1]. In addition, no guarantees are made by BGP regarding convergence (*i.e.*, ASes may adopt and discard paths indefinitely [2]).

There has been some seminal work in terms of deriving conditions that guarantee protocol convergence. Griffin *et al.* [3] introduced the *stable paths problem* (SPP), a formalism to reason about the steady-state behavior of BGP. Their solution concept is the stable assignment, where every AS is assigned

its most preferred path out of its available choices. They also showed that the lack of a dispute wheel (*i.e.*, a cyclic dependency in path preferences) is a sufficient condition for convergence. Gao *et al.* [4] showed that restricting the path preferences of ASes to be consistent with their commercial / economic relationships (*i.e.*, prefer customer paths over peer / provider paths) is also a sufficient condition for guaranteeing convergence. Feamster *et al.* [5] showed that the lack of a dispute ring (*i.e.*, a dispute wheel where nodes have path preferences of a special form) under filtering (*i.e.*, preferentially advertising routes) is a necessary condition for convergence. Bounds on BGP's convergence time, under different link failure models, have also been studied (*e.g.*, [6]–[8]).

Many distributed algorithms were also developed to mitigate the effects of harmful policy interactions. This is done by passing diagnostic information alongside route update messages (*e.g.*, a cost metric [9], a precedence metric [10], path-histories [11], as well as event-related tokens [12], [13]).

Other solutions constrain the policy freedom of ASes to a generalized form of shortest path routing, thus guaranteeing convergence (*e.g.*, [4], [10], [14]). There are also numerous offline methods for addressing policy conflicts [15] and analyzing static policy configurations [16]. Other methods focus on identifying the root causes of instability [17].

The *dynamic policy routing model* (DPR) introduced in [18] extends SPP with *discrete synchronous time*. DPR captures the propagation of path changes in any dynamic network irrespective of its time-varying topology. Using DPR the authors established several principles of policy routing dynamics for routing instances that abide by the commercial / economic guidelines in [4]. These principles provided insight into which ASes could directly induce path changes in one another and how cycles of routing updates could manifest in the network.

In this work, we extend DPR by introducing several novel structures, namely, causation fences and policy digraphs that provide further insight into how the dynamics of policy routing manifest in the network. The novelty in this work is that we consider *any configuration of routing policies* that do not

necessarily abide by the commercial / economic guidelines in [4] as assumed in [18]. Using our extensions to DPR, we make the following contributions:

- We introduce policy digraphs that represent the network’s routing dynamics and prove that any sequence of route updates propagating across nodes in the network is a path in the policy digraph, while a dispute wheel is a cycle.
- We prove that the root cause of any cycle of routing update messages can be inferred as either a transient route flap or a policy conflict. More specifically, we prove that any cycle of route updates where a node ends up with a more preferred path must be due to a policy conflict. To the best of our knowledge, this is the most generalized theoretical result for detecting policy conflicts.
- We develop SAFETYPULSE—a token-based distributed algorithm to detect policy conflicts in any dynamic network. SAFETYPULSE has several novel characteristics, namely, it is privacy preserving, computationally efficient and provably correct. In contrast to INTERFERENCEBEAT introduced in [18], that detects potentially unsafe violations of the commercial / economic guidelines in [4], SAFETYPULSE detects policy conflicts in a general setting without restricting the routing policies in any way.

In the following section we provide an overview of these contributions using a simple example.

II. OVERVIEW OF MAIN RESULTS

Consider the sample SPP instance, commonly referred to as BAD GADGET, shown in Figure 1 on the left. The destination is node 0. Each node has a path preference list consisting of two paths where the most preferred path is at the top. For example, node 1 prefers path $\langle 1430 \rangle$ over the direct path $\langle 10 \rangle$.

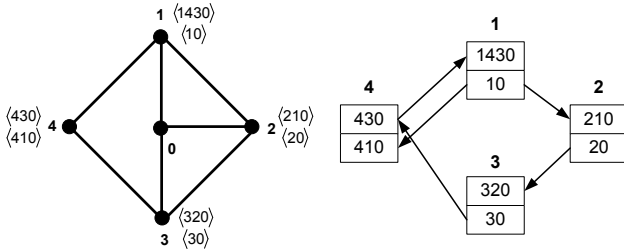


Fig. 1. BAD GADGET instance (left) and its policy digraph (right).

Figure 1 on the right outlines the *policy digraph* of BAD GADGET. Each node in BAD GADGET is represented by a “ladder”. Each step in the ladder denotes a path from the corresponding node’s path preference list. The node’s most preferred path is at the top of the ladder. Two steps (*i.e.*, paths) in different ladders are connected by a directed edge if the source path is a subpath of the target path. We refer to such edges as “subpath” edges. A valid “walk” can start from any step on any ladder and can go down the ladders and across the subpath edges connecting different ladders. Consider the following sample walk:

$$\langle 20 \rangle \langle 320 \rangle \langle 30 \rangle \langle 430 \rangle \langle 1430 \rangle \langle 10 \rangle \langle 210 \rangle \langle 20 \rangle$$

Walks in this structure capture the routing dynamics of BAD GADGET. By routing dynamics we mean how path changes could potentially propagate in the network or more specifically *how paths could potentially be adopted and discarded*. For example, if path $\langle 20 \rangle$ is adopted after link $(2,0)$ becomes available, then path $\langle 320 \rangle$ will also be adopted since it is node 3’s most preferred path. These dependencies are captured by the subpath edges. Walking down the ladder captures the effect of adopting or discarding a less preferred path due to a change in the availability of a path higher up the ladder. For example, if path $\langle 210 \rangle$ gets adopted, moving from path $\langle 210 \rangle$ to path $\langle 20 \rangle$ captures the effect of discarding path $\langle 20 \rangle$ by node 2. This results in path $\langle 320 \rangle$ getting discarded by node 3.

The policy digraph provides valuable insight into the routing dynamics. For example, a path in the policy digraph captures how far routing update messages can potentially propagate. In other words, the longer the paths, the longer it could take for the transient dynamics to die out following a topology change (*e.g.*, a link failure). *We prove that any valid sequence of route updates is a path in the policy digraph.*

On the other hand, a policy conflict (or dispute wheel) is a cycle in the policy digraph where a *path* is repeated. More generally, given the structure of the policy digraph, if a *node* is involved in a cycle of route updates such that it ends up with a more preferred path, then a policy conflict also exists. In our sample walk node 2 initially had path $\langle 20 \rangle$ but ended up with path $\langle 210 \rangle$. Clearly the cycle can repeat indefinitely as the walk can go down the ladder at node 2 and follow the same sequence of nodes again. *We prove that any cycle of route updates where a node ends up with a more preferred path must be due to a policy conflict. Otherwise, we prove that the cycle must be due to a transient route flap.*

It is important to note that our policy digraphs are easier to construct, visualize and reason about compared to dispute digraphs [3]. Dispute digraphs require the *relative path rankings* across nodes to be considered. Our policy digraphs also provide intuition into the operation of many existing solutions that pass diagnostic information alongside route updates. In particular, they provide insight into how the diagnostic information should be *encoded*. For example, SPVP [11] exchanges extended path histories to detect policy conflicts. The existence of a policy conflict is inferred when a node adopts and discards the same *path* in a cycle of update messages. To detect the cycle from our sample walk, SPVP encodes the exchanged path histories as:

$$\begin{aligned} & \langle +20 \rangle \langle +320 \rangle \langle -430 \rangle \langle -1430 \rangle \langle +210 \rangle \langle -320 \rangle \\ & \langle +430 \rangle \langle +1430 \rangle \langle -210 \rangle \end{aligned}$$

In SPVP, any node that switches between two paths always appends the more preferred path. When a switch to a more preferred path is made, a + is appended. Conversely, when a switch to a less preferred path is made, a – is appended. To detect a policy conflict, SPVP needs one cycle of updates to adopt path $\langle 210 \rangle$ and another cycle to discard it.

One could also use policy digraphs to synthetically construct policy routing instances with more complex dynamics. Con-

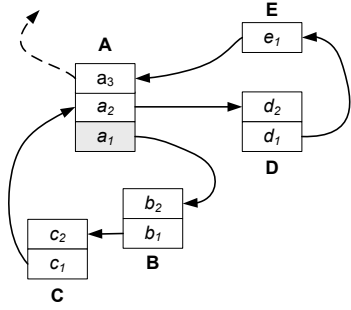


Fig. 2. Sample policy digraph.

consider a walk in Figure 2 that starts when path a_1 is adopted. The adoption of path a_1 triggers a cycle of updates that makes path a_2 available. Since nodes A, B and C are involved in a policy conflict, path a_2 will get withdrawn after another cycle of updates. The more preferred path a_3 , however, may be adopted via another cycle of updates through nodes D and E. If the network stabilizes, no policy conflict will be detected. This highlights that it could potentially take a long time for a node to *adopt and discard* a path. Also, any changes in the underlying topology could stop the propagation of path changes causing policy conflicts to go undetected.

SAFETYPULSE, our token-based distributed algorithm, leverages our theoretical results to construct a more generalized detector for policy conflicts. A node does not need to flap on the *same path* two (or more) times for a conflict to be detected. Instead, if a *node* is triggered twice by a cycle of route updates, checking if the node ended up with a more preferred path is sufficient. This is irrespective of how the underlying topology changes over time. To compare the rankings of the paths involved in the cycle, SAFETYPULSE requires each node to know the path that triggered the cycle. Such is the type of information that must be encoded in SAFETYPULSE's token.

The rest of the paper is organized as follows. Section III provides an overview of the DPR model introduced in [18]. In Section IV we utilize our DPR extensions to derive all our theoretical results that will serve as the foundation for our SAFETYPULSE algorithm in Section V. Finally, we conclude the paper in Section VI.

III. DYNAMIC POLICY ROUTING MODEL

The central notions in DPR, as initially introduced in [18], are that of *action* and *causation*. An action corresponds to a possible routing decision made upon the reception of a routing update message. A causing node corresponds to the node sending that update message. DPR models these two events to construct a *causation chain* over time where each node causes its successor along the chain to take an action.

A. Basic Notation

DPR extends SPP's notation. Time is represented by a non-negative, discrete index $t = [0, \infty)$. The network is represented by a time-dependent graph $G = (V, E)$:

- The set V represents the nodes.
- The set E represents the time-dependent edges. If node u is connected to v at time t , then $(u, v)^t \in E$. Conversely, a lack of connectivity at time t , due to a link failure, is represented by $(u, v)^t \notin E$.

A node's preferential ranking of paths is represented by the \succ operator. If u prefers P over Q then $P \succ Q$. The empty path is $\langle \rangle$. If a path P is forbidden then $\langle \rangle \succ P$. All paths with repeating nodes are forbidden. Paths in G are sequences of the form $P = \langle u_0 u_1 \dots u_n d \rangle$, where d is a distinguished destination node. The concatenation of a path P with node u is $\langle u P \rangle$. A DPR instance consists of a graph and a path preference set, $D = (G, \succ)$.

At any time t , each node u has a path P to the destination d . Path assignments are represented by the function π such that $P = \pi(u, t)$. The available path choices via all possible neighbors v are:

$$\text{Choices}(u, t) = \langle \rangle \cup \{ \langle u, \pi(v, t) \rangle \mid (u, v)^t \in E \}$$

The current best path is $\text{Best}(u, t) = \max_{\succ} \text{Choices}(u, t)$. The state of each node is its best path from the previous round where $\pi(u, 0) = \langle \rangle$ and $\pi(u, t) = \text{Best}(u, t - 1)$. The next-hop of u_0 's assigned path $\pi(u_0, t) = \langle u_0 u_1 \dots u_n d \rangle$ is: $u_1 = \text{NextHop}(u_0, t)$.

Remark 1. While DPR does not explicitly model BGP attributes, such an extension is possible and would only affect the preferential ranking of paths by nodes. This may lead to a different assignment of paths by the function π .

B. Causation Chains and Cycles

Actions represent a change in a node's chosen path between two time steps. A node u performs an action at time t if $\pi(u, t) \neq \pi(u, t + 1)$. Every action of a node is caused by a neighboring node. The cases of action and causation are partitioned by node u 's next-hop node v and the relative ranking of node u 's new and old paths. The functions $\text{Action}(u, t)$ ¹ and $\text{Cause}(u, t)$ are defined in Table I. Consider the first row where node u performs a StepUp action and switches to a new path through a more preferred next-hop node v such that:

$$\begin{aligned} \pi(u, t) &\prec \pi(u, t + 1) \\ \text{NextHop}(u, t) &\neq \text{NextHop}(u, t + 1) \end{aligned}$$

Sample cases of action and causation using a policy routing instance commonly known as BAD GADGET are shown in Figure 3. At time $t = 1$, node 3 performs a StepDown action as it is forced to discard path $\langle 320 \rangle$ and adopt path $\langle 30 \rangle$. This is due to node 2 adopting path $\langle 210 \rangle$ at time $t = 0$. Such sequences of action and causation represent a causation chain that is formally defined next.

Definition 1 (Causation Chains). A *causation chain* is a sequence of nodes where each node y_{i-1} causes the action of y_i . It is represented by $Y = \langle y_0 y_1 \dots y_k \rangle^t$ where

¹We consider actions that are slightly different from the ones in [18] where the next-hop node is not part of each action's condition.

Action(u, t)	Cause(u, t)	Condition	Explanation
StepUp	$v = \text{NextHop}(u, t + 1)$	$\pi(u, t) \prec \pi(u, t + 1)$, $\text{NextHop}(u, t) \neq \text{NextHop}(u, t + 1)$	Node v was not node u 's next hop at time t . However, v advertised a new path to u at time t , causing u to choose a more preferred path through v at time $t + 1$.
StepDown	$v = \text{NextHop}(u, t)$	$\pi(u, t) \succ \pi(u, t + 1)$, $\text{NextHop}(u, t) \neq \text{NextHop}(u, t + 1)$	Node v was node u 's next hop at time t . However, node v changed its path at time t , causing u to choose a less preferred path at time $t + 1$.
StepSame	$v = \text{NextHop}(u, t)$ $= \text{NextHop}(u, t + 1)$	$\pi(u, t) \neq \pi(u, t + 1)$, $\text{NextHop}(u, t) = \text{NextHop}(u, t + 1)$	Node v was node u 's next hop at time t . Node v changed its path at time t , which u chooses to use at time $t + 1$.

TABLE I
CASES FOR ACTION AND CAUSATION.

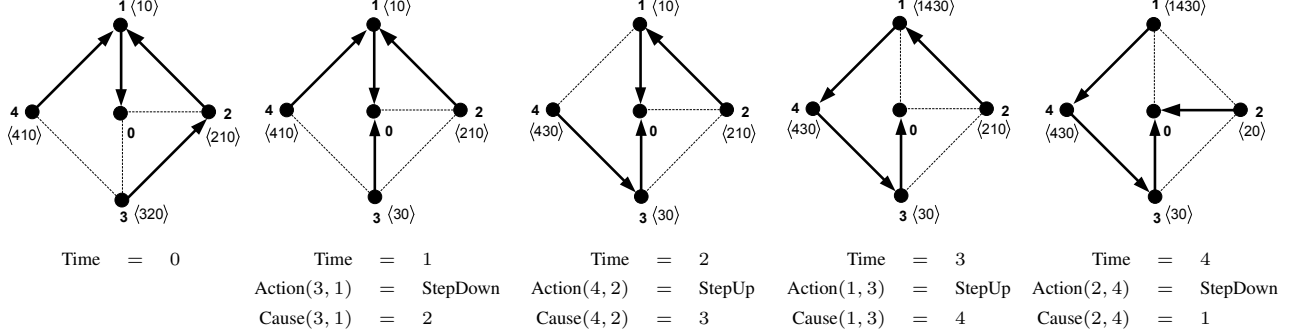


Fig. 3. Sample actions and causation for BAD GADGET.

$Cause(y_i, t + i) = y_{i-1}$ for all $0 < i \leq k$. Time t is defined with respect to y_0 , and it takes i time steps to build the causation chain up to node y_i .

The causation chain in Figure 3 is $\langle 2 \ 3 \ 4 \ 1 \rangle^0$.

Definition 2 (Causation Cycles). A causation cycle is a causation chain $Y = \langle y_0 \ y_1 \ \dots \ y_k \rangle^t$ with a repeated node where $y_0 = y_k$.

A causation cycle in Figure 3 is $\langle 2 \ 3 \ 4 \ 1 \ 2 \rangle^0$.

Remark 2. In terms of the synchronicity of DPR, the authors in [19] show that this is not a drawback and that DPR has sufficient expressive power to model asynchronicity. For completeness, we provide a brief overview of the key ideas in Appendix A.

IV. OUR EXTENSIONS TO DPR

We extended DPR by introducing several novel *time-invariant* structures, namely, *causation fences* and *policy digraphs*. The main components of our extensions to the DPR model and the theoretical results we derived are outlined in Figure 4.

We prove that causation chains manifest in a manner that can be precisely defined. We prove that any causation chain is composed of alternating *adopting* and *discarding* subchains. Using this property we introduce the time-invariant structure, *causation fences*, that capture the key path changes along a causation chain. Using causation fences we introduce *policy digraphs* and prove that causation chains and dispute wheels are paths and cycles in the policy digraph, respectively. Finally, we prove that a causation cycle, representing a cycle of routing

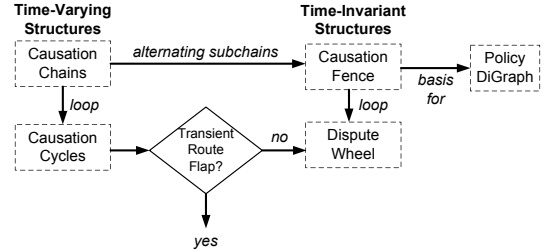


Fig. 4. Overview of our extensions to DPR.

update messages, is either due to a transient route flap or a dispute wheel and that the root cause can be inferred.

A. Causation Fences

Next we distill the *time-invariant properties* of causation chains using a structure we call the *causation fence*. We first show that causation chains are not random sequences of nodes (and their associated actions) as one would expect. Instead, the propagation of path changes in the network can be precisely formalized. More specifically, causation chains can be decomposed into two alternating types of subchains, namely, *adopting* and *discarding* subchains.

A causation subchain consists of consecutive nodes $\langle y_i \ y_{i+1} \ \dots \ y_j \rangle^{t+i}$ where y_i and y_j are the head and tail nodes, respectively. The head node introduces a change into the subchain by changing its current path. Hence, $\pi(y_i, t+i) \neq \pi(y_i, t+i+1)$. The time t is defined with respect to the first node on the original causation chain and it takes i time steps to reach node y_i in the subchain.

In an adopting subchain the head node y_i makes a new

path available that all subsequent nodes adopt. In Figure 5, for example, node 1 makes path $\langle 10 \rangle$ available that node 2 adopts. Node 3 in turn adopts path $\langle 3210 \rangle$ when node 2 makes path $\langle 210 \rangle$ available.

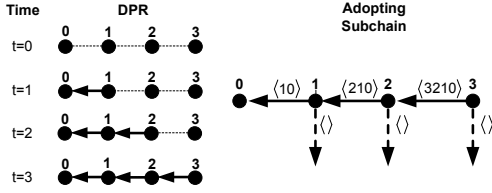


Fig. 5. An example of an adopting subchain. Adopted/discarded paths are represented by solid/dotted arrows, respectively.

Definition 3 (Adopting Subchain). An adopting subchain of Y is $\langle y_i y_{i+1} \dots y_j \rangle^{t+i}$ from y_i to y_j for $i < j$ where $Action(y_k) \neq StepDown$ for all $i < k \leq j$. This is irrespective of y_i 's action.

On the other hand, all nodes in a discarding subchain are initially using a path through the head node y_i . However, y_i discards this path, forcing all subsequent nodes to choose alternate paths.

Definition 4 (Discarding Subchain). A discarding subchain of Y is $\langle y_i y_{i+1} \dots y_j \rangle^{t+i}$ from y_i to y_j for $i < j$ where $Action(y_k) \neq StepUp$ for all $i < k \leq j$. This is irrespective of y_i 's action.

Lemma 1 (Chain Decomposition). Every causation chain $Y = \langle y_0 y_1 \dots y_k \rangle^t$ can be decomposed into alternating adopting/discarding subchains, $Y = Y^0 Y^1 \dots Y^n$, where the tail node of subchain Y^i is the head node of subchain Y^{i+1} .

Proof. This can be trivially shown with a recursive construction. Starting with a causation chain $Y = \langle y_0 y_1 \dots y_k \rangle^t$, we look at the last node y_k and add it to the end of a new subchain Y' . We construct either an adopting or a discarding subchain depending on y_k 's action. If the action of y_k is StepUp or StepSame, then Y' is an adopting subchain. We continue adding nodes y_i to Y' starting from $i = k - 1$ until we reach a node y_j such that $j \leq i$ and its action is StepDown. At this point we start constructing a discarding subchain. We continue recursing until we reach y_0 which is added to the current subchain Y' regardless of its action. ■

This will serve as the basis for constructing our time-invariant causation fence structure. Figure 6 shows the alternating subchains of BAD GADGET.

The causation fence is a structure that distills the core elements (*i.e.*, path changes) in a causation chain. In particular, it only concerns itself with the head and tail nodes of adopting/discarding subchains. The only paths that the causation fence concerns itself with are the adopted and discarded paths in the subchains.

Definition 5 (Causation Fence). A causation fence is formally defined by $F = (\mathcal{N}, \mathcal{R}, \mathcal{Q})$ where:

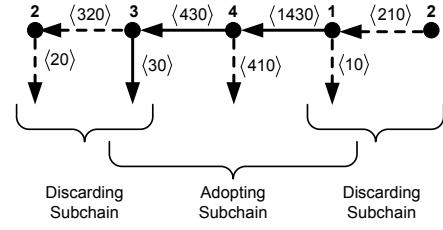


Fig. 6. Alternating subchains of BAD GADGET. Adopted/discarded paths are represented by solid/dotted arrows, respectively. Horizontal paths are more preferred than vertical paths.

- \mathcal{N} is the set of, not necessarily unique, n pivot nodes such that $\mathcal{N} = \{u_0, \dots, u_{n-1}\}$.
- \mathcal{R} is the set of rim paths, where each $R_i \in \mathcal{R}$ is a path from u_i to u_{i-1} .
- \mathcal{Q} is the set of spoke paths, where each $Q_i \in \mathcal{Q}$ is a path from u_i to destination d .
- Each node u_i (except the first and last nodes) prefers a path through its rim and neighbor's spoke path over its own spoke path: $R_i Q_{i-1} \succ Q_i$.

The causation fence can be seen as an open-ended dispute wheel. A sample causation fence is shown in Figure 7. The first and last pivot nodes are missing their (potential) rim and (potential) spoke paths, respectively. The exact manner in which a causation fence manifests (*i.e.*, the alternating adopting and discarding subchains property), is what will allow us to precisely infer the root cause of a causation cycle.

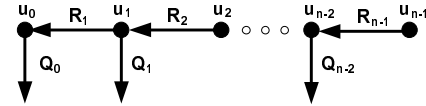


Fig. 7. Causation fence.

Lemma 2 (Chain-Fence Relationship). Every causation chain $Y = \langle y_0 \dots y_k \rangle^t$ is equal to the concatenated rim paths $R_1 \dots R_{n-1}$ of a causation fence $F = (\mathcal{N}, \mathcal{R}, \mathcal{Q})$.

Proof. Using Lemma 1, we break up the causation chain Y into n causation subchains Y^0, Y^1, \dots, Y^{n-1} , where each subchain Y^r is of the form: $Y^r = \langle y_0^r \dots y_s^r \rangle^{t^r}$. The first node y_0 in causation chain Y and the end node y_s^r of each subchain Y^r are added as pivot nodes into the causation fence F . The rim paths of F are the paths that connect each pair of pivot nodes, u_i to u_{i-1} . There are two cases to consider. If the pivot nodes are part of an adopting subchain then the first pivot node u_{i-1} is the head of the subchain. Pivot node u_{i-1} makes a new path available that all subsequent nodes along the subchain including u_i adopt. Thus, during the course of routing, once an adopting subchain is built, all nodes in the subchain are on the rim path that is being created. This rim path connects u_i to u_{i-1} . A similar argument follows if the pivot nodes are part of a discarding subchain where all nodes in the subchain were on the rim path, connecting u_i

to u_{i-1} , that is being discarded. Note that the causation chain propagates in the opposite direction of the paths being created. ■

Figure 8 shows the causation fence induced by the causation chain in Figure 6.

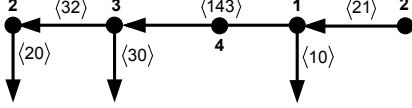


Fig. 8. Causation fence example.

B. Dispute Wheels

Griffin *et al.* introduced *dispute wheels* in [3], where their existence is a necessary condition for an SPP instance to not have a stable assignment. A dispute wheel, as shown in Figure 9, represents a cyclical set of path preferences. A dispute wheel W is defined by $W = (\mathcal{N}, \mathcal{R}, \mathcal{Q})$, where:

- \mathcal{N} is the set of n unique pivot nodes such that $\mathcal{N} = \{u_{n-1}, \dots, u_0\}$.
- \mathcal{R} is the set of rim paths, where each $R_i \in \mathcal{R}$ is a path from u_i to u_{i-1} (with subscripts modulo n).
- \mathcal{Q} is the set of spoke paths, where each $Q_i \in \mathcal{Q}$ is a path from u_i to d .
- The path preference of each node u_i is $R_i Q_{i-1} \succ Q_i$.

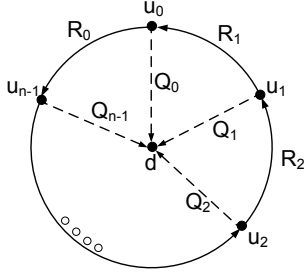


Fig. 9. Dispute wheel.

Here we introduce *proper* dispute wheels where the rim paths form a simple cycle (*i.e.*, no nodes are repeated other than the starting and ending node) and show that every dispute wheel *must* contain a proper wheel inside it.

Theorem 1. Every non-proper dispute wheel $W = (\mathcal{N}, \mathcal{R}, \mathcal{Q})$ contains within it a proper dispute wheel.

Proof: Assume W is not proper, then there exists a non-pivot node v such that $v \in R_i$ and $v \in R_j$, where $i < j$, as shown in Figure 10.

From W a smaller dispute wheel $W' = (\mathcal{N}', \mathcal{R}', \mathcal{Q}')$ can be constructed. There are two cases for this construction, depending on the path preferences of v :

- 1) $R_j(v)Q_{j-1} \succ R_i(v)Q_{i-1}$. W' is defined as:

$$\begin{aligned} \mathcal{N}' &= \{v, u_{j-1}, \dots, u_i\} \\ \mathcal{R}' &= \{R_j(v), R_{j-1}, \dots, R_{i+1}, R_i(u_i, v)\} \\ \mathcal{Q}' &= \{R_i(v)Q_{i-1}, Q_{j-1}, \dots, Q_{i+1}, Q_i\} \end{aligned}$$

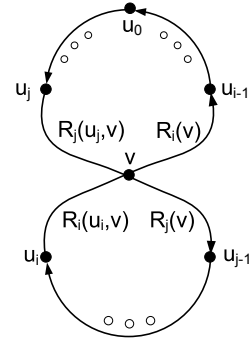


Fig. 10. Non-proper dispute wheel. $P(a, b)$ is the subpath of P starting with a and ending with b . $P(a)$ is the subpath of P starting with a .

This results in the dispute wheel in Figure 11.

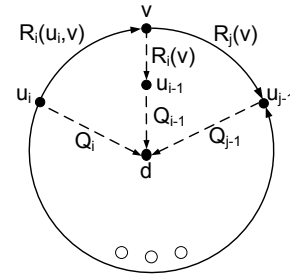


Fig. 11. Smaller dispute wheel case 1.

- 2) $R_j(v)Q_{j-1} \preceq R_i(v)Q_{i-1}$. W' is defined as:

$$\begin{aligned} \mathcal{N}' &= \{u_{n-1}, \dots, u_j, v, u_{i-1}, \dots, u_0\} \\ \mathcal{R}' &= \{R_{n-1}, \dots, R_j(u_j, v), R_i(v), R_{i-1}, \dots, R_0\} \\ \mathcal{Q}' &= \{Q_{n-1}, \dots, Q_j, R_j(v)Q_{j-1}, Q_{i-1}, \dots, Q_0\} \end{aligned}$$

This results in the dispute wheel in Figure 12.

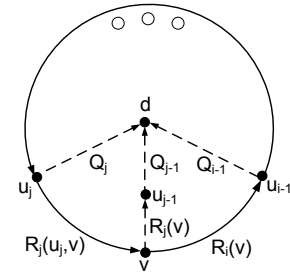


Fig. 12. Smaller dispute wheel case 2.

Thus, every non-proper dispute wheel W contains a smaller dispute wheel W' . Either W' is proper or it also contains a smaller dispute wheel W'' . Since this reasoning can only repeat a finite number of iterations, every non-proper dispute wheel W contains a proper dispute wheel. ■

In the next section we prove that every cycle in a policy digraph represents a dispute wheel and vice versa.

C. Policy Digraphs

Policy digraphs simultaneously represent several DPR structures. In particular, we prove that causation chains and dispute wheels are represented as paths and cycles, respectively.

Definition 6 (Policy Digraph). *Given a DPR instance $D = (G, \succ)$, the policy digraph is $O(\succ) = (E, V)$ where each node $P \in V$ represents a realizable path in G and is referred to as a pnode. Between each pair of pnodes P and Q , there can be one of two edges:*

- *Subpath Edge.* If $Q = \langle u P \rangle$ for some node u in G , then P has a subpath edge to Q .
- *Policy Edge.* If $P \succ Q$, then P has a policy edge to Q .

To simplify the representation of a policy digraph O , all pnodes in O that are paths originating from a single node $u \in G$ are represented by a single set of stacked boxes—a stacked pnode. Each pnode within a stacked pnode has an *implicit* policy edge to every pnode below it. A sample policy digraph can be seen in Figure 1 on the right.

Theorem 2 (Chains in Policy Digraphs). *Every causation chain $Y = \langle y_0 y_1 \dots y_k \rangle$ of a DPR instance $D = (G, \succ)$ is a path in its corresponding policy digraph $O(\succ)$.*

Proof. From Lemma 2, every causation chain is equal to the concatenated rim paths of a causation fence represented by: $F = \{\mathcal{N}, \mathcal{Q}, \mathcal{R}\}$. Each pivot node u_i prefers a path through its rim and neighbor's spoke path over its own spoke path: $R_i Q_{i-1} \succ Q_i$. Thus, causation fence F is a path in policy digraph O as shown in Figure 13. This in turn implies that every causation chain Y is a path in O .

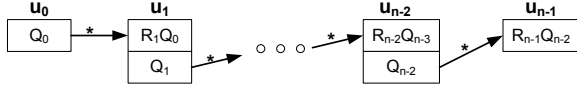


Fig. 13. Causation fences are paths in policy digraphs. The * notation implies a series of subpath edges through pnodes.

Theorem 3 (Cycles in Policy Digraphs). *Every dispute wheel $W = \{\mathcal{N}, \mathcal{Q}, \mathcal{R}\}$ of a DPR instance $D = (G, \succ)$ is a cycle in its corresponding policy digraph $O(\succ)$. Similarly, every cycle in $O(\succ)$ corresponds to a dispute wheel W .*

Proof. This can be seen by drawing the policy and subpath edges for each pnode (*i.e.*, realizable path) of W in O , as shown in Figure 14. A sample cycle (and hence dispute wheel) could start and end at pnode $R_0 Q_{n-1}$ as follows: $\langle R_0 Q_{n-1} Q_0 R_1 Q_0 Q_1 \dots Q_{n-1} R_0 Q_{n-1} \rangle$

D. Detecting Dispute Wheels

Once a causation cycle $Y = \langle y_0 y_1 \dots y_k \rangle^t$ where $y_0 = y_k$ is realized, it implies that the change instigated by y_0 caused a series of actions to propagate along Y until y_k (*i.e.*, y_0)

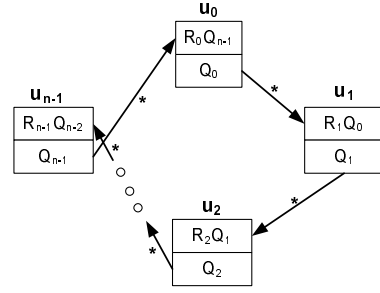


Fig. 14. Dispute wheels are cycles in policy digraphs and vice versa.

receives another route update. Given any causation cycle Y , we answer the following questions:

- Could the cause that induced Y be inferred?
- Could y_0 perform that inference *locally and independently*?

To infer the exact cause that induced Y , we show that if y_k has a more preferred path at the end of the causation cycle, at time $t + k$, than the path it had at time t , then a dispute wheel *must* exist. Otherwise a transient route flap occurred at time t (*i.e.*, a path was withdrawn or made available). We also show that y_0 can indeed perform that inference locally, but *not independently*. This has implications on how policy conflicts can be detected in practice.

We know that any causation cycle Y , of a DPR instance $D = (G, \succ)$, induces a causation fence $F = \{\mathcal{N}, \mathcal{R}, \mathcal{Q}\}$ where the first and last pivot nodes are the same, $u_0 = u_{n-1}$, as shown in Figure 15. Using F , we show the necessary condition for F to be a dispute wheel in Lemma 3. That condition is based on the relative ranking of paths Q_0 and $R_{n-1} Q_{n-2}$, irrespective of whether these paths were adopted or discarded. In Lemma 4 and Lemma 5 we show how these paths can be determined. This allows us to infer either the existence of a dispute wheel in Theorem 4, or the occurrence of a transient route flap in Theorem 5. Finally, we outline how y_0 could theoretically infer the existence (or lack thereof) of dispute wheels.

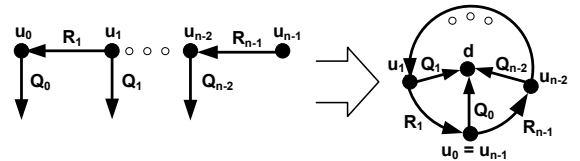


Fig. 15. If $u_0 = u_{n-1}$ and $Q_0 \prec R_{n-1} Q_{n-2}$ then a causation fence is a dispute wheel.

Lemma 3 (Fence-Wheel Relationship). *A causation fence $F = \{\mathcal{N}, \mathcal{R}, \mathcal{Q}\}$ of a DPR instance $D = (G, \succ)$, induced by a causation cycle of size k , where the first and last pivot nodes are the same, $u_0 = u_{n-1}$, is a dispute wheel if $Q_0 \prec R_{n-1} Q_{n-2}$.*

Proof. A sample causation fence is outlined in Figure 15. Pivot node u_0 has a spoke path Q_0 but not a rim path while

pivot node u_{n-1} has a rim path R_{n-1} but not a spoke path. A dispute wheel W can be constructed from F as shown by removing pivot node u_0 and setting $Q_{n-1} = Q_0$. ■

Lemma 4. *Given a causation fence $F = (\mathcal{N}, \mathcal{R}, \mathcal{Q})$ of a DPR instance $D = (G, \succ)$, induced by a causation cycle of size k , where the first pivot node in F is u_0 , $Q_0 = \pi(u_0, t + a)$ for some time offset $a \in \{0, 1\}$. If u_0 is part of an adopting subchain then $a = 1$. Otherwise, $a = 0$.*

Proof. The offset a simply determines whether the path Q_0 of node u_0 is the current path $\pi(u_0, t)$ or the new path $\pi(u_0, t + 1)$. As shown in Figure 15, node u_0 only has a spoke path Q_0 . If u_0 is part of an adopting subchain then subsequent nodes along the subchain are adopting a new path via u_0 . This implies that Q_0 must have become available and hence $Q_0 = \pi(u_0, t + 1)$ where $a = 1$. If, on the other hand, u_0 is part of a discarding subchain then subsequent nodes along the subchain are discarding the path they were initially using via u_0 . This implies that Q_0 must have been discarded and hence $Q_0 = \pi(u_0, t)$ where $a = 0$. ■

Lemma 5. *Given a causation fence $F = (\mathcal{N}, \mathcal{R}, \mathcal{Q})$ of a DPR instance $D = (G, \succ)$, induced by a causation cycle of size k , where the last pivot node in F is u_{n-1} , $R_{n-1}Q_{n-2} = \pi(u_{n-1}, t + k + b)$ for some time offset $b \in \{0, 1\}$. If u_{n-1} performed a StepDown then $b = 0$. Otherwise, $b = 1$.*

Proof. The offset b simply determines whether the path $R_{n-1}Q_{n-2}$ of node u_{n-1} is the current path $\pi(u_{n-1}, t + k)$ or the new path $\pi(u_{n-1}, t + k + 1)$. The offset b simply determines whether u_{n-1} should consider the current path $\pi(u_{n-1}, t + k)$ or the new path $\pi(u_{n-1}, t + k + 1)$. As shown in Figure 15, pivot node u_{n-1} only has path $R_{n-1}Q_{n-2}$. If pivot node u_{n-1} performed a StepDown then it is part of a discarding subchain where it discards path $R_{n-1}Q_{n-2}$. Hence, $R_{n-1}Q_{n-2} = \pi(u_{n-1}, t + k)$ where $b = 0$. Conversely, if u_{n-1} performed a StepUp or StepSame then it is part of an adopting subchain where it adopts path $R_{n-1}Q_{n-2}$. Hence, $R_{n-1}Q_{n-2} = \pi(u_{n-1}, t + k + 1)$ where $b = 1$. ■

Theorem 4 (Dispute Wheel Inference). *Given a causation cycle Y , such that $Y = \langle y_0 y_1 \dots y_k \rangle^t$ where $y_0 = y_k$, there exists time offsets $a \in \{0, 1\}$ and $b \in \{0, 1\}$ such that if $\pi(y_0, t + a) \prec \pi(y_k, t + k + b)$ then a dispute wheel exists around Y .*

Proof. Let F be the causation fence induced by Y . Using Lemma 4 we can determine time offset a and hence path Q_0 . Similarly, using lemma 5 we can determine time offset b and hence path $R_{n-1}Q_{n-2}$. From Lemma 3 we know that if the condition $Q_0 \prec R_{n-1}Q_{n-2}$ is satisfied then the causation fence F is a dispute wheel. Hence, the existence (or lack thereof) of a dispute wheel can be inferred. ■

Theorem 5 (Route Flap Inference). *Given a causation cycle Y , such that $Y = \langle y_0 y_1 \dots y_k \rangle^t$ where $y_0 = y_k$, if no*

dispute wheel exists then y_k received a transient route flap during the causation cycle.

Proof. From Theorem 4, there exists time offsets $a \in \{0, 1\}$ and $b \in \{0, 1\}$ such that the condition $\pi(y_0, t + a) \succ \pi(y_k, t + k + b)$ holds, otherwise a dispute wheel must exist. Thus path $\pi(y_0, t + a)$ had to be withdrawn by y_0 's next-hop neighbor during the causation cycle to force y_k to use the new, less preferred, path $\pi(y_k, t + k + b)$. Otherwise y_k would not have changed its path $\pi(y_0, t + a)$. This would imply that $\pi(y_0, t + a) = \pi(y_k, t + k + b)$ which is a contradiction. ■

Remark 3. *A causation cycle Y is triggered by one and only one event. An event could be a change in a link's availability causing a node to adopt or discard a particular path. If multiple events occur, their effects would be propagated along separate causation chains.*

If node y_0 observes causation cycle Y , to infer the existence of a dispute wheel, node y_0 must:

- Compute time offset a to determine path Q_0
- Compute time offset b to determine path $R_{n-1}Q_{n-2}$

The computation of offset b depends only on the action of y_k at time $t + k$ (Lemma 5). The computation of offset a is dependent on whether y_0 is a part of an adopting or a discarding subchain (Lemma 4). Let y_i be the first node in Y after y_0 whose action is not a StepSame. If the action of y_i is a StepUp then y_0 is part of an adopting subchain. Otherwise, y_0 is part of a discarding subchain.

Thus, given the type of subchain that y_0 belongs to, the dispute wheel inference problem can be solved. The solution is indeed local but cannot be performed independently—it requires the cooperation of the first node along Y that performed a StepUp or StepDown action.

V. SAFETYPULSE

Dispute wheels may result in protocol divergence. Permanent divergence due to dispute wheels, however, has not been observed in practice. Nevertheless, the detection of dispute wheels is of practical value to system administrators. By their fundamental structure, dispute wheels represent cyclic policy conflicts, which break from the traditional tiered architecture of the Internet [4], and could potentially lead to unbounded dynamics. SAFETYPULSE is a distributed algorithm to detect dispute wheels. Once dispute wheels are detected, they can be reported to administrators for further analysis.

A. Overview of Algorithm

SAFETYPULSE piggybacks *messages* alongside route updates. One possible implementation of SAFETYPULSE on BGP would be to use message options. Each node places a child *token* in this message. As a node receives a route update with this message, it chooses a new path and broadcasts a new message alongside its own route update. SAFETYPULSE essentially sends messages between nodes along causation chains.

If a node y receives a message from a neighbor which has y 's token, then it can be inferred that y has been involved in a causation cycle. Assume that node y sent out a token at time t_{out} and received the token back at time t_{in} . A dispute wheel can be detected by comparing the relative ranking of y 's realized paths around these times. Using Theorem 4, it can be inferred that a dispute wheel exists if for two given time offsets $a \in \{0, 1\}$ and $b \in \{0, 1\}$:

$$\pi(y, t_{\text{out}} + a) \prec \pi(y, t_{\text{in}} + b)$$

Generally speaking, this means that if node y had a more preferred route around the time when it received the token (at time $t_{\text{in}} + b$) than around the time when it sent out the token (at time $t_{\text{out}} + a$), then a dispute wheel exists.

The time offsets a and b represent whether the paths used are the ones adopted or discarded at times t_{out} and t_{in} , respectively. Time offset a is determined by the structure of the causation cycle. According to Lemma 4, it depends on whether y is part of an adopting or a discarding subchain. As we will see, time offset a can be computed by a third party node on the causation cycle. Time offset b , on the other hand, is determined by node y 's action at time t_{in} . According to Lemma 5, if y performed a StepDown then $b = 0$. Otherwise, $b = 1$.

The information in the token received by node y is enough for y to recover paths $\pi(y, t_{\text{out}} + a)$ and $\pi(y, t_{\text{in}} + b)$ for the comparison. We describe the SAFETYPULSE algorithm in three sections as shown in Figure 16.

- 1) Sending out token with ProcessNode()
- 2) Computing time offset with SetTimeOffset()
- 3) Receiving token with DetectDisputeWheel()

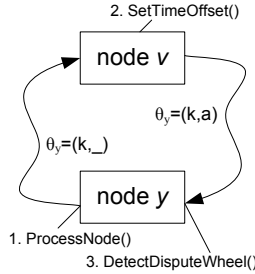


Fig. 16. Overview of SAFETYPULSE algorithm.

B. Sending the Token

We define $M(y, t)$ to be the SAFETYPULSE message that node y sends out alongside its route update at time t . In general, if node y changes its assigned path at time t then it has performed an action, switching from path $\pi(y, t)$ to path $\pi(y, t+1)$. Every time y performs an action, it stores the paths associated with its action, $\pi(y, t)$ and $\pi(y, t+1)$, in a hashtable using a newly generated key k . The token to be sent out is $\theta_y = (k, _)$, where k is the key identifying the action performed and $_$ is an empty slot in which the time offset a will be placed by another node. The new message $M(y, t+1)$ to be sent out alongside a route update at time $t+1$ following an action performed by y at time t must contain the following:

```

1: function PROCESSNODE( $y, t$ )
2:   Best( $y, t$ )  $\leftarrow$  max $_{\succ}$  Choices( $y, t$ )
3:    $\pi(y, t+1) \leftarrow$  Best( $y, t$ )
4:    $\theta_y \leftarrow \emptyset$ 
5:   if  $\pi(y, t) \neq \pi(y, t+1)$  then
6:      $k \leftarrow$  new key
7:     Store( $k, (\pi(y, t), \pi(y, t+1))$ )
8:      $\theta_y \leftarrow (k, \_)$ 
9:      $M(y, t+1) \leftarrow M(\text{Cause}(y, t), t) + \theta_y$ 

```

Fig. 17. SAFETYPULSE token creation and action storage.

- the message received initially from the node that caused the action
- node y 's new token θ_y

More formally, if $\pi(y, t) \neq \pi(y, t+1)$ then:

$$M(y, t+1) \leftarrow M(\text{Cause}(y, t), t) + \theta_y$$

Messages are propagated along causation chains where each node along the chain appends its token to the received message that triggered an action and sends out a new message. The algorithm for sending the token is outlined in Figure 17.

C. Receiving the Token

When a node y receives a token θ_y that it has previously created in a routing update message, it checks to see if a dispute wheel has been created. The contents of the token are $\theta_y = (k, a)$ where k represents the key to lookup the action and a represents whether to use the discarded or the adopted path of the action. Note that a will be created by a third party node as described in the next section. Here, we assume that a has been set appropriately and $\pi(y, t_{\text{out}} + a)$ can be determined.

Next, using Lemma 5 we determine the second time offset b to find $\pi(y, t_{\text{in}} + b)$. According to Theorem 4, if:

$$\pi(y_0, t+a) \prec \pi(y_k, t+k+b)$$

then a dispute wheel exists around Y . Using this information, the dispute wheel detection algorithm can be constructed as shown in Figure 18.

D. Computing time offset

The remaining part is to determine time offset a . In Lemma 4, we showed that the value of a is dependent on the type of subchain that y belongs to. This can be determined by the action of the next node, v , along the causation cycle. If v performed a StepUp then y is in an adopting subchain. If v performed a StepDown then y is in a discarding subchain. If v performed a StepSame, then y 's subchain type is decided by v 's next node in the causation chain. Thus a node can fill in the time offsets of the uncategorized nodes based on the action performed. If a node performs a StepDown or StepUp action, it can fill the time offsets with 0 or 1, respectively. The algorithm in Figure 19 shows how third-party nodes can fill in the time offset a .

```

1: function DETECTDISPUTEWHEEL( $y, t$ )
2:   if  $\theta_y \in M(\text{Cause}(y, t), t)$  then
3:      $\theta_y = (k, a)$ 
4:      $(P_1, P_2) \leftarrow \text{Lookup}(k)$ 
5:     if  $a = 0$  then
6:        $P_{\text{test}} \leftarrow P_1$ 
7:     else
8:        $P_{\text{test}} \leftarrow P_2$ 
9:     if  $\text{Action}(y, t) = \text{StepDown}$  then
10:       $b \leftarrow 0$ 
11:    else
12:       $b \leftarrow 1$ 
13:    if  $P_{\text{test}} \prec \pi(y, t + b)$  then
14:       $\text{ReportDisputeWheel}(P_{\text{test}}, \pi(y, t + b))$ 

```

Fig. 18. SAFETYPULSE token receipt and dispute wheel detection.

```

1: function SETTIMEOFFSET( $y, t$ )
2:   for all unclassified  $\theta_v = (k, \_)$   $\in M(y, t + 1)$  do
3:     if  $\text{Action}(y, t) = \text{StepUp}$  then
4:        $\theta_v \leftarrow (k, 1)$ 
5:     else if  $\text{Action}(y, t) = \text{StepDown}$  then
6:        $\theta_v \leftarrow (k, 0)$ 

```

Fig. 19. SAFETYPULSE time offset computation.

E. SAFETYPULSE Algorithm

The overall SAFETYPULSE algorithm is outlined in Figure 20. Each node y at time t simply executes the three algorithms described above.

```

1: function SAFETYPULSE( $y, t$ )
2:    $\text{ProcessNode}(y, t)$ 
3:    $\text{SetTimeOffset}(y, t)$ 
4:    $\text{DetectDisputeWheel}(y, t)$ 

```

Fig. 20. SAFETYPULSE algorithm.

F. Space Requirements

The token sent by every node y has two parts, the key k and the time offset a . The key needs to index an action stored locally at node y . If node y is expected to switch between 2^i paths, then the size of k only needs to be i . The time offset a can be represented by two bits, b_0b_1 . The first bit b_0 is initially set to 0, indicating that a has not been set. The second bit b_1 is set to a random bit. Once a third party node v wants to set a , it manipulates $a = b_0b_1$ as follows: set b_0 to 1 and flip b_1 if the action of v is a StepUp. When node y receives $a = b_0b_1$ it checks if b_0 is set and if b_1 is flipped (compared to a locally stored version of a). If so, then node y knows that it should check against the adopted path. Otherwise, node y checks against the discarded path. Thus the overhead added by each node is negligible.

G. Characteristics of SAFETYPULSE

SAFETYPULSE distinguishes itself by having a unique set of characteristics:

- **Provably Correct.** SAFETYPULSE is provably correct with any dynamic network since it is based on a comprehensive theory of policy routing dynamics.
- **Privacy Guarantees.** Path preferences are not revealed between neighbors.
- **Efficient Space.** Each node only appends a small token of space complexity $O(1)$ to a route it sees.
- **Policy Freedom.** Since SAFETYPULSE is a dynamic detection algorithm, it does not require any restrictions on routing policies to be imposed. Also, it enables results even in the case of piecemeal adherence to the protocol—only ASes along the causation chains / cycles to be diagnosed need to adopt the protocol.

VI. CONCLUSIONS

In this paper, we extended the DPR model, initially introduced in [18], to further characterize the dynamics of interdomain routing in the presence of policy conflicts. We introduced policy digraphs which allowed us to identify a more efficient detector for policy conflicts. Leveraging results derived from our extensions to DPR, we introduced SAFETYPULSE—a distributed policy conflict detection algorithm that excels in all criteria parameters: time / space overhead, privacy, soundness, and ease of adoption.

REFERENCES

- [1] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed Internet Routing Convergence," *IEEE/ACM Trans. Netw.*, vol. 9, pp. 293–306, June 2001.
- [2] K. Varadhan, R. Govindan, and D. Estrin, "Persistent Route Oscillations in Inter-domain Routing," *Comput. Netw.*, 1996.
- [3] T. Griffin, F. Shepherd, and G. Wilfong, "The Stable Paths Problem and Interdomain Routing," *IEEE/ACM Trans. Netw.*, vol. 10, no. 2, pp. 232–243, Apr 2002.
- [4] L. Gao and J. Rexford, "Stable Internet Routing Without Global Coordination," *IEEE/ACM Trans. Netw.*, vol. 9, no. 6, pp. 681–692, 2001.
- [5] N. Feamster, R. Johari, and H. Balakrishnan, "Implications of Autonomy for the Expressiveness of Policy Routing," in *SIGCOMM*, 2005, pp. 25–36.
- [6] F. Wang, L. Gao, and J. Qiu, "On Understanding of Transient Interdomain Routing Failures," in *ICNP*, 2005.
- [7] D. Obradovic, "Real-time Model and Convergence Time of BGP," in *INFOCOM*, 2002.
- [8] D. Pei, B. Zhang, D. Massey, and L. Zhang, "An analysis of Convergence Delay in Path Vector Routing Protocols," *Comput. Netw.*, vol. 50, no. 3, pp. 398–421, 2006.
- [9] J. Cobb and R. Musunuri, "Enforcing Convergence in Inter-Domain Routing," *GLOBECOM*, vol. 3, November 2004.
- [10] C. T. Ee and V. Ramachandran and B.G. Chun and K. Lakshminarayanan and S. Shenker, "Resolving Inter-Domain Policy Disputes," in *SIGCOMM*, 2007, pp. 157–168.
- [11] T. Griffin and G. T. Wilfong, "A Safe Path Vector Protocol," in *INFOCOM*, 2000, pp. 490–499.
- [12] S. Yilmaz and I. Matta, "An Adaptive Management Approach to Resolving Policy Conflicts," in *IFIP Networking 2007*, Atlanta, Georgia, May 2007.
- [13] Ahronovitz, J.-C. Knig, and C. Saad, "A Distributed Method for Dynamic Resolution of BGP Oscillations," in *IPDPS*, 2006.
- [14] T. G. Griffin and J. L. Sobrinho, "Metarouting," in *SIGCOMM*, 2005, pp. 1–12.

- [15] R. Govindan, C. Alaettinoglu, G. Eddy, D. Kessens, S. Kumar, and W. san Lee, "An Architecture for Stable, Analyzable Internet Routing," *IEEE Network*, vol. 13, pp. 29–35, 1999.
- [16] N. Feamster and H. Balakrishnan, "Verifying the Correctness of Wide-Area Internet Routing," MIT, Tech. Rep., May 2004.
- [17] A. Feldmann, O. Maennel, Z. Mao, A. Berger, and B. Maggs, "Locating Internet Routing Instabilities," in *SIGCOMM*, September 2004.
- [18] S. Epstein, K. Mattar, and I. Matta, "Principles of Safe Policy Routing Dynamics," in *ICNP*, 2009.
- [19] —, "Principles of Safe Policy Routing Dynamics," Computer Science Department, Boston University, Tech. Rep. BUCS-TR-2009-13, April 2009.

APPENDIX A ASYNCHRONICITY WITH DPR

The key idea is that modeling link delays is sufficient to capture asynchronicity without adding any new constructs to a regular DPR instance $D = (\succeq, G)$. At any time t , each link $(u, v)^t \in E$ admits a variable time delay between 1 and a finite upper limit M . This delay is specified by the function $L(u, v, t)$ which outputs an integer in $[1, M]$. The time delays are considered ordered such that $L(u, v, t) - L(u, v, t+k) < k$. Thus $L(u, v, 4) = 100$ and $L(u, v, 5) = 2$ are not allowed since v would get u 's path at time 5 before receiving u 's path at time 4. From DPR instance D and delay function L , a new asynchronous DPR instance $D' = (\succeq', G')$ can be constructed to simulate D with time delays.

For every two nodes u, v in D , there are $M - 1$ transit nodes added to D' such that $u, v \in V \Rightarrow x_i^{uv} \in V'$ for $2 \leq i \leq M$. These transit nodes represent the "communication wire" between every two nodes. The dynamic nature of the links will be used to control the length of this communication wire. If $L(u, v, t) = 5$, then a path of length 5 between u and v through the transit nodes will appear at time t . An example of a delay 3 between nodes u and v can be seen in Figure 21.

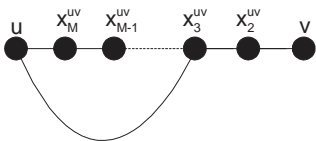


Fig. 21. Transit nodes simulating a delay of $L(u, v, t) = 3$.

The path preferences of $D' = (\succeq', G')$ discount the presence of transit nodes in paths. Let the operation `RemoveTransit` remove all transit nodes. This operation allows the asynchronous path preferences to be derived from the original synchronous path preferences. Thus for all non-transit nodes $u \in V'$: $P_1^u \succeq^t P_2^u$ iff `RemoveTransit`(P_1^u) \succeq^t `RemoveTransit`(P_2^u). Each transit node x_i^{uv} prefers a path through its source node u over its transit neighbor toward the source x_{i+1}^{uv} . Paths containing sequences in the opposite direction (from x_i^{uv} to x_{i-1}^{uv}) are forbidden. Finally, in terms of causation chains, given delay $L(u, v, t) = 3$, the causation chain $\langle u \ v \rangle^t$ in D corresponds to the causation chain $\langle u \ x_3^{uv} \ x_2^{uv} \ v \rangle^t$ in D' .