

2009-01-30

Foundational Theory for Understanding Policy Routing Dynamics

Mattar, Karim; Epstein, Sam; Matta, Ibrahim. "Foundational Theory for Understanding Policy Routing Dynamics", Technical Report BUCS-TR-2009-001, Computer Science Department, Boston University, January 30, 2009. [Available from: <http://hdl.handle.net/2144/1724>]

<https://hdl.handle.net/2144/1724>

"Downloaded from OpenBU. Boston University's institutional repository."

Foundational Theory for Understanding Policy Routing Dynamics

Karim Mattar Samuel Epstein Ibrahim Matta

Computer Science, Boston University
Email: {kmattar, samepst, matta}@cs.bu.edu

Technical Report BUCS-TR-2009-001
January 30, 2009

ABSTRACT

In this paper we introduce a theory of policy routing dynamics based on fundamental axioms of routing update mechanisms. We develop a *dynamic policy routing* model (DPR) that extends the static formalism of the *stable paths problem* (introduced by Griffin et al.) with discrete synchronous time. DPR captures the propagation of path changes in any dynamic network irrespective of its time-varying topology. We introduce several novel structures such as causation chains, dispute fences and policy digraphs that model different aspects of routing dynamics and provide insight into how these dynamics manifest in a network.

We exercise the practicality of the theoretical foundation provided by DPR with two fundamental problems: *routing dynamics minimization* and *policy conflict detection*. The dynamics minimization problem utilizes policy digraphs, that capture the dependencies in routing policies irrespective of underlying topology dynamics, to solve a graph optimization problem. This optimization problem explicitly minimizes the number of routing update messages in a dynamic network by optimally changing the path preferences of a minimal subset of nodes.

The conflict detection problem, on the other hand, utilizes a theoretical result of DPR where the root cause of a causation cycle (*i.e.*, cycle of routing update messages) can be precisely inferred as either a transient route flap or a dispute wheel (*i.e.*, policy conflict). Using this result we develop SafetyPulse, a token-based distributed algorithm to detect policy conflicts in a dynamic network. SafetyPulse is privacy preserving, computationally efficient, and provably correct.

1. INTRODUCTION

The Internet consists of around 17,000 autonomous systems (ASes). Each AS represents an Internet Service Provider (ISP), company or university, that is managed independently. BGP is the defacto routing protocol 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 available 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 inter-domain routers may take *tens of minutes* to reach a consistent view of the network topology after a fault [1]. In addition, no guarantees are made by BGP regarding convergence (*i.e.*, ASes may adopt and discard paths indefinitely [2]).

Griffin *et al.* introduced the *stable paths problem* (SPP) in [3], a formalism to reason about the steady-state behavior of BGP under a static setting (static topology and routing policies). Their solution concept is the stable assignment, where every AS is assigned its most preferred path out of its available choices. They identified policy conflicts known as *dispute wheels* as the necessary condition for the lack of a stable assignment.

Existing work based on SPP's formalism or work that operates exclusively under the static setting assumption has several limitations, namely:

- SPP does not give any insight into the transient behavior of networks. This makes reasoning about the dynamic behavior due to misconfigured or malicious routing policies, or networks with sporadic or time-varying connectivity extremely difficult.
- Algorithms that attempt to detect and resolve policy conflicts are either ad hoc or cumbersome. For example, counting [4] and other token-based [5] approaches are heuristics and their correctness cannot be guaranteed. History-based solutions [6], on the other hand, incur a huge message exchange overhead and do not allow the participating ASes to maintain any privacy.
- Next-generation routing architectures that guarantee convergence to a stable assignment, such as Metarouting [7] or Gao-Rexford [8] are heavy handed, requiring every node to comply. They

do not provide results for partial adherence. Such schemes also limit the freedom of AS administrators to choose their routing policies.

Despite its limitations, the static setting assumption is generally used because there is a common consensus that explicitly modeling routing dynamics is difficult or intractable. For example, statements such as “Analyzing the convergence properties of a dynamic routing protocol like BGP is very difficult” [9] appear frequently in the literature.

In this paper we introduce foundational theory that provides insight into policy routing dynamics and instability. We develop a dynamic policy routing model, DPR, that extends SPP with *discrete synchronous time*. Whereas SPP is concerned with stable assignments of paths in a static setting, DPR focuses on modeling the propagation of path changes in a dynamic setting (networks with time-varying topologies)¹. Using DPR we make the following contributions:

- We formalize the dynamics minimization problem using policy digraphs, which capture the dependencies in routing policies irrespective of underlying topology dynamics, to solve a graph optimization problem. This optimization problem explicitly minimizes the potential number of routing update messages in a dynamic network. This is done by optimally changing the path preferences of a minimal subset of nodes.
- We develop SafetyPulse, a token-based distributed algorithm to detect policy conflicts in a dynamic network. SafetyPulse has several novel characteristics, namely, it is privacy preserving, computationally efficient, and provably correct.

The rest of the paper is organized as follows: Section 2 presents SPP using a sample policy routing instance that will be used throughout the paper. Section 3 provides a brief overview of DPR and its applications. Section 4 formalizes the time-varying DPR structures that model the propagation of path changes in a dynamic network. Section 5 formalizes the time-invariant DPR structures that are derived from analyzing the transient routing dynamics. Section 6 presents policy digraphs that form the basis of the routing dynamics minimization problem discussed in Section 7. Sections 8 and 9 outline the theoretical results and algorithms, respectively, to detect policy conflicts in a dynamic network. Section 10 discusses the related work. Finally, Section 11 presents our conclusions and future work.

¹Changes in network topologies are assumed to be due to link failures.

2. STABLE PATHS PROBLEM

Griffin *et al.* introduced the *stable paths problem* (SPP) in [3] as a tool to reason about policy routing. SPP provides a steady-state analysis to determine if a particular instance of policy routing has a stable state. In SPP, the routing network is represented by a graph: $G = (V, E)$ where each AS is represented by a node $v \in V$. If two nodes u and v are connected then $(u, v) \in E$.

Paths in G are represented by sequences of the form:

$$P = \langle u_0 u_1 \dots u_n d \rangle$$

where d is a distinguished destination node. At node u_0 , the next-hop of P is denoted by:

$$u_1 = \text{NextHop}(P)$$

The empty path is represented by: $\langle \rangle$. The concatenation of a path P with node u is represented by: $\langle u P \rangle$. The set of paths originating from a particular node u can be denoted as \mathcal{P} .

Each node wishes to obtain a path to d . Each node u has a set preference over the paths, represented by \succ_u . This preference forms a total order over $\mathcal{P} \cup \langle \rangle$. For ease of notation, we represent the combined path preferences of all nodes with the partial order \succ . If a path P is forbidden then $\langle \rangle \succ P$. All paths with repeating nodes are forbidden.

An instance of SPP is comprised of the network and the path preferences of each of its nodes: (G, \succ) .

2.1 Stable Assignment

In policy routing each node u broadcasts to its neighbors its current path P to the destination node d . Each node chooses its most preferred path over the set provided by its neighbors. The goal of SPP is to find a *stable assignment*, which is a directed tree, confluent at d . Each node u in this stable assignment is satisfied if the path from u to d is preferred over paths through its neighboring nodes. If each node is satisfied, then the tree is stable (*i.e.*, will not change).

We represent a path assignment with the function π that maps each node to a particular path. The paths available to a particular node u can be represented as:

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

A node’s best path is the most preferred path among the paths available to it:

$$\text{Best}(u) = \max_{\succ} \text{Choices}(u)$$

A path assignment π is stable if each node is assigned its most preferred path out of its choices:

$$\text{Stable}(\pi) \Leftrightarrow \text{Best}(u) = \pi(u) \text{ for all } u \in V$$

Griffin *et al.* showed in [3] that it is NP-Complete to determine whether a stable assignment exists.

2.2 Dispute Wheels

Griffin *et al.* also introduced *dispute wheels* in [3], whose existence is a necessary condition for an SPP instance to not have a stable assignment. A dispute wheel, as shown in Figure 1, represents a cyclical set of path preferences. A dispute wheel W is formally 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 .
- Each node u_i prefers a path through its rim and neighbor's spoke path over its own spoke path:

$$R_i Q_{i-1} \succ Q_i$$

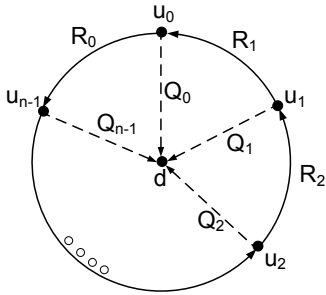


Figure 1: Dispute wheel.

2.3 Sample SPP Instance

A sample SPP instance can be seen in Figure 2 where the destination d is node 0. Each node has a path preference list consisting of two paths where the most preferred path is the topmost path. For example, node 1 prefers path $\langle 1430 \rangle$ over the direct path $\langle 10 \rangle$.

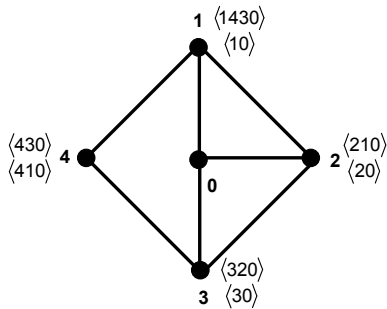


Figure 2: SPP Instance BAD GADGET.

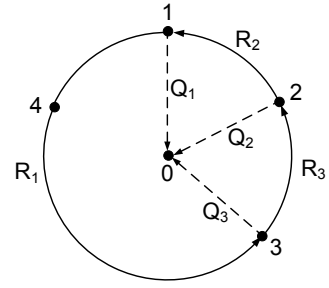


Figure 3: Dispute wheel in BAD GADGET.

This SPP instance is commonly called BAD GADGET and is known to have at least one dispute wheel as shown in Figure 3. We will use this SPP instance as a running example throughout this paper.

3. DYNAMIC POLICY ROUTING MODEL

The *dynamic policy routing* model (DPR) captures the dynamics and potential instability of policy routing. Whereas the focus of SPP is the stable assignment of paths in a static network, DPR concerns itself with the propagation of path changes over time in a dynamic network. DPR extends SPP with discrete synchronous time. This is analogous to modeling the transient dynamics of a system as opposed to its steady-state behavior.

3.1 Overview

We provide here an overview of DPR by analyzing the routing dynamics of a sample SPP instance, BAD GADGET. We do not provide any formal definitions and focus only on presenting the core applications of DPR, namely, *routing dynamics minimization* and *policy conflict detection* from an intuitive (and informal) perspective.

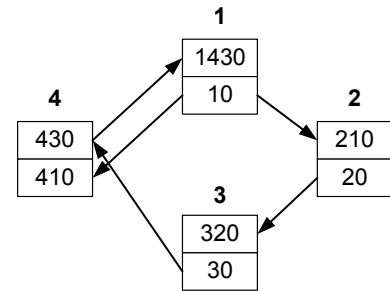


Figure 4: Policy digraph of BAD GADGET.

Figure 4 outlines the *policy digraph* of BAD GADGET. While the definition of policy digraphs is provided in Section 6, for ease of exposition we present here a simple representation. Each node in BAD GADGET is represented in the policy digraph by a “ladder” where

each step in the ladder denotes a path from the node’s path preference list. The node’s most preferred path is at the top of the ladder. Any two paths in different ladders, for example paths $\langle 10 \rangle$ and $\langle 210 \rangle$ are connected if one is a subpath of the other. Since $\langle 10 \rangle$ is a subpath of $\langle 210 \rangle$ there is an edge connecting $\langle 10 \rangle$ to $\langle 210 \rangle$. We will 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 up the chutes (*i.e.*, using the edges connecting different ladders).

Let us consider two sample walks starting from path $\langle 210 \rangle$:

Walk 1: $\langle 210 \rangle \langle 20 \rangle \langle 320 \rangle \langle 30 \rangle \langle 430 \rangle \langle 1430 \rangle \langle 10 \rangle$
 Walk 2: $\langle 210 \rangle \langle 20 \rangle \langle 320 \rangle \langle 30 \rangle \langle 430 \rangle \langle 410 \rangle$

Walks in this structure capture the routing dynamics of BAD GADGET. By routing dynamics, we mean how path changes propagate in the network or more specifically how paths could be *adopted* and *discarded*. For example, if path $\langle 20 \rangle$ is adopted then path $\langle 320 \rangle$ is also adopted since it is node 3’s most preferred path. Such dependencies are captured via 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, which results in path $\langle 320 \rangle$ getting discarded by node 3 (a dependency that is captured by the subpath edge between $\langle 20 \rangle$ and $\langle 320 \rangle$).

The policy digraph structure provides valuable insight into the routing dynamics. A path in the graph 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 change (*e.g.*, a link failure). This forms the basis of our routing dynamics minimization problem where we use policy digraphs to solve a graph optimization problem. That optimization problem minimizes some function of the possible path lengths by optimally changing the path preferences of a minimal subset of nodes.

On the other hand, policy conflicts (or dispute wheels) are represented as cycles in the graph where the first and last paths are the same. For example, consider the following cycle:

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

From Figure 4, one can see that if node 1 instigates a change in the network and the route updates propagate in such a way that a new, more preferred, route eventually reaches node 1 then a dispute wheel exists. If the cycle starts at path $\langle 10 \rangle$, for example, and ends at path $\langle 1430 \rangle$, which is more preferred, clearly the cycle can repeat indefinitely as the walk can go down the ladder at node 1 and follow the same cycle all over again. This

assumes that the path preferences do not change over time. This observation forms the basis of our policy conflict detection algorithm.

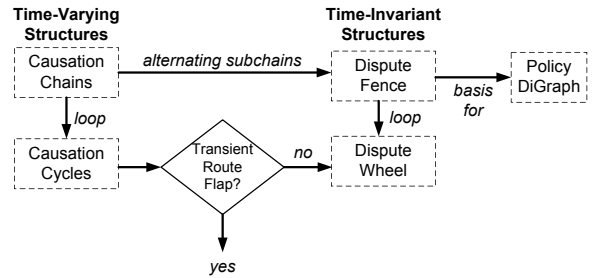


Figure 5: Overview of DPR.

We provide a brief overview of the theory of DPR. The main components of DPR are outlined in Figure 5. The central notions in DPR are that of *action* and *causation* where one node causes another to perform an action. 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 explicitly models these two events to construct a *causation chain* over time which is a sequence of nodes where each node causes an action in its successor on the chain. Thus, causation chains capture the propagation of path changes in a network.

In Section 5, we prove that causation chains are not random sequences of nodes (and their associated actions). Instead, the routing dynamics (*i.e.*, how routing path changes propagate) manifest in a manner that can be precisely defined. We prove that any causation chain is composed of alternating *adopting* and *discarding* subchains. The head node of an adopting subchain makes a path available that all subsequent nodes adopt. Conversely, the head node of a discarding subchain discards a path that forces all subsequent nodes to choose alternate paths. Using the alternating subchains property we introduce the time-invariant structure *dispute fences* that only represent the head and tail nodes of the alternating subchains. Dispute fences capture the key path changes along a causation chain. Capturing the transient dynamics in terms of a time-invariant structure greatly simplifies the analysis of routing dynamics.

In Section 6, we introduce *policy digraphs* that capture the dependencies in routing policies irrespective of the topology dynamics. Nodes in the policy digraph represent realizable paths while edges represent different types of dependencies between these paths. We prove that causation chains and dispute wheels are paths and cycles in the policy digraph, respectively. Using policy digraphs we formalize the routing dynamics minimization problem (RDMP) in Section 7.

In Section 8, using dispute fences we prove that a causation cycle, representing a cycle of routing update

messages, is either due to a transient route flap or a dispute wheel and that the exact reason can be precisely inferred. This theoretical result forms the basis of SafetyPulse, a token-based distributed algorithm to detect policy conflicts in a dynamic network in Section 9.

3.2 Basic Notation

In DPR, time is represented by a non-negative, discrete index t such that: $t = [0, \infty)$. The network is represented by a time-dependent graph $G = (V, E)$ such that:

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

In DPR, a node's preferential ranking of paths is represented by the (strict) \succ operator. If u prefers P over Q , then: $P \succ Q$.

A DPR instance consists of a graph and a path preference set: $D = (G, \succ)$. At each time index t , every node u has a path to d , represented by:

$$P = \pi(u, t)$$

The available path choices of a node u at time t , via all possible neighbors v , are represented by $\text{Choices}(u, t)$ where:

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

The $\text{Best}(u, t)$ notation represents the current best path for u at time t :

$$\text{Best}(u, t) = \underset{\succ}{\max} \text{Choices}(u, t)$$

The states of nodes at each time t is their best path of the previous round. For all nodes $u \in V$:

$$\begin{aligned} \pi(u, 0) &= \langle \rangle \\ \pi(u, t) &= \text{Best}(u, t-1) \end{aligned}$$

The next-hop of u_0 's assigned path

$$\pi(u_0, t) = \langle u_0 \ u_1 \ \dots \ u_n \ d \rangle$$

is denoted by:

$$u_1 = \text{NextHop}(u_0, t) = \text{NextHop}(\pi(u_0, t))$$

A path P is *realized* for node u iff there exists a time t such that $\pi(u, t) = P$.

4. TIME-VARYING DPR STRUCTURES

We introduce the causation chain structure where routing dynamics are represented as a simple sequence of actions. Each node in the sequence instigates the action taken by the next node along the chain.

DPR reasons about routing dynamics using two constructions: actions and causation. Actions represent a change in a node's chosen path between two time steps. A node u performs an action at time t if:

$$\pi(u, t) \neq \pi(u, t+1)$$

DPR also includes the notion of *causation*. 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 and relative ranking of its new and old paths, as shown in Table 1. Consider the first row in Table 1, for example, where node u performs a StepUp action. This implies that it switches to a new path through a more preferred next-hop node v such that:

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

The cases of action and causation can be seen in Figure 6, using the BAD GADGET instance in Figure 2. At time instant $t = 1$, node 3 performs a StepDown action as it is forced to discard path $\langle 320 \rangle$ and adopt path $\langle 30 \rangle$ due to node 2 adopting path $\langle 210 \rangle$ at time $t = 0$.

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

$$\text{Cause}(y_i, t+i) = y_{i-1} \text{ for all } 0 < i \leq k$$

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 .

Causation chains represent the propagation of path changes in a dynamic network. An example causation chain in Figure 6 is: $\langle 2 \ 3 \ 4 \ 1 \rangle^0$.

DEFINITION 2 (CAUSATION CYCLES). A *causation cycle* is a causation chain with a repeated node: $Y = \langle y_0 \ y_1 \ \dots \ y_k \rangle^t$ where, $y_0 = y_k$.

An example causation cycle in Figure 6 is:

$$\langle 2 \ 3 \ 4 \ 1 \ 2 \rangle^0$$

Since the DPR instance represented in Figure 6 contains a dispute wheel, there can be an infinite number of causation cycles, assuming no changes in path preferences.

5. TIME-INVARIANT DPR STRUCTURES

In this section we present the time-invariant DPR structures, namely, dispute fences and dispute wheels. We focus on explaining the purpose and usefulness of the foundational theorems we present, as well as on providing sufficient intuition for all the formal proofs.

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 $t + 1$.
None	$v = \emptyset$	$\pi(u, t) = \pi(u, t + 1)$	Node u uses the same path at time t and $t + 1$

Table 1: Cases for action and causation.

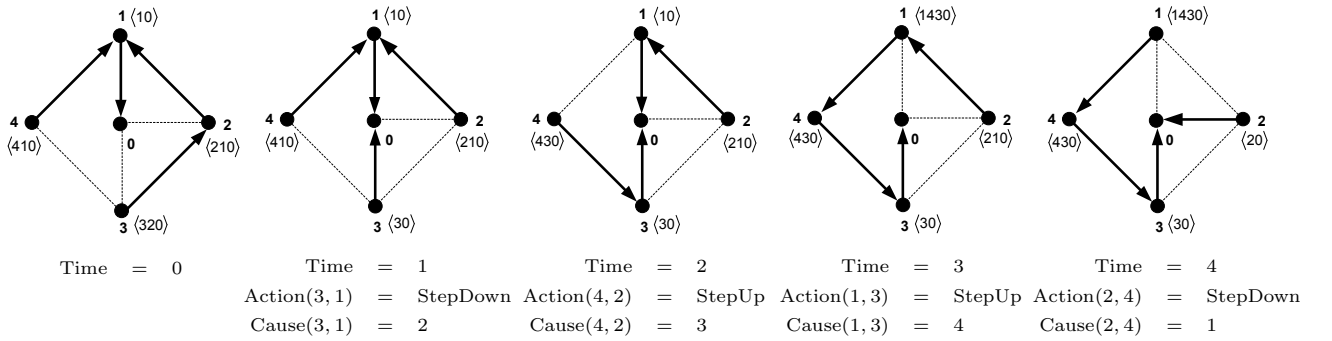


Figure 6: Sample actions and causation for BAD GADGET .

5.1 Causation Subchains

We first show that causation chains are not random sequences of nodes (and their associated actions). Instead, routing dynamics (*i.e.*, how routing changes propagate in a network) manifest in a manner that can be precisely defined. We show that causation chains can be broken into two alternating types of subchains: *adopting* and *discarding* subchains.

DEFINITION 3 (CAUSATION SUBCHAIN). A *causation subchain* consists of a series of consecutive nodes:

$$\langle y_i y_{i+1} \dots y_j \rangle^{t+i}$$

where the first node y_i is referred to as the *head node*. 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 causation chain and it takes i time steps to reach node y_i .

In an adopting subchain, the head node y_i makes a new path available that all subsequent nodes adopt. In Figure 7, 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.

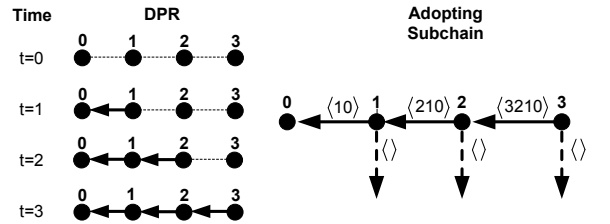


Figure 7: An example of an adopting subchain. Adopted/discarded paths are represented by solid/dotted arrows, respectively.

DEFINITION 4 (ADOPTING SUBCHAIN). We define an *adopting subchain* as a subchain of Y :

$$\langle y_i y_{i+1} \dots y_j \rangle^{t+i}$$

from y_i to y_j where $i < j$ with the following property:

$$\text{Action}(y_k) \neq \text{StepDown} \text{ for all } i < k \leq j$$

Since the action of all nodes after the head node y_i is either StepUp or StepSame, these nodes are adopting the path created by y_i . This is irrespective of y_i 's action.

In a discarding subchain, all nodes in the chain 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. In Figure 8, for example, node 1 discards path $\langle 10 \rangle$ which causes node 2 to discard path $\langle 210 \rangle$. This in turn causes node 3 to discard path $\langle 3210 \rangle$.

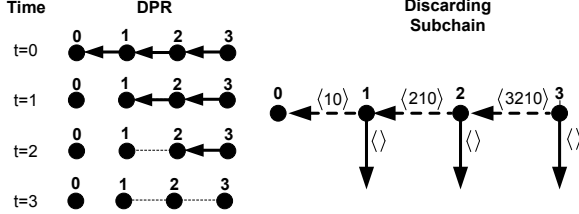


Figure 8: An example of a discarding subchain. Adopted/discarded paths are represented by solid/dotted arrows, respectively.

DEFINITION 5 (DISCARDING SUBCHAIN). We define a discarding subchain as a subchain of Y :

$$\langle y_i y_{i+1} \dots y_j \rangle^{t+i}$$

from y_i to y_j where $i < j$ with the following property:

$$\text{Action}(y_k) \neq \text{StepUp} \text{ for all } i < k \leq j$$

Since the action of all nodes after the head node y_i is either StepDown or StepSame, these nodes are discarding their current path through y_i . This is again irrespective of y_i 's action.

THEOREM 1. Every causation chain Y , such that $Y = \langle y_0 y_1 \dots y_k \rangle^t$ 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 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' that is being constructed. We construct either an adopting or a discarding subchain based on y_k 's action.

Construct Adopting Subchain: 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 that point Y' has been completely built, and is subtracted from Y (leaving y_j in

Y). We then recurse, starting with y_j again. Since y_j is a StepDown, the next subchain subsumed will be a discarding subchain.

Construct Discarding Subchain: If the action of y_k is StepDown, then Y' is a discarding subchain. Again 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 StepUp. At that point Y' has been completely built, and is subtracted from Y (leaving y_j in Y). We then recurse, starting with y_j again. Since y_j is a StepUp, the next subchain subsumed will be an adopting subchain.

For both adopting and discarding subchains, we continue recursing until we reach y_0 . At that point we add y_0 to the current Y' regardless of its action. From this construction we can see that every time an adopting subchain ends, a discarding subchain starts and every time a discarding subchain ends, an adopting subchain starts. Hence, the subchains must be alternating. \square

These results will serve as the basis for constructing the time-invariant structure dispute fences.

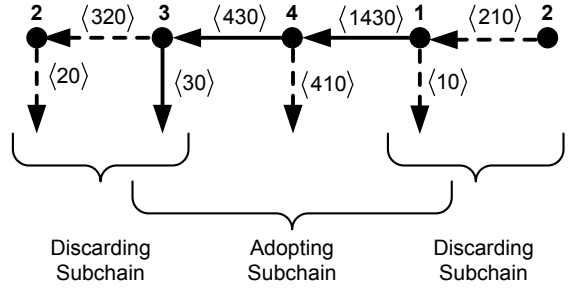


Figure 9: Alternating subchains of BAD GADGET

Figure 9 shows the alternating subchains given the actions and causations of BAD GADGET in Figure 6. Dotted arrows represent paths that were discarded in the causation chain whereas solid arrows represent paths that were adopted. Horizontal paths are more preferred than vertical paths.

5.2 Dispute Fences

The dispute 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 dispute fence concerns itself with are the adopted and discarded paths in the subchains.

DEFINITION 6 (DISPUTE FENCE). A dispute fence is formally defined by $F = (\mathcal{N}, \mathcal{R}, \mathcal{Q})$ where:

- \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 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 dispute fence can be seen as an open-ended dispute wheel, as shown in Figure 10, where the first and last pivot nodes are missing their (potential) rim and spoke paths, respectively.

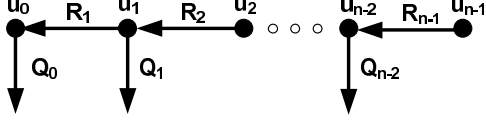


Figure 10: Dispute fence.

THEOREM 2. 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 dispute fence $F = (\mathcal{N}, \mathcal{R}, \mathcal{Q})$.

PROOF. Using theorem 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 dispute fence F . This creates n pivot nodes u_0, u_1, \dots, u_{n-1} . The first subchain Y^0 contributes two pivot nodes while all other subchains Y^r , $r > 0$, contribute one pivot node to F .

In general, 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 depending on whether such a pair of pivot nodes is part of an adopting or a discarding subchain.

Adopting Subchain: 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} .

Discarding Subchain: Similarly, if the pivot nodes are part of a discarding subchain then the first pivot node u_{i-1} is the head of the subchain. Pivot node u_{i-1} discards a path that forces all subsequent nodes along the subchain including u_i to discard their current path through u_{i-1} . Thus, during the course of routing, once

a discarding subchain is built, all nodes in the subchain were once on the rim path that is being discarded. This rim path once connected u_i to u_{i-1} .

Thus, in general the corresponding rim paths starting with pivot node y_s^r are the reverse ordering of the nodes in the subchains:

$$\langle y_s^r y_{s-1}^r \dots y_0^r \rangle$$

This order is reversed from Y^r because the causation chain propagates in the opposite direction of the paths being created. \square

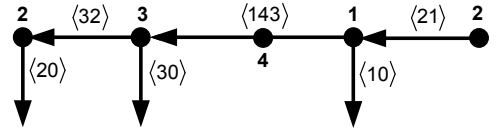


Figure 11: Dispute fence example.

Figure 11 shows the dispute fence induced by the causation chain in Figure 9.

5.3 Dispute Wheels

The dispute wheel is a fundamental structure that represents cyclic policy conflicts. Dispute wheels are also fundamental in DPR. 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. This novel result is of both theoretical and practical importance.

THEOREM 3. 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 12. We refer to $P(a, b)$ as the subpath of P starting with a and ending with b and $P(a)$ as the subpath of P starting with a .

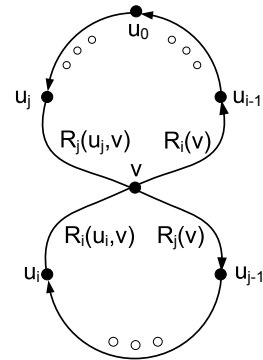


Figure 12: Non-proper dispute wheel.

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:

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

This results in the dispute wheel in Figure 13.

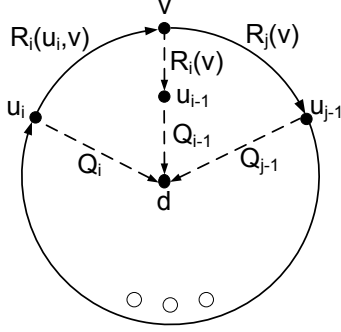


Figure 13: Smaller dispute wheel case 1.

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 14.

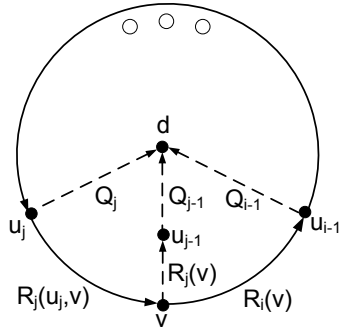


Figure 14: Smaller dispute wheel case 2.

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'' . This reasoning can only continue a finite number of iterations. Thus every dispute wheel W contains a proper dispute wheel. \square

6. ROUTING DYNAMICS: THEORY

In this section we introduce the necessary theory to reason about routing dynamics. We introduce *policy digraphs* which are directed graphs that have the properties outlined in Table 2. A policy digraph is similar to the *dispute digraph* introduced by Griffin *et al.* in [3], but has several novel characteristics, namely:

1. Has a simple definition
2. Is independent of the network topology dynamics
3. Represents several DPR structures simultaneously
4. Is less dense as the number of edges is on the order of the number of nodes

DPR Structure	Policy Digraph Representation
node	stacked pnode
realizable path	pnode
causation chain	path
dispute wheel	cycle

Table 2: Properties of policy digraphs.

Given a DPR instance, $D = (G, \succ)$, which consists of a graph and a path preference set, we define the policy digraph O to be $O(\succ) = (E, V)$. Each node $P \in V$ represents a realizable path in G . We refer to a node in O 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 :

$$P \rightarrow_{\text{Subpath}} Q$$

- **Policy Edge.** If $P \succ Q$, then P has a *policy* edge to Q :

$$P \rightarrow_{\text{Policy}} Q$$

Figure 4 shows the policy digraph of BAD GADGET in Figure 2. 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. Each set of stacked boxes associated with a node u in G essentially represents node u 's list of preferred paths. We refer to a set of stacked boxes as a *stacked pnode*. Each pnode within a stacked pnode has an *implicit* policy edge to every pnode stacked below it. Subpath edges are represented by the arrows linking two pnodes in different stacked pnodes together. Walks along the policy digraph can be thought of as a reverse ‘‘chutes and ladders’’ game. One can walk down the ladders using policy edges (down the pnodes in a stacked pnode) and up the chutes using subpath edges (between pnodes in different stacked pnodes).

THEOREM 4. Every causation chain $Y = \langle y_0 y_1 \dots y_k \rangle$ of a DPR instance $D = (G, \succ)$ is represented by a path in its corresponding policy digraph $O(\succ)$.

PROOF. From theorem 2, every causation chain is equal to the concatenated rim paths of a dispute fence, represented by: $F = \{\mathcal{N}, \mathcal{Q}, \mathcal{R}\}$ where 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, dispute fence F is represented by a path in policy digraph O as shown in Figure 15.

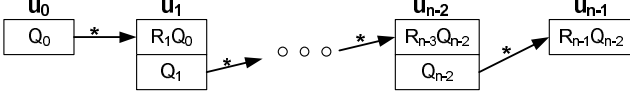


Figure 15: Dispute fences are paths in policy digraphs.

Note that the $*$ notation implies a series of subpath edges through pnodes. Thus every causation chain Y is represented by a path in O . \square

DEFINITION 7 (LENGTH OF POLICY DIGRAPH). We define the length of a policy digraph:

$$\text{Length}(O)$$

to be the longest path of O with repeating nodes. This represents the longest possible causation chain given a DPR instance.

THEOREM 5. Every dispute wheel $W = \{\mathcal{N}, \mathcal{Q}, \mathcal{R}\}$ of a DPR instance $D = (G, \succ)$ is represented by 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 16.

One possible cycle could start at pnode $R_0 Q_{n-1}$. The cycle propagates down the ladders (*i.e.*, policy edges in each stacked pnode) and up the chutes (*i.e.*, subpath edges connecting the stacked pnodes), all the way around, returning to pnode $R_0 Q_{n-1}$. Hence, the pnodes in the cycle are as follows:

$$\langle R_0 Q_{n-1} \ Q_0 \ R_1 Q_0 \ Q_1 \ \dots \ Q_{n-1} \ R_0 Q_{n-1} \rangle$$

\square

7. ROUTING DYNAMICS MINIMIZATION PROBLEM

In this section we formalize the *routing dynamics minimization problem* (RDMP). RDMP is an optimization problem whose goal is to change path preferences to minimize routing dynamics.

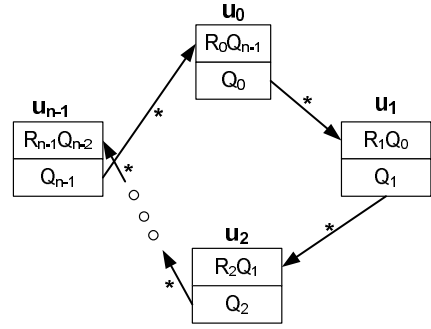


Figure 16: Dispute wheels are cycles in policy digraphs and vice versa. $\text{Length}(O) = \infty$.

7.1 Formal Definition of RDMP

The goal of RDMP is to find the path preferences \succ which will minimize routing dynamics over any time-varying network G . The optimal path preference \succ is to be selected from a set of possible preferences Ω . This set reflects the degree of flexibility in changing the nodes' preferences. An example Ω is where only the path preferences of a single node v can be changed.

The time-varying network $G = (V, E)$ has known vertices V , but unknown edges E . Thus the preferences are minimized over any possible change in the network topology. The routing dynamics of the network, represented by causation chains, is the metric to be minimized. In particular, the goal of RDMP is to minimize the size of the longest causation chain of G , represented by $Y_{\max}(G)$.

Formally stated, RDMP is a min-max problem that finds the preference \succ in Ω which minimizes the dynamics (length of the longest causation chain) of the worst possible network G :

$$\arg \min_{\succ \in \Omega} \max_G |Y_{\max}(G)|$$

Another way of describing the goal of RDMP is to use policy digraphs. The RDMP problem can be restated as a minimization of the length of the resulting policy digraph:

$$\arg \min_{\succ \in \Omega} \text{Length}(O(\succ))$$

7.2 Complexity of RDMP

For RDMP, to determine whether a cycle-free configuration exists is NP-Complete. To find a cycle-free configuration that minimizes the diameter of the policy digraph is NP-Hard. This can be shown with standard reductions using 3SAT and MAX-3SAT, respectively. Due to the ordinary nature of the reductions, they are omitted.

7.3 Sample RDMP Instance

We use our running example from Figure 2 as a sam-

ple problem. The resulting policy digraph can be seen in Figure 4. We define Ω to be the set of preferences in which the path preferences of only *two* nodes can be changed from the original preferences shown in Figure 2. No forbidden path can be made realizable and no currently realizable path can be made forbidden. Using the policy digraph, the resulting possible lengths are outlined in Table 3. Thus the optimal solution is changing/swapping the path preferences of nodes 1 and 3 as shown in the resulting policy digraph in Figure 17.

Nodes with Preference Changes	Length
1,2	4
1,3	3
1,4	5
2,3	4
2,4	∞
3,4	∞

Table 3: Possible policy digraph lengths.

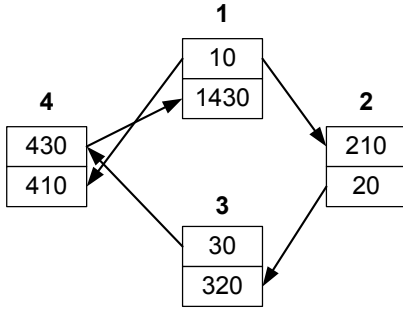


Figure 17: Policy digraph of the optimal solution. It has a length of 3.

8. CONFLICT DETECTION: THEORY

In this section we present several theoretical results using DPR to detect (or infer) the existence of policy conflicts. To this end, we show that causation cycles represent the connection between policy conflicts (*i.e.*, dispute wheels) and routing dynamics. 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 (and their associated path changes) to propagate along Y until y_k (*i.e.*, y_0) receives another route update. With this in mind, given any causation cycle Y , we answer the following questions:

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

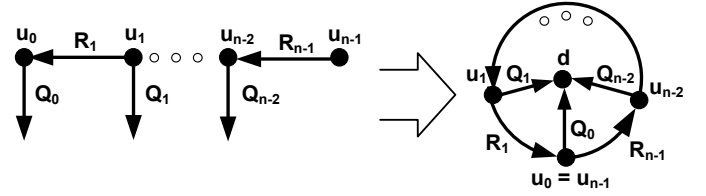


Figure 18: If $u_0 = u_{n-1}$ and $Q_0 \prec R_{n-1}Q_{n-2}$ then a dispute fence is a dispute wheel.

We show that a causation cycle is induced by one of two possible causes:

- Transient route flap (where a path is withdrawn)
- Dispute wheel (cyclic dependency in routing policies)

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 .

From an algorithmic and a practical perspective we 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 as discussed in Section 9.

A brief overview of the theoretical derivations in this section is outlined next. We know that any causation cycle Y , of a DPR instance $D = (G, \succ)$, induces a dispute 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 18. Using F , we show the necessary condition for F to be a dispute wheel in lemma 1. 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 2 and lemma 3 we show how these paths can be determined. This allows us to infer either the existence of a dispute wheel in theorem 6, or the occurrence of a transient route flap in corollary 1. Finally, we outline how y_0 could theoretically infer the existence (or lack thereof) of dispute wheels in practice.

LEMMA 1. A dispute fence $F = \{\mathcal{N}, \mathcal{R}, \mathcal{Q}\}$, of a DPR instance $D = (G, \succ)$, induced by a causation cycle 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 dispute fence is outlined in Figure 18. 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 $Q_{n-1} = Q_0$.

Note that this dispute wheel might be non-proper (*i.e.*, repeating nodes exist), however from theorem 3

we know that a proper dispute wheel is guaranteed to exist in W . \square

LEMMA 2. Given a dispute fence $F = (\mathcal{N}, \mathcal{R}, \mathcal{Q})$, of a DPR instance $D = (G, \succ)$, induced by a causation chain $Y = \langle y_0 y_1 \dots y_k \rangle^t$ 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. As shown in Figure 18, 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 $a = 0$. \square

LEMMA 3. Given a dispute fence $F = (\mathcal{N}, \mathcal{R}, \mathcal{Q})$, of a DPR instance $D = (G, \succ)$, induced by a causation chain $Y = \langle y_0 y_1 \dots y_k \rangle^t$ 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. As shown in Figure 18, 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 $b = 1$. \square

THEOREM 6. 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 dispute fence induced by Y . Using lemma 2 we can determine time offset a and hence path Q_0 . Similarly, using lemma 3 we can determine time offset b and hence path $R_{n-1}Q_{n-2}$. From lemma 1 we know that if the condition

$$Q_0 \prec R_{n-1}Q_{n-2}$$

is satisfied, then the dispute fence F is a dispute wheel. Hence, the existence (or lack thereof) of a dispute wheel can be inferred. \square

COROLLARY 1. 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.

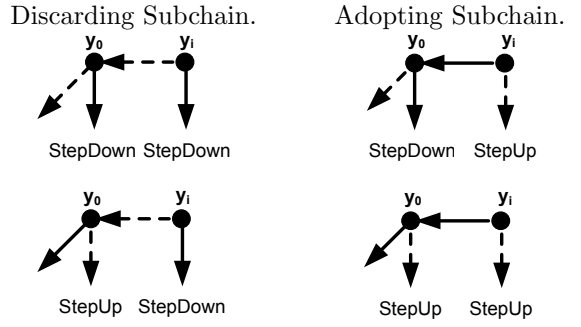


Figure 19: If the action of node y_i is a StepUp/StepDown, then y_0 is part of an adopting/discarding subchain, respectively. The type of subchain which y_0 is a part of is independent of y_0 's action.

PROOF. From theorem 6, 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 and would have continued to use the 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. \square

We next consider node y_0 's ability to infer the cause of causation cycle Y . If node y_0 observes Y , with induced dispute fence F , to infer the existence of a dispute wheel, y_0 must be able to:

- Compute time offset a to determine path Q_0
- Compute time offset b to determine path $R_{n-1}Q_{n-2}$
- Compare the ranking of these two paths

Clearly comparing the ranking of the two paths is trivial. Computing offset b is also possible as it only depends on y_k 's action at time $t + k$ as we can see from lemma 3. Computing offset a , however, is non-trivial as it requires y_0 to know what type of subchain it is a part of, which is not possible. Instead, subsequent nodes in Y need to be considered. If the action of some subsequent node y_i in the causation cycle is a StepUp, then y_i , as well as y_0 are part of an adopting subchain. Conversely, if the action of y_i is a StepDown, then y_0 is part of a discarding subchain. If the action of y_j , $0 < j \leq i$ is a StepSame, then y_0 's subchain type is decided by some subsequent node further along the causation chain.

This is illustrated in Figure 19. The dotted/solid arrows represent paths that were discarded/adopted, respectively. Vertical/horizontal lines represent less/more preferred paths, respectively.

Thus, given the type of subchain that y_0 belongs to, the existence of a dispute wheel inference problem can be solved. This solution is indeed local but not possible independently as it requires the co-operation of node y_i (the first node along the causation cycle Y that performed a StepUp/StepDown).

9. CONFLICT DETECTION: PRACTICE

Dispute wheels represent the necessary conditions for routing divergence. Permanent divergence of routing 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 [8] and could potentially lead to unbounded dynamics.

The SafetyPulse algorithm is a distributed algorithm to provably detect dispute wheels. Once dispute wheels are detected, they can be reported to administrators for further analysis. SafetyPulse distinguishes itself by leveraging the results of the DPR model to have a unique set of beneficial characteristics:

- **Provably Correct.** SafetyPulse is provably correct with any dynamic network. Thus any changes in the network topology do not affect the correctness of dispute wheel detection.
- **Privacy Guarantees.** Path preferences are not revealed between neighbors.
- **Efficient Space.** Each node only adds a small token of space complexity $O(1)$ to a route it sees.
- **Provider 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.

9.1 Overview of Algorithm

SafetyPulse is a dynamic policy conflict detection algorithm, which piggybacks *messages* alongside route updates. One possible implementation of SafetyPulse on BGP would be to use message options. Each node can place a child *token* in this piggybacked 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 piggybacks messages along causation chains between nodes.

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 inferred to exist

by comparing the relative ranking of y 's realized paths around these times. Using theorem 6, 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 2, 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 3, 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 20.

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

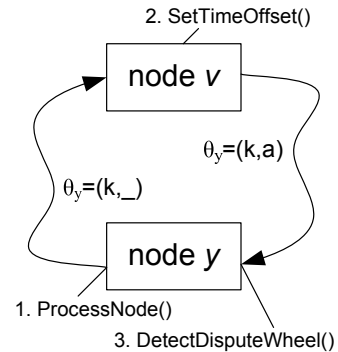


Figure 20: Overview of SafetyPulse algorithm.

9.2 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 $\theta_y = (k, -)$,

```

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

```

Figure 21: SafetyPulse token creation and action storage.

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:

- 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:

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

These 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 21.

9.3 Receiving the Token

When a node y receives a token θ_y that it has created previously 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 some 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, we need to determine the second time offset b to find $\pi(y, t_{\text{in}} + b)$. From lemma 3 we know that b depends on the action performed by y . If y is a StepDown then $b = 0$. Otherwise $b = 1$. According to theorem 6, 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 22.

```

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

```

Figure 22: SafetyPulse token receipt and dispute wheel detection.

9.4 Computing time offset

The one part missing is how to determine which time offset a to use. In Section 8, we showed that the value of a is dependent on the type of subchain that y belongs to. This type 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 23 shows how third-party nodes can fill in the time offset a .

```

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

```

Figure 23: SafetyPulse time offset computation.

9.5 SafetyPulse Algorithm

The overall SafetyPulse algorithm in Figure 24 for each node y at time t can be described simply as the combination of the three algorithms described above.

9.6 Implementation Issues

The token sent by every node has two parts, the key k and the time offset a . The key needs to index an action stored in the home node y . If node y is expected to switch between 2^i paths within a reasonable time

```

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

```

Figure 24: SafetyPulse algorithm.

frame, then the size of k only needs to be that of i . Thus if y switches between 16 routes, the key is 4 bits long.

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.
- If action of v is a StepUp, flip b_1 .

When node y receives $a = b_0b_1$ it will check to see if b_0 is set and if b_1 is flipped (compared to a locally stored version of a), then node y knows that it should use the adopted path. Otherwise if b_1 is unchanged, then node y knows that it should use the discarded path. Thus the overhead caused by each node is negligible.

10. RELATED WORK

It has been noted that inter-domain routing (BGP) can persist indefinitely and lead to permanent divergence [10][2]. The stable paths problem [3] was introduced to address this issue, providing steady-state analysis of policy-based path vector protocols.

There have been numerous attempts to address the routing divergence issue. One of the first attempts to solve SPP, was a history-based dynamic protocol that ran concurrently with BGP [6]. Other less computationally expensive protocols to deduce routing conflicts include counting metrics [4], or passing a token [11][5]. Another approach is to constrain the policy freedom of ASes to a generalized form of shortest path routing, guaranteeing convergence [12] [7] [8]. There are numerous offline methods to address routing conflicts, such as the centralized Internet Routing Repository [13], and downloadable tools to analyze static policies [14].

Analyzing the dynamics of inter-domain routing is also an important topic. It has been shown that the convergence time of inter-domain routing can last up to 30 minutes [1]. There has yet to be a standardized way to address the dynamics of policy routing. In [15], a state graph was used to give sufficient conditions for transient routing failures and upper bounds for routing changes. [16] annotated structures from SPP with time bounds to provide upper bounds for convergence. [17] provided a general framework for analyzing the upper bounds of routing dynamics for shortest path routing

under single link failure. Other methods include identifying the route causes of instability [18][19] or recovering from instability using backup routes [9].

11. CONCLUSIONS AND FUTURE WORK

In this paper we introduced the dynamic policy routing model (DPR). It provides a rigorous framework for understanding the transient behavior of policy routing. We started by introducing a routing example, BAD GADGET, and its steady-state analysis. Using results from the DPR model, we showed how the routing dynamics of BAD GADGET can be analyzed/minimized using policy digraphs and the routing dynamics minimization problem. We also used DPR to introduce SafetyPulse, a distributed policy conflict detection algorithm. SafetyPulse excelled in all criteria parameters for such algorithms: time/space overhead, privacy, soundness, and ease of adoption.

The routing dynamics minimization problem (RDMP) can be used on AS preferences to infer how to best improve BGP convergence time. RDMP can be used on theoretic models, such as Gao-Rexford models [8] and various relaxations of its constraints. Future work on RDMP can also focus on developing efficient algorithms for determining optimal solutions given different constraints.

The DPR model can be extended in a number of ways. Path preferences can be modified to be time-dependent, enabling DPR to model misconfigured or malevolent nodes. Though asynchronous time and activation sequences do not affect the fundamentals of DPR, their inclusion can provide insight into the upper bounds of convergence time.

12. ACKNOWLEDGMENT

This work has been partially supported by National Science Foundation awards: CISE/CCF #0820138, CISE/CSR #0720604, CISE/CNS #0524477, CNS/ITR #0205294, and CISE/EIA RI #0202067.

13. 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] J. Cobb and R. Musunuri, "Enforcing Convergence in Inter-Domain Routing,"

- GLOBECOM*, vol. 3, pp. 1353–1358, November 2004.
- [5] S. Yilmaz and I. Matta, “An Adaptive Management Approach to Resolving Policy Conflicts,” in *IFIP Networking 2007*, Atlanta, Georgia, May 2007.
 - [6] T. Griffin and G. T. Wilfong, “A Safe Path Vector Protocol,” in *INFOCOM*, 2000, pp. 490–499.
 - [7] T. G. Griffin and J. L. Sobrinho, “Metarouting,” in *SIGCOMM*, 2005, pp. 1–12.
 - [8] L. Gao and J. Rexford, “Stable Internet Routing Without Global Coordination,” *IEEE/ACM Trans. Netw.*, vol. 9, no. 6, pp. 681–692, 2001.
 - [9] L. Gao, T. Griffin, and J. Rexford, “Inherently Safe Backup Routing with BGP,” in *IEEE INFOCOM*, 2001, pp. 547–556.
 - [10] T. G. Griffin and G. Wilfong, “An Analysis of BGP Convergence Properties,” in *SIGCOMM*, 1999, pp. 277–288.
 - [11] Ahronovitz, J.-C. Knig, and C. Saad, “A Distributed Method for Dynamic Resolution of BGP Oscillations,” in *IPDPS*, 2006.
 - [12] 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.
 - [13] 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.
 - [14] N. Feamster and H. Balakrishnan, “Verifying the Correctness of Wide-Area Internet Routing,” MIT, Tech. Rep., May 2004.
 - [15] F. Wang, L. Gao, and J. Qiu, “On Understanding of Transient Interdomain Routing Failures,” in *ICNP*, 2005.
 - [16] D. Obradovic, “Real-time Model and Convergence Time of BGP,” in *INFOCOM*, 2002.
 - [17] 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.
 - [18] M. Caesar, L. Subramanian, and R. Katz, “Towards Localizing Root Causes of BGP Dynamics,” UC Berkeley, Tech. Rep. CSD-04-1302, 2004.
 - [19] A. Feldmann, O. Maennel, Z. Mao, A. Berger, and B. Maggs, “Locating Internet Routing Instabilities,” in *SIGCOMM*, September 2004.