BOSTON UNIVERSITY

GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

POLICY ROUTING DYNAMICS: THEORY AND APPLICATIONS

by

KARIM ABDEL MAGID MATTAR SHABAN

Bachelor of Science, Computer Systems Engineering, UMass Amherst, 2003
Master of Art, Computer Science, Boston University, 2008

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2011

Approved by

First Reader

Abraham Matta, PhD
Associate Professor of Computer Science

Second Reader

Azer Bestavros, PhD
Professor of Computer Science

Third Reader

Sharon Goldberg, PhD
Assistant Professor of Computer Science

Acknowledgments

First and foremost I would like to thank my advisor, Dr. Abraham Matta, for his support, guidance, and infinite patience. I would also like to thank my friend and collaborator Samuel Epstein. Without Abraham and Samuel, this thesis would have never been possible. I owe my PhD to them in so many ways that no words will ever be enough to give them credit for what they have helped me accomplish. I am truly indebted.

I would like to thank my parents Abdel Magid Mattar and Salwa Mattar, my brother Marwan Mattar, and my girlfriend Kathleen O'Boyle for always being there for me. I do not know where I would be today without their support and their final push to help me finish my PhD. I feel lucky to be so loved and to have such a close and supportive family.

During my seven years at BU, I had the pleasure of being part of many gangs. I would like to thank the Egyptian gang: George Atia, Hany Morcos, Raymond Sweha, Mina Guirguis, Christine Bassem, and Vatche Ishakian. Vatche is not Egyptian but he is as close as one can be. I would like to thank the Greek gang: Panagiotis Papapetrou, Michalis Potamias, Niky Riga, Georgios Smaragdakis, Vassilis Athitsos, Giorgos Zervas, and Nikolaos Laoutaris. I would like to thank the Indian gang: Anukool Lakhina, Vijay Erramilli, Ashwin Thangali, Sowmya Manjanatha, Kanishka Gupta, and Parminder Chhabra. I would like to thank the Brazilian gang: Marisa Vasconcelos, Nahur Fonseca, and Michel Machado. I would also like to thank Flavio Esposito, Jorge Londono, Yuting Zhang, Joseph Akinwumi, Andrej Cvetkovski, Gonca Gursun, Chong Wang, Ching Chang, Feifei Li, Vitaly Ablavsky, Tai-Peng Tian, Johanna Brewer, Rui Li, and Yarom Gabay. I'm sure I've missed many names. For that I'm truly sorry. I hope everyone knows just how much their friendship and support meant to me.

I would like to thank all my committee members: Dr. Azer Bestavros, Dr. John Byers, Dr. Assaf Kfoury, Dr. Mark Crovella, and Dr. Sharon Goldberg. I could not have asked for a committee with more expertise in networking than the one I had. I'm very lucky to have studied under each and every one of them.

Last but not least I would like to thank Ellen Grady, Jennifer Streubel, Paul Stauffer, Ernest

Kim, Wesley Harrel, Austin Wolfe, Charles Willis, and all the other staff members at the Computer Science department. I will never forget their infinite patience and help with all sorts of administrative, technical, and personal problems. They really make BU a better place and certainly made my life a lot easier, especially when I felt like there was no one else to turn to. I'm forever indebted to all of them.

POLICY ROUTING DYNAMICS: THEORY AND APPLICATIONS

(Order No.)

KARIM ABDEL MAGID MATTAR SHABAN

Boston University, Graduate School of Arts and Sciences, 2011

Major Professor: Abraham Matta, Associate Professor of Computer Science

ABSTRACT

The Internet consists of thousands of autonomous systems (ASes). Each AS represents an Internet Service Provider or the network of a large organization, that is managed independently. The Border Gateway Protocol is the policy routing protocol of choice for connecting these ASes, while allowing them to set their routing policies independently. Routing policies (or path preferences) 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, where ASes may have conflicting policies causing them to continually advertise new routing updates for extended periods of time.

We introduce a theoretical framework for policy routing dynamics (*i.e.*, how path changes propagate in the network) that is based on the specifics of routing update mechanisms. Unlike existing models, our Dynamic Policy Routing (DPR) model introduces several structures that capture how path changes propagate in any network under dynamic topology and path preference changes. We demonstrate the utility of DPR by applying it to three problems: minimizing routing dynamics, detecting policy conflicts, and deriving properties of safe (*i.e.*, convergent) routing dynamics.

For minimizing routing dynamics we formulate the Routing Dynamics Minimization Problem (RDMP) which solves a graph optimization problem. RDMP aims to minimize the longest possible sequence of routing update messages in a dynamic network by changing the path preferences of nodes. We show that solving RDMP in general is NP-Hard and under restrictions can be solved in

polynomial time.

For detecting policy conflicts we prove that the root cause of any cycle of routing update messages can be precisely inferred as either transient or potentially persistent due to the existence of a policy conflict. We then develop SAFETYPULSE, a token-based distributed algorithm, to detect policy conflicts in a dynamic network.

For deriving properties of safe routing dynamics we establish three properties that provide insight into which ASes can directly induce route changes in one another, and how cycles of routing updates can be manifested in the network. We then develop INTERFERENCEBEAT, a token-based distributed algorithm, to check adherence to these properties.

Contents

List of Tables

List of Figures

List of Abbreviations

AS	. . . . .	Autonomous System
BGP	. . . . .	Border Gateway Protocol
DPR	. . . . .	Dynamic Policy Routing
ISP	. . . . .	Internet Service Provider
RDMP	. . . . .	Routing Dynamics Minimization Problem
SPP	. . . . .	Stable Paths Problem
SPVP	. . . . .	Safe Path Vector Protocol

Chapter 1

Introduction

The Internet consists of thousands of autonomous systems (ASes). Each AS represents an Internet Service Provider or the network of a large organization, that is managed independently. Today, the Border Gateway Protocol (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 is known to suffer from slow convergence time, where ASes continually advertise new routing updates for extended periods of time before reaching a stable path assignment. Experimental measurements show that interdomain routers may take tens of minutes to reach a consistent view of the network after a fault [Labovitz et al., 2001].

Route flapping, the process of adopting and discarding paths, can be highly disruptive given the associated communication and processing overheads. Route flaps can be transient (*i.e.*, short-term) due to temporary changes in topology or path preferences. Route flaps can also be persistent due to conflicting routing policies across ASes (*i.e.*, policies that cannot be satisfied simultaneously [Varadhan et al., 1996]). Unnecessarily switching between routes reduces QoS predictability, increases delay variability, causes service disruption, as well as increases packet loss [Labovitz et al., 2000]. In addition, one can imagine that continually switching between routes would make managing the network in terms of capacity planning / dimensioning, and traffic engineering much harder as the paths used and their associated traffic loads become less predictable [Quoitin et al., 2005, Uhlig and Bonaventure, 2004, Feamster and Rexford, 2007]. In general, network operators strive for a stable policy configuration (*i.e.*, a set of path preferences) where the routing dynamics are bounded and converge quickly. By routing dynamics we mean how path changes are propa-

gated across nodes in the network.

The difficulty in managing the disruptive properties associated with BGP due to the autonomy in setting routing policies has led to a plethora of research related to understanding its convergence properties and steady-state behavior. In particular, Griffin *et al.* introduced the Stable Paths Problem (SPP), a formalism to reason about the steady-state behavior of BGP [Griffin et al., 2002]. SPP has become the standard for modeling BGP and the basis for many novel extensions over the years. SPP considers the stable assignment of paths, where every AS is assigned its most preferred path out of its available choices. The authors showed that the existence of a dispute wheel or policy conflict (*i.e.,* a cyclic dependency in path preferences that could lead to paths being adopted and discarded indefinitely) is a necessary condition for divergence (*i.e.,* the lack of a stable assignment of paths). Gao *et al.,* on the other hand, showed that restricting the path preferences of ASes to be consistent with their commercial / economic relationships is sufficient for guaranteeing convergence [Gao and Rexford, 2001]. We refer to these restrictions from hereon as the Gao-Rexford (economic) guidelines.

1.1 Motivation

SPP is seminal work which introduces modern policy routing theory and represents a unifying framework for understanding steady-state analysis. To study routing dynamics (*i.e.,* how path changes propagate across nodes in the network), current routing models aim to capture asynchronicity and the timing of BGP updates, which make the models cumbersome.

This thesis takes a different approach. We show that important properties about routing dynamics can be derived using a simple theoretical framework that extends SPP. The routing dynamics in our proposed model are actually synchronous with discrete time. We show that properties derived using our synchronous routing model can be applied to asynchronous routing dynamics. This allows us to prove interesting results regarding the dynamics of policy routing using a simple model. Our model also captures dynamic topology and path preference changes. Our contributions are outlined in more detail in the following section.

Our model starts by considering the specifics of how routing happens in the Internet today.

During the course of routing, a router has periodic windows in which it receives routing update messages from its neighbors. From these updates, the router can optionally choose to change its current path (*i.e.,* perform an action) or maintain its current path. We use the notion of *causation*, where every action by a node is caused by one specific neighbor. Thus the routing dynamics can be seen as a collection of causation chains, where each node's action is caused by the previous node on the chain. We select a natural definition of causation in which causation chains are started only by root causes (*i.e.,* link availability changes or policy changes). We find that the notion of causation is sufficient to derive interesting properties of routing dynamics as a whole.

1.2 Thesis Contributions

We introduce DPR which constitutes several novel structures such as causation chains, causation fences, and policy digraphs that model different aspects of routing dynamics (*i.e.,* how path changes propagate across nodes in the network) and provide insight into how these dynamics manifest in the network. We demonstrate the utility of DPR by applying it to three problems: minimizing routing dynamics, detecting policy conflicts, and deriving properties of safe routing dynamics.

In terms of minimizing routing dynamics we make the following contributions:

- We introduce policy digraphs, a time-invariant structure which captures how routing update messages can propagate in the network. We utilize policy digraphs to formalize the Routing Dynamics Minimization Problem (RDMP). RDMP solves a graph optimization problem that aims to minimize the longest possible sequence of routing update messages in a dynamic network. This is done by changing the path preferences of nodes.

- We show that finding a policy configuration (*i.e.,* a set of path preferences) which minimizes the length (*i.e.,* the size of the longest walk with possibly repeated nodes) of the policy digraph is NP-Hard.

- We show that under certain restrictions, such as having nodes abide by the Gao-Rexford

guidelines that guarantee safety (*i.e.,* convergence), finding a policy configuration which minimizes the length of the policy digraph can be solved in polynomial time.

In terms of policy conflict detection we make the following contributions:

- We introduce causation fences, a time-invariant structure which under certain conditions represents a dispute wheel. We utilize causation fences to 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.

- We develop SAFETYPULSE, a token-based distributed algorithm, which leverages our theoretical result for detecting policy conflicts in any dynamic network. SAFETYPULSE has several characteristics, namely, it is computationally efficient, provably correct, and backwards compatible. SAFETYPULSE diagnoses and monitors the health of the network by detecting policy conflicts that could potentially lead to unbounded routing dynamics in realtime.

In terms of deriving properties of safe routing dynamics we make the following contributions:

- We introduce causation chains, a time-varying structure that captures how the action of one node on the chain causes its successor to take an action. We utilize causation chains to establish three properties of safe routing dynamics. By "safe" we mean routing instances where all the nodes abide by the Gao-Rexford guidelines that guarantee safety. Here, by safety we mean the convergence of the policy routing protocol (*e.g.,* BGP) to a stable assignment of paths across all nodes (*i.e.,* when no more path changes are propagated in the network). The non-interference property provides insight into which ASes can directly induce route changes in one another. The single cycle property and the multi-tiered cycle property both provide insight into how cycles of routing updates can manifest in the network. These properties hold irrespective of changes in the underlying topology or changes in path preferences.

- We develop INTERFERENCEBEAT, a token-based distributed algorithm, to check adherence to these properties. To enhance INTERFERENCEBEAT we model four common policy vio-

lations of the Gao-Rexford guidelines and characterize the resulting dynamics. INTERFERENCEBEAT diagnoses and monitors the health of the network by detecting invalid routing dynamics (*i.e.,* causation chains that do not adhere to the derived properties) in realtime.

1.3 Thesis Organization

This thesis is organized as follows. Chapter 2 provides an overview of the related work in the area. Chapter 3 outlines our models for policy routing dynamics. In particular, the Dynamic Policy Routing (DPR) model and the economic DPR model. Chapter 4 formalizes the Routing Dynamics Minimization Problem (RDMP) and shows the complexity classes for various variants of the problem. Chapter 5 solves the conflict detection problem and provides pseudocode for SAFETYPULSE. Chapter 6 derives properties of safe routing dynamics and provides pseudocode for INTERFERENCEBEAT. This chapter also models four common policy violations and characterizes the resulting dynamics. Finally, the thesis concludes in chapter 7 and Appendix A addresses the synchronicity of DPR.

Chapter 2

Related Work

There has been some seminal work in terms of understanding the behavior of BGP, in particular its steady-state behavior and convergence properties. As mentioned earlier, Griffin *et al.* introduced the Stable Paths Problem (SPP) and showed that the existence of a dispute wheel (*i.e.,* a cyclic dependency in path preferences) is a necessary condition for divergence (*i.e.,* the lack of a stable assignment of paths). Feamster *et al.* 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 [Feamster et al., 2005]. Cittadini *et al.* also provided necessary and sufficient conditions for safety under filtering [Cittadini et al., 2009]. Their result is based on the presence of a dispute reel (*i.e.,* a special case of the dispute wheel and a generalization of the dispute ring). Sami *et al.* showed that having a unique stable assignment is a necessary condition for convergence [Sami et al., 2009].

There have been attempts to provide routing models which are guaranteed to have a solution in that the model will somehow incorporate the endless oscillations of the disputing nodes. One such approach is the Fractional Stability Model (FSM), where each node chooses a "mixed strategy" such that there is a probability associated with selecting each path presented by a neighboring node [Haxell and Wilfong, 2008]. Since all non-cooperative games have a Nash equilibrium, instances of the FSM are guaranteed to have a solution. Sink equilibriums represent another model which has been applied to routing [Fabrikant and Papadimitriou, 2008]. This approach models dynamics as a graph where every state of the system is a node in the graph and state transitions are edges.

Wang *et al.* developed a BGP routing model to understand transient routing failures [Wang et al., 2009]. The model consists of routers acting asynchronously through the use of an activation

sequence. Under the assumption that all nodes abide by the Gao-Rexford guidelines [Gao and Rexford, 2001], they derived sufficient conditions for transient route failure and upper bounds on the duration of transient route failures. Bounds on BGP's convergence time, under different restricted link failure models, have also been studied (*e.g.*, [Wang et al., 2005, Obradovic, 2002, Pei et al., 2006]).

There are research efforts that utilize other fields, such as game theory, to understand and model policy routing. Since each router can be represented as an ordered list of path preferences, routing is compatible with mechanism design [Feigenbaum et al., 2006b]. Using this framework, each node can be given a payment as an incentive to truthfully reveal its policy preferences. A globally optimal set of path assignments can then be determined from these preference revelations. This method can be further optimized with distributed algorithmic mechanism design [Feigenbaum and Shenker, 2002], where the routers jointly determine the payment methods without requiring a centralized coordinator [Feigenbaum et al., 2006a]. One problem with this approach, however, is that although routers typically rank path preferences using a numeric function, the values associated with each path are arbitrary. For example, if a router assigns two paths numeric values, 1000 and 50, respectively, it cannot be interpreted that the first path is valued 20 times more than the second path. However, it is shown that if no disputes exist in the routing policies (*i.e.*, there are no dispute wheels), then the routers are incentive compatible in that they are motivated to act truthfully with respect to their path preferences [Levin et al., 2008].

In contrast to existing BGP models, our Dynamic Policy Routing (DPR) model extends SPP with discrete synchronous time to capture the propagation of path changes across nodes. DPR does not utilize other fields, does not attempt to model the asynchronicity in BGP, and is not restricted to static topologies or static path preferences. DPR produces results that are invariant to dynamic topology (*i.e.*, multiple changes in link availability) and path preference (*i.e.*, policy configuration) changes.

Gao *et al.* showed that restricting the path preferences of ASes to be consistent with their commercial / economic relationships (*e.g.*, prefer customer paths over provider paths) is sufficient for guaranteeing convergence [Gao and Rexford, 2001]. Other solutions constrain the policy freedom

of ASes to a generalized form of shortest path routing, thus guaranteeing convergence (*e.g.*, [Ee et al., 2007, Griffin and Sobrinho, 2005, Gao and Rexford, 2001]). In general, conditions that guarantee convergence by limiting the freedom of AS administrators in choosing their routing policies are heavy-handed as they require every node to comply. They also do not provide any guarantees under partial adherence.

The properties of safe routing dynamics we derive in Chapter 6 consider a notion of safety that is based on the Gao-Rexford guidelines in [Gao and Rexford, 2001]. These guidelines are modelled by our Economic DPR model in Chapter 3.4.

Many distributed algorithms were developed to mitigate the effects of harmful policy interactions. This is done by passing diagnostic information alongside routing update messages (*e.g.*, a cost metric [Cobb and Musunuri, 2004], a precedence metric [Ee et al., 2007], path-histories [Griffin and Wilfong, 2000], as well as event-related tokens [Yilmaz and Matta, 2007, Ahronovitz et al., 2006]). Many of these solutions, however, are either ad hoc or cumbersome. For example, counting [Cobb and Musunuri, 2004] and other token-based [Yilmaz and Matta, 2007] approaches are heuristics and their correctness cannot be guaranteed. The theoretical history-based protocol, Safe Path Vector Protocol (SPVP) introduced in [Griffin and Wilfong, 2000], on the other hand, incurs a large message exchange overhead, requires more time (*i.e.*, more messages) to detect a policy conflict, and does not provide an explicit condition for detecting the occurrence of a transient route flap.

SAFETYPULSE, our token-based distributed algorithm, leverages our novel theoretical results in Chapter 5 to detecting policy conflict. In particular, we identify the root cause of a causation cycle as either a transient route flap or a policy conflict, an inference that SPVP is unable to make. SAFETYPULSE has several characteristics, namely, it is computationally efficient (a constant factor reduction in message size and number of messages when compared to SPVP), provably correct, and backwards compatible.

There are numerous offline methods for addressing policy conflicts [Govindan et al., 1999] and analyzing static policy configurations [Feamster and Balakrishnan, 2004]. Other methods focus on identifying the root causes of instability [Feldmann et al., 2004]. In general, these offline

methods have not been successful in practice as network administrators are reluctant to disclose their routing policies or configurations to any central repository for further analysis.

Chapter 3

Modeling Policy Routing Dynamics

In this chapter we outline our models for policy routing dynamics. Our Dynamic Policy Routing (DPR) model extends the static formalism of the Stable Paths Problem (SPP) [Griffin et al., 2002] with discrete synchronous time. DPR captures the propagation of path changes in any network irrespective of its time-varying topology or time-varying path preferences. Our economic DPR model, on the other hand, captures the economic constraints that are typical of commercial relationships (or agreements) between ASes in the Internet [Gao and Rexford, 2001]. We refer to these constraints, which have been shown to make BGP free from policy conflicts, as the Gao-Rexford guidelines.

3.1 Stable Paths Problem

3.1.1 Overview

We start with a sample SPP instance that can be seen in Figure 3·1 (Left) where the destination 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 $\langle 10 \rangle$. This SPP instance is called BAD GADGET [Griffin and Wilfong, 2000] and is known to have at least one dispute wheel as shown in Figure 3·1 (Right). BAD GADGET also has no stable assignment as any initial path assignment leads to paths being adopted and discarded indefinitely. We use BAD GADGET as a running example throughout this thesis.

3.1.2 Basic Notation

Definition 1 (Network)**.** 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 $(u, v) \in E$.

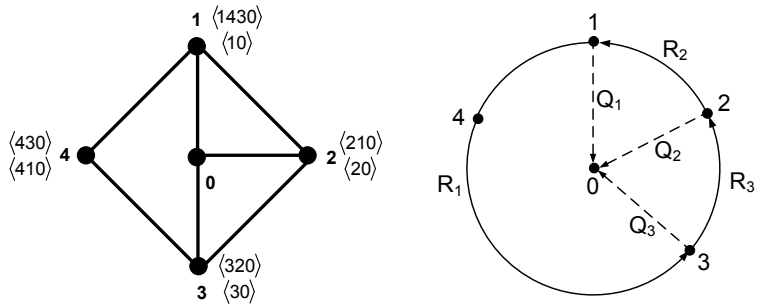


Figure 3·1: SPP instance BAD GADGET (Left) and its dispute wheel (Right). A dispute wheel consists of pivot nodes. Two types of paths are represented in a dispute wheel, namely, rim paths labelled R_i and spoke paths labelled Q_i. Rim paths connect pivot nodes while spoke paths connect each pivot node to the destination. Each pivot node i has the property that it prefers its rim and neighbor's spoke path, R_iQ_{i-1}, over its direct spoke path, Q_i.

Definition 2 (Paths)**.** Paths in G are represented by sequences of the form:

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

where d is a distinguished destination node. 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 $\mathcal{P}$.

Definition 3 (Next-Hop Neighbor)**.** At node u_0, the next-hop of $P = \langle u_0\ u_1\ \ldots\ u_n\ d \rangle$ is denoted by:

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

Definition 4 (Path Preferences)**.** 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.

Definition 5 (SPP Instance)**.** An instance of SPP is comprised of the network and the path preferences of each of its nodes: $(G, \succ)$.

Definition 6 (Stable Path 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:

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

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 [Griffin et al., 2002] that it is NP-Complete to determine whether a stable assignment exists.

Definition 7 (Dispute Wheel)**.** Griffin *et al.* introduced *dispute wheels* in [Griffin et al., 2002], whose existence is a necessary condition for an SPP instance to not have a stable assignment. A dispute wheel, as shown in Figure 3·2, 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 $\mathcal{N} = \{u_{n-1}, \ldots, u_0\}$.
- $\mathcal{R}$ 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:

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

3.2 Dynamic Policy Routing Model with Static Path Preferences

3.2.1 Overview

In this section we outline the DPR model, our theoretical framework for modeling policy routing dynamics, with static path preferences that will be used in chapter 4 and chapter 5.

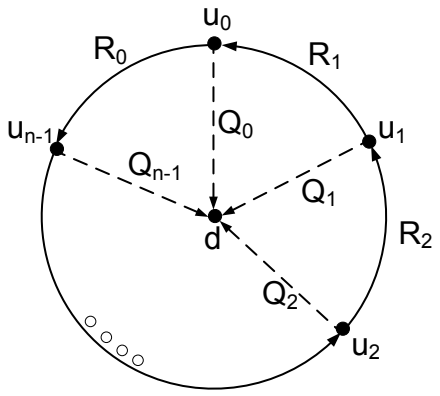


Figure 3·2: Dispute wheel.

We introduce the basic notation underlying the DPR model and define several novel structures such as causation chains, causation fences, and policy digraphs that model different aspects of routing dynamics (*i.e.,* how path changes propagate across nodes in the network) and provide insight into how these dynamics manifest in the network. We focus here on presenting the main intuition behind the model and the structures developed.

Consider the sample SPP instance, BAD GADGET, shown in Figure 3·1. In the course of routing, nodes adopt and discard paths as they attempt to reach a stable path assignment. The adopted and discarded paths are the paths that the node switches to and from, respectively. A stable path assignment exists when no node is able to switch to a more preferred path and no more path changes are propagated in the network.

While SPP is concerned with the stable assignment of paths, DPR is concerned with the propagation of path changes in the network. The central notions in DPR are that of *action* and *causation*. An action, Action(u, t), corresponds to a possible routing decision made by node u at time t upon the reception of a routing update message. Three possible actions include a StepUp (adopting a more preferred path via a different next-hop neighbor) or StepDown (adopting a less preferred path via a different next-hop neighbor) or StepSame (adopting a new path via the same next-hop neighbor). A causing node, Cause(u, t), corresponds to the node sending the routing update message to node u at time t that triggered the action. More specifically, if node u performs a StepUp, the

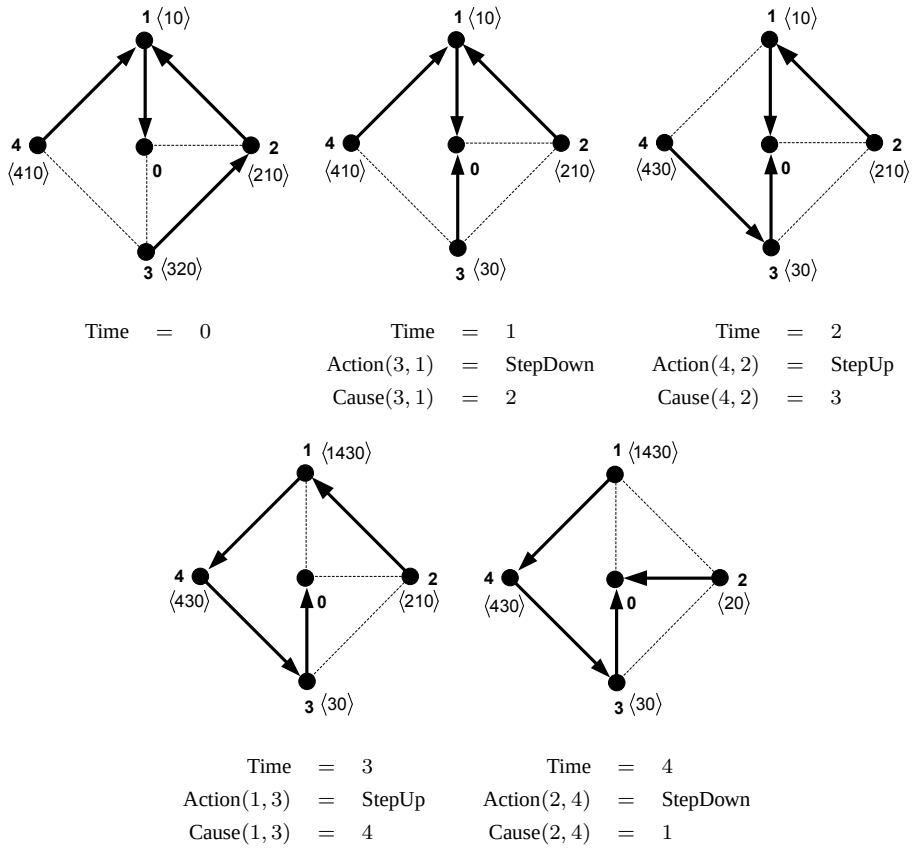


Figure 3·3: Sample actions and causation for BAD GADGET.

causing node is the next-hop neighbor that provided node u with the more preferred path. On the other hand, if node u performs a StepDown / StepSame, the causing node is the next-hop neighbor that removed node u's current path. DPR models these two events to construct a causation chain over time, $\langle y_1\ y_2 \ldots y_k \rangle$, where each node y_i causes its successor along the chain to take an action.

Figure 3·3 outlines a few sample cases of action and causation for BAD GADGET. 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. A sample causation chain is $\langle 2\ 3\ 4\ 1 \rangle$ while a sample causation cycle, where a node is triggered twice to update its path, is $\langle 2\ 3\ 4\ 1\ 2 \rangle$.

It is important to note that each causation chain starts at a root cause that can be either a change

in a link's availability or a change in a node's path preferences. A causation chain terminates, on the other hand, when a node does not take an action (*i.e.,* is not affected by this round of route updates and hence maintains its current path). A causation chain represents a single sequence of nodes that induce path changes in one another. A network would typically observe many causation chains as a result of a single root cause. In other words, the individual chains branch out in such a way that their aggregation across space and time would form a complex graph (or forest) that simultaneously represents many sequences of route updates propagating in the network. In this thesis whenever we talk about a causation chain we are referring to only one such sequence of route updates propagating in the network.

Another set of possible actions include RankInc (adopting a more preferred path) or RankDec (adopting a less preferred path) or RankSame (staying with the current path) without any restrictions on the next-hop neighbor used. Depending on the application, a different set of possible actions may be more appropriate. For chapters 4 and 5, the actions StepUp, StepDown, and StepSame are used to construct the causations chains, since we care about how the adopted / discarded paths depend on the nodes along the chain. Also, we only consider static path preferences as outlined in Section 3.2. For chapter 6, on the other hand, the actions RankInc, RankDec, and RankSame are used since we only care about the relative ranking of the current path and the new (or next) path. Also, we consider time-varying path preferences as outlined in Section 3.3.

Using DPR we introduce a time-invariant structure we call a causation fence, which under certain conditions represents a dispute wheel (or policy conflict). The exact manner in which a causation fence manifests, in terms of what we call adopting and discarding subchains, for BAD GADGET is outlined in Figure 3·4.

These adopting and discarding subchains allow us to construct a causation fence which distills the core elements (*i.e.,* path changes) in a causation chain. In particular, a causation fence only concerns itself with the head and tail nodes of the adopting / discarding subchains as shown in Figure 3·5. The causation fence can be seen as an open-ended dispute wheel where each pivot node also prefers its rim and neighbor's spoke path over its own spoke path. For example, pivot node 3 prefers its rim and neighbor's spoke path $\langle 320 \rangle$ over its own spoke path $\langle 30 \rangle$. The condition

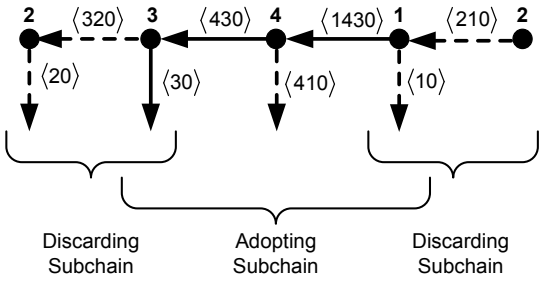


Figure 3·4: Alternating subchains of BAD GADGET. Adopted/discarded paths are represented by solid/dotted arrows, respectively. Horizontal paths are more preferred than vertical paths.

under which a causation fence does indeed represent a dispute wheel allows us to infer the root cause of a causation cycle.

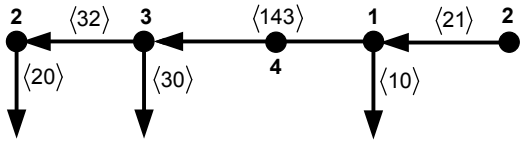


Figure 3·5: Causation fence of BAD GADGET.

Using DPR we introduce another time-invariant structure we call a policy digraph which captures how routing update messages can propagate in the network. Figure 3·6 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. Such 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 $\langle 20 \rangle$ in the policy digraph captures the effect of node 2 adopting path $\langle 210 \rangle$ and discarding path $\langle 20 \rangle$. This results in path $\langle 320 \rangle$ getting discarded by node 3 since path $\langle 20 \rangle$ is no longer available.

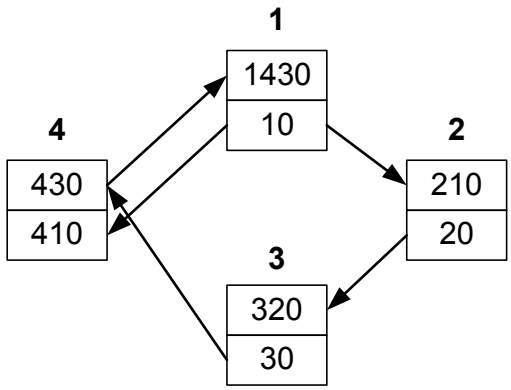


Figure 3·6: Policy digraph of BAD GADGET.

The policy digraph provides insight into the routing dynamics. A path (or walk) 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. *We prove that a dispute wheel is represented as a cycle in the policy digraph.* Policy digraphs will be used in the formulation of RDMP in chapter 4.2.

It is important to note that our policy digraphs are easier to construct and visualize when compared to dispute digraphs [Griffin and Wilfong, 2000]. In particular, the conditions for constructing a dispute arc require the relative rankings of paths across nodes to be compared—something that is not required for constructing our policy digraph. Policy digraphs are simpler since they are concerned with capturing the propagation of path changes across nodes in the network. In particular,

they are concerned with capturing how paths are adopted and discarded. Dispute digraphs, on the other hand, are concerned with how policy conflicts can manifest and hence consider the relative rankings of paths across nodes which makes the structure more complicated.

3.2.2 Basic Notation

DPR extends SPP's notation as follows.

Definition 8 (Time)**.** Time is represented by a non-negative, discrete index t such that: $t \in [0, \infty)$.

Definition 9 (Network)**.** The network is represented by a graph $G = (V, E)$:

- Each vertex $u \in V$ represents an AS.
- Each edge in E is time dependent: $(u, v)^t \in E$ if u is connected to v at time t. Conversely, a lack of connectivity between u and v at time t (*i.e.,* link failure) is represented by $(u, v)^t \notin E$.

There exists a distinguished destination node, represented as *root*, where *root* $\in V$. In other words, DPR considers a single destination prefix.

Definition 10 (Paths)**.** Paths are sequences of nodes of the form: $\langle u_1 \ u_2 \ \ldots \ u_k \rangle$ where the destination node *root* is u_k. The empty path is denoted by $\langle\rangle$. A concatenation of a node u with a path Q is represented as: $P = \langle u \ Q \rangle$. A path originating from u is represented by P^u. The set of paths originating from u is represented by $\mathcal{P}^u$.

Definition 11 (Path Preferences)**.** Each node u has a unique preference over paths originating at u. This ranking is represented by the $\succeq$ operator. If u prefers P^u over Q^u then: $P^u \succeq Q^u$. If u prefers P^u over Q^u then: $P^u \succeq Q^u$. Strict preference is defined by:

$$P^u \succ Q^u \text{ iff } P^u \succeq Q^u \text{ and } Q^u \not\succeq P^u$$

For each node $u \in V$, $\succeq$ is a total order over $\mathcal{P}^u \cup \langle\rangle$. Thus each node u has an ordered preference over all its paths to *root*. If two paths start with different nodes, then they have no preference relation. Forbidden paths P are those ranked below the empty path for all times: $\langle\rangle \succ P$. All paths with repeating nodes are forbidden.

Definition 12 (DPR Instance)**.** A Dynamic Policy Routing (DPR) instance consists of a graph and a path preference $D = (\succeq, G)$.

Definition 13 (Best Paths)**.** At each time index t, every node u has a path to *root*, represented by $P^u = \pi(u,t)$. The available path choices of a node, via all possible neighbors v, are represented by Choices(u,t) where:

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

The Best(u,t) notation represents the current best path for u:

$$\text{Best}(u,t) = \max_{\succeq}\ \text{Choices}(u,t)$$

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

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

The path used by node u at time t, $\pi(u,t)$, was its best path at time $t-1$, Best$(u,t-1)$. This best path was determined using the ranking $\succeq$.

Definition 14 (Next-Hop Neighbor)**.** The ρ notation is used to represent the next-hop neighbor of a current path:

$$\rho(u,t) = \text{NextHop}(\pi(u,t))$$

Definition 15 (Realized Paths)**.** A path P^u is *realized* iff there exists a time t such that $\pi(u,t) = P^u$.

Proposition 1 (Forbidden Paths)**.** *Forbidden paths are never realized.*

Proof. Assume not. Then there exists a forbidden path P^u, a node u, and a time t such that $\pi(u,t) = P^u$. However $\langle\rangle \succ P^u$ so $P^u \neq \text{Best}(u,t-1)$ which is a contradiction. □

Proposition 2 (Path Deconstruction)**.** *If* $\rho(u_0,t) = u_1$ *then* $\pi(u_0,t) = \langle u_0\ \pi(u_1,t-1)\rangle$

Proof. By the definition of π, $\pi(u_0,t) = \text{Best}(u_0,t-1)$ so $\pi(u_0,t) \in \text{Choices}(u_0,t-1)$. So by the definition of Choices, $\pi(u_0,t) = \langle u_0\ \pi(u_1,t-1)\rangle$, where $u_1 = \rho(u_0,t)$. □

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.

#	Action(u,t)	Cause(u,t)	Condition	Explanation
1	StepUp	$v = \rho(u, t+1)$	$\pi(u,t) \prec \pi(u,t+1)$, $\rho(u,t) \neq \rho(u,t+1)$	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 $t+1$.
2	StepDown	$v = \rho(u,t)$	$\pi(u,t) \succ \pi(u,t+1)$, $\rho(u,t) \neq \rho(u,t+1)$	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 $t+1$.
3	StepSame	$v = \rho(u,t)$ $= \rho(u,t+1)$	$\pi(u,t) \neq \pi(u,t+1)$, $\rho(u,t) = \rho(u,t+1)$	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 3.1: Cases for action and causation.

3.2.3 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 Action(u,t) and Cause(u,t) are defined in Table 3.1. 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:

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

Definition 16 (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 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.

Definition 17 (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 $y_0 = y_k$.

Remark 2. In terms of the synchronicity of DPR, we show that this is not a drawback and that DPR has sufficient expressive power to model asynchronicity in Appendix A.

3.2.4 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 $\langle y_i\ y_{i+1}\ \ldots\ y_j\rangle^{t+i}$ 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 3·7, 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.

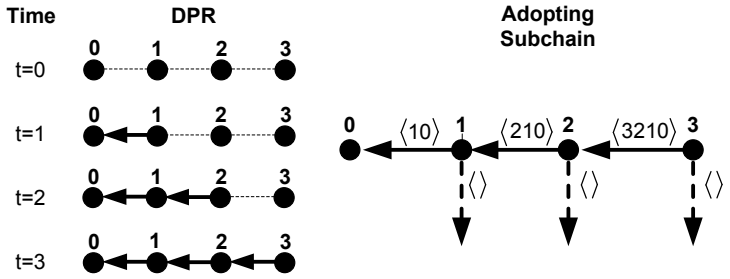


Figure 3·7: An example of an adopting subchain. Adopted/discarded paths are represented by solid/dotted arrows, respectively.

Definition 18 (Adopting Subchain)**.** An adopting subchain of Y is $\langle y_i\ y_{i+1}\ \ldots\ y_j\rangle^{t+i}$ 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 19 (Discarding Subchain)**.** A discarding subchain of Y is $\langle y_i\ y_{i+1}\ \ldots\ y_j\rangle^{t+i}$ 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 3·8 shows the alternating subchains of BAD GADGET.

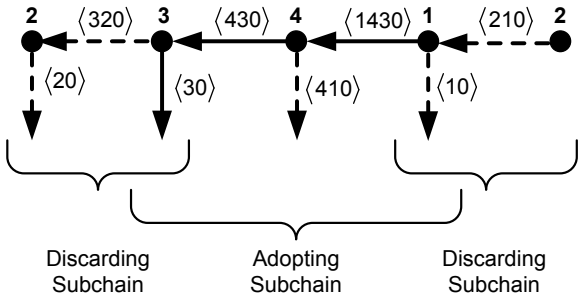


Figure 3·8: Alternating subchains of BAD GADGET. Adopted/discarded paths are represented by solid/dotted arrows, respectively. Horizontal paths are more preferred than vertical paths.

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 20 (Causation Fence)**.** A causation 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 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 3·9. 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.

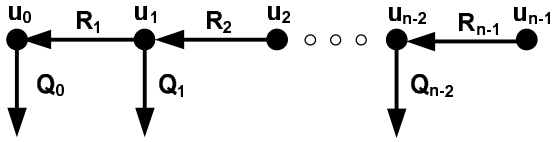


Figure 3·9: Causation fence.

Lemma 2 (Chain-Fence Relationship)**.** *Every causation chain $Y = \langle y_0 \ \ldots y_k \rangle^t$ is equal to the concatenated rim paths $R_1 \ldots R_{n-1}$ 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

$$Y^0, Y^1, \ldots Y^{n-1}$$

where each subchain Y^r is of the form

$$Y^r = \langle y_0^r \ldots y_s^r \rangle^{t^r}$$

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 3·10 shows the causation fence induced by the causation chain in Figure 3·8.

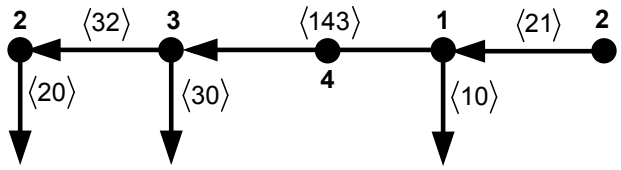


Figure 3·10: Causation fence example.

3.2.5 Dispute Wheels

Griffin *et al.* introduced dispute wheels in [Griffin et al., 2002], where their existence is a necessary condition for an SPP instance to not have a stable assignment. A dispute wheel, as shown in Figure 3·11, represents a cyclical set of path preferences.

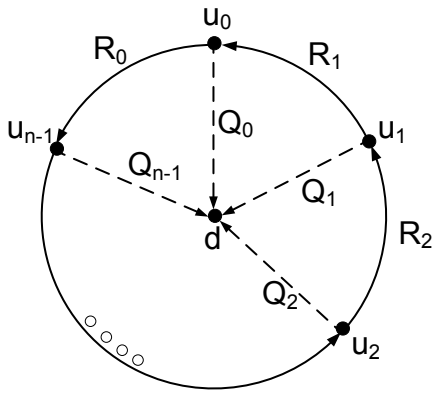


Figure 3·11: Dispute wheel.

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 3·12.

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:

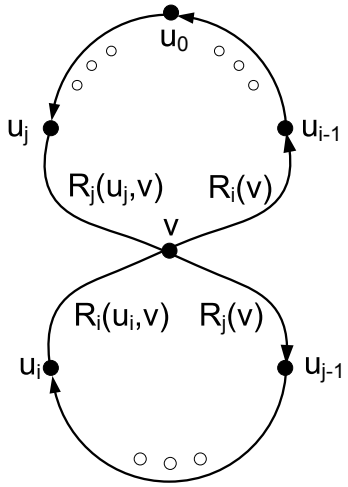


Figure 3·12: Non-proper dispute wheel. $P(a,b)$ is the subpath of P starting with a and ending with b. $P(a)$ is the subpath of P starting with a.

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

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

This results in the dispute wheel in Figure 3·13.

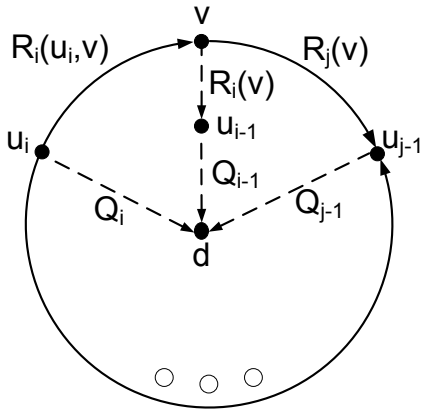


Figure 3·13: Smaller dispute wheel case 1.

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

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

This results in the dispute wheel in Figure 3·14.

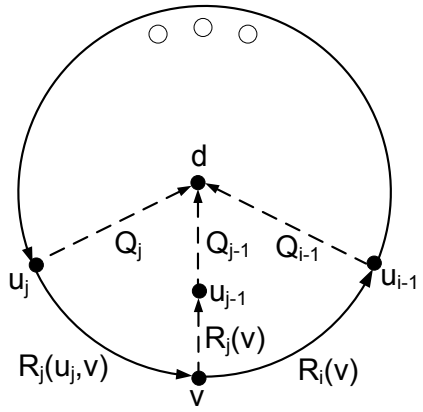


Figure 3·14: Smaller dispute wheel case 2.

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''. 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.

3.2.6 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 21 (Policy Digraph)**.** A DPR instance is defined in terms of a time-varying graph $G = (V, E)$ and a set of path preferences $\succ$. Given a DPR instance $D = (G, \succ)$, the policy digraph is denoted by $O(\succ) = (V', E')$ where each node $P \in V'$ represents a realizable path in $\succ$ 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 $O(\succ)$, all pnodes in $O(\succ)$ that are paths originating from a single node $u \in V$ 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 3·15.

It is important to note that each pnode must have an incoming subpath edge. The policy digraph in Figure 3·15 includes the destination node for our sample BAD GADGET instance. For simplicity, we omit the destination node from all our policy digraphs in the rest of this thesis.

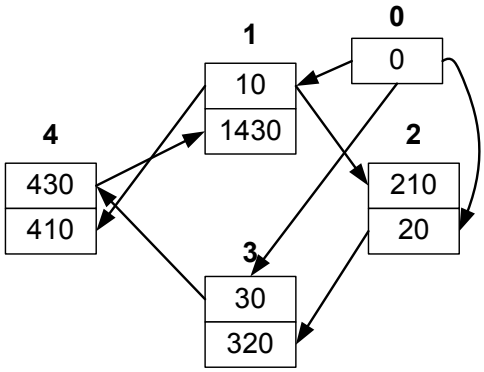


Figure 3·15: Policy digraph of BAD GADGET including the destination node 0.

Theorem 2 (Chains in Policy Digraphs)**.** *Every causation chain $Y = \langle y_0\ y_1\ \ldots y_k \rangle$ 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 3·16. This in turn implies that every causation chain Y is a path in O.

□

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.*

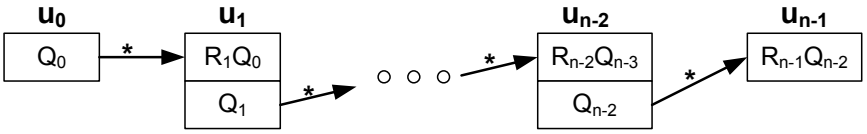


Figure 3·16: Causation fences are paths in policy digraphs. The $*$ notation implies a series of subpath edges through pnodes.

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 3·17. A sample cycle (and hence dispute wheel) could start and end at pnode R_0Q_{n-1} as follows: $\langle R_0Q_{n-1}\ \ Q_0\ \ R_1Q_0\ \ Q_1\ \ \ldots\ Q_{n-1}\ \ R_0Q_{n-1}\rangle$

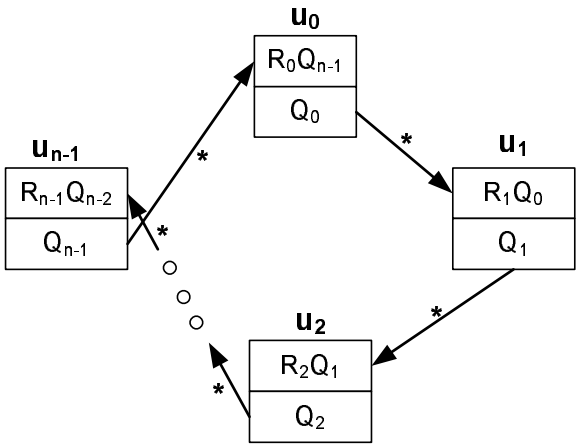


Figure 3·17: Dispute wheels are cycles in policy digraphs and vice versa.

□

Remark 3. Policy digraphs essentially complement the SPP framework and represent a novel structure for understanding and analyzing the dynamics of policy routing.

Definition 22 (Length of Policy Digraph)**.** We define the length of a policy digraph:

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

to be the number of times subpath edges are traversed in the longest path (or walk) of $O(\succ)$ with repeating nodes (*i.e.*, policy edges traversed in the walk are not accounted for in the length). This represents the longest possible causation chain. If $O(\succ)$ has a cycle, then $\text{Length}(O(\succ)) = \infty$.

3.3 Dynamic Policy Routing Model with Time-Varying Path Preferences

3.3.1 Overview

In this section we outline the DPR model with time-varying path preferences that will be used in chapter 6 to derive the properties of safe (*i.e.*, convergent) policy routing dynamics.

3.3.2 Basic Notation

Definition 23 (Path Preferences)**.** At each time t, each node u has a unique preference over paths originating at u. This dynamic ranking is represented by the $\succeq^t$ operator. If u prefers P^u over Q^u at time t then: $P^u \succeq^t Q^u$. If u prefers P^u over Q^u for all t then: $P^u \succeq Q^u$. Strict preference is defined by:

$$P^u \succ^t Q^u \text{ iff } P^u \succeq^t Q^u \text{ and } Q^u \not\succeq^t P^u$$

For all times t, for each node $u \in V$, $\succeq^t$ is a total order over $\mathcal{P}^u \cup \langle\rangle$. Thus each node u has an ordered preference over all its paths to *root*. If two paths start with different nodes, then they have no preference relation. Forbidden paths P are those ranked below the empty path for all times: $\langle\rangle \succ P$. All paths with repeating nodes are forbidden.

Definition 24 (DPR Instance)**.** A Dynamic Policy Routing (DPR) instance consists of a graph and a path preference $D = (\succeq^t, G)$.

Definition 25 (Best Paths)**.** At each time index t, every node u has a path to *root*, represented by $P^u = \pi(u,t)$. The available path choices of a node, via all possible neighbors v, are represented by Choices(u,t) where:

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

The Best(u,t) notation represents the current best path for u:

$$\text{Best}(u,t) = \max_{\succeq t} \text{ Choices}(u,t)$$

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

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

The path used by node u at time t, $\pi(u,t)$, was its best path at time $t-1$, Best$(u,t-1)$. This best path was determined using the ranking $\succeq^{t-1}$.

#	Action(u,t)	Cause(u,t)	Condition	Explanation
1	RankDec	$v = \rho(u,t)$	$\pi(u,t) \succ^t \pi(u,t+1)$	Node v was the next hop of u's chosen path at time t. However, node v changed its path at time t, causing u to choose a less preferred path at time $t+1$.
2	RankInc	$v = \rho(u,t+1)$	$\pi(u,t) \prec^t \pi(u,t+1)$	Node v advertised a new path at time t, causing u to choose a more preferred path through v at time $t+1$.
3	RankSame	$v =$ empty	$\pi(u,t) = \pi(u,t+1)$	v is empty, because u's path did not change between times t and $t+1$.

Table 3.2: Cases for action and causation.

3.3.3 Causation Chains and Cycles

Here we consider slightly different actions when constructing our causation chains and cycles. The cases for action and causation can be found in Table 3.2 where only the relative rankings of the current and new paths are considered, irrespective of the next-hop neighbor being used. Also, the path preferences here are time-varying.

A sample causation chain can be seen in Figure 3·18.

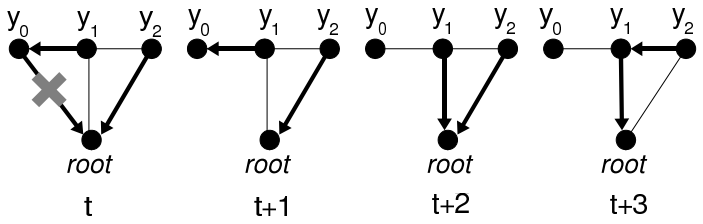


Figure 3·18: Causation chain $Y = \langle y_0\ y_1\ y_2 \rangle^t$. A link failure between y_0 and *root* occurred at time t, causing y_0 to have no path to root at time $t+1$. This causes y_1 to switch to a less preferred path at time $t+2$, where Cause$(y_1, t+1) = y_0$ with causation condition 1. This causes y_2 to switch to a more preferred path via y_1 at time $t+3$, where Cause$(y_2, t+2) = y_1$ with causation condition 2.

Definition 26 (Simple and Non-Simple Causation Cycles)**.** Given a causation chain of the form $\langle y_0\ y_1\ \ldots\ y_k\ y_{k+1} \rangle^t$, if $y_0 = y_k$ then a causation cycle $\langle y_0\ y_1\ \ldots\ y_k \rangle^t$ exists. If $y_1 \neq y_{k+1}$, then the cycle is *simple*, otherwise the cycle is *non-simple*. The following causation chains contain simple and non-simple cycles:

Simple: $\langle y_0\ y_1\ y_2\ y_0\ y_3 \rangle^t$
Non-Simple: $\langle y_0\ y_1\ y_2\ y_0\ y_1 \rangle^t$

A sample causation cycle is shown in Figure 3·19.

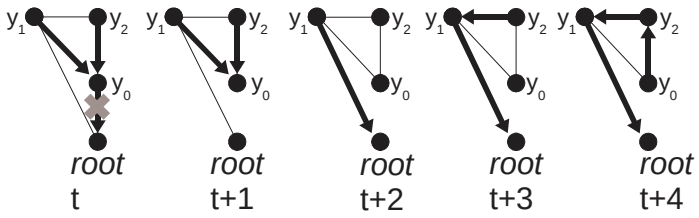


Figure 3·19: Causation cycle $Y = \langle y_0\ y_1\ y_2\ y_0 \rangle^t$. A link failure between y_0 and *root* occurred at time t, causing y_0 to have no path to root at time $t+1$. This causes y_1 to switch to a less preferred path at time $t+2$, where Cause$(y_1, t+1) = y_0$ with causation condition 1. This causes y_2 to switch to a path through y_1 at time $t+3$, where Cause$(y_2, t+2) = y_1$ with causation condition 2. The cycle is closed with y_0 switching to a path via y_2 at time $t+4$, where Cause$(y_0, t+3) = y_2$ with causation condition 2. Note the existence of a separate causation chain $Y' = \langle y_0\ y_2 \rangle^t$ when y_2 switches to the empty path at time $t+2$ with causation condition 1.

3.4 Economic DPR Model

3.4.1 Overview

In this section we model the economic constraints that are typical of commercial relationships (or agreements) between ASes in the Internet [Gao and Rexford, 2001]. We refer to routing policy instances that adhere to the Gao-Rexford guidelines as *safe* and ones that do not adhere as *potentially unsafe*. The Gao-Rexford guidelines we consider are as follows:

1. Every node is customer, peer, or provider to its neighboring nodes. The commercial agreement (*i.e.*, relationship) between any two nodes does not change over time.

2. Each node prefers a path through a customer over a path through a peer / provider and prefers a path through a peer over a path through a provider.

3. Each node provides transit service only to its customers. This is achieved by configuring the appropriate import / export policies for paths (*i.e.*, which paths are advertised to which neighbors and which paths are accepted from which neighbors). The end result of these import / export policies is that all paths are valley-free. A valley-free path consists of zero

or more customer-to-provider links followed by an optional peering link followed by zero or more provider-to-customer links.

4. A node cannot be a provider to itself. There are no customer-provider cycles. Furthermore, a node cannot be both a (direct or indirect) provider and a (direct or indirect) peer to another node as shown in Figure 3·20.

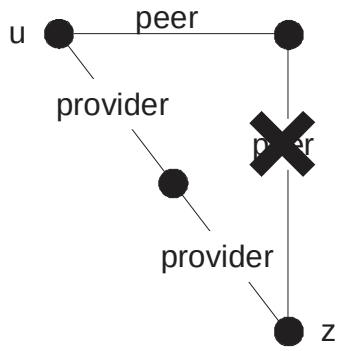


Figure 3·20: Strict economic relationships where node u cannot be an indirect provider and an indirect peer to node z. The crossed edge represents a peering edge that cannot exist in this configuration as it would make node u both an indirect provider and peer to node z.

The restrictions of the economic model, in particular the types of relationships allowed between nodes, enable equivalence classes of peers as shown in Figure 3·21.

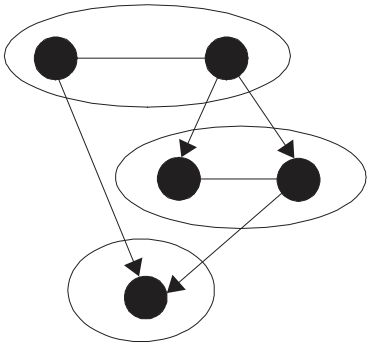


Figure 3·21: Equivalence classes of peers in economic DPR.

The economic constraints we consider are a stricter version of the Gao-Rexford guidelines which are sufficient to guarantee stability in a static graph. In particular, the Gao-Rexford guidelines have no restrictions on the relative ordering of peer and provider paths. Thus, the economic DPR model is also safe. We utilize economic DPR when considering safe policy routing instances that are guaranteed to converge. More specifically, we utilize it when we introduce economic RDMP in Chapter 4 and when we derive properties of safe policy routing dynamics in chapter 6.

3.4.2 Basic Notation

Definition 27 (Economic Operator)**.** The economic relationship between nodes are described using the operator $\succeq_\$$. This operator is essential for reasoning about the economic relationships between nodes in both paths and causation chains. A *strict* economic relation is defined by:

$$u \succ_\$ v \text{ iff } u \succeq_\$ v \text{ and } u \not\preceq_\$ v$$

and an equivalence relation is defined by:

$$u =_\$ v \text{ iff } u \succeq_\$ v \text{ and } u \preceq_\$ v$$

Economic relationships can be derived from the operator $\succeq_\$$:

- If u is a customer of v then $u \prec_\$ v$.
- If u is a provider to v then $u \succ_\$ v$.
- If u is a peer to v then $u =_\$ v$.

The properties of the economic operator $\succeq_\$$ can be modeled using *pre-order* conditions [Davey and Priestley, 2002]:

1. (reflexive) $x \succeq_\$ x$
2. (transitive) $x \succeq_\$ y$ and $y \succeq_\$ z$ implies $x \succeq_\$ z$

The following transitive relationships hold:

$$x \succ_\$ y \text{ and } y \succeq_\$ z \text{ implies } x \succ_\$ z$$

$$x \succeq_\$ y \text{ and } y \succ_\$ z \text{ implies } x \succ_\$ z$$

Definition 28 (Customer, Peer, and Provider Paths)**.** We define paths by the economic relationship between a path's starting node u and its next-hop. For all paths P^u:

$$\begin{aligned} \text{Customer}(P^u) \quad &\text{iff} \quad u \succ_\$ \text{NextHop}(P^u) \\ \text{Peer}(P^u) \quad &\text{iff} \quad u =_\$ \text{NextHop}(P^u) \\ \text{Provider}(P^u) \quad &\text{iff} \quad u \prec_\$ \text{NextHop}(P^u) \end{aligned}$$

Definition 29 (Valley)**.** We define a valley to be a sequence of three distinct nodes $\langle a\ b\ c \rangle$ satisfying the condition:

$$a \succeq_\$ b \preceq_\$ c$$

The four types of valleys can be seen in Figure 3·22. Every valley-free sequence is a series of zero or more ascending customer-to-provider relationships, followed by an optional peer relationship, followed by a series of zero or more descending provider-to-customer relationships.

Figure 3·22: Valleys

Definition 30 (Economic DPR Instances)**.** An economic DPR instance $(\succeq_\$, \succeq^t, G)$ satisfies the following conditions:

1. All paths which have a valley are forbidden (*i.e.*, are not realizable).
2. Customer paths are always preferred over peer/provider paths and peer paths are always preferred over provider paths. Thus given paths P_1^u and P_2^u:

$$\begin{aligned} \text{Customer}(P_1^u) \text{ and not Customer}(P_2^u) \quad &\Rightarrow \quad P_1^u \succ P_2^u \\ \text{Peer}(P_1^u) \text{ and Provider}(P_2^u) \quad &\Rightarrow \quad P_1^u \succ P_2^u \end{aligned}$$

Chapter 4

Minimizing Policy Routing Dynamics

4.1 Overview

In this chapter we formulate the Routing Dynamics Minimization Problem (RDMP) and show its complexity class both in the general case and under certain restrictions. The formal proofs can also be found in [Mattar et al., a, Mattar et al., 2010a]. We focus first on motivating the problem and on presenting the main intuition behind our results.

RDMP utilizes policy digraphs, a time-invariant structure which captures how routing update messages can propagate in the network, to solve a graph optimization problem. This optimization problem minimizes the longest possible sequence of routing update messages in a dynamic network by changing the path preferences of nodes. The minimization of the longest possible sequence is only one possible optimization goal that represents a simple min-max optimization. The correspondence between the length of the longest sequence and the occurrence of a dispute wheel (or policy conflict) is of particular interest to us and partly motivated this choice. If the longest possible sequence has infinite length, it implies that a dispute wheel (or policy conflict) exists. Hence, if the optimization produces an RDMP instance whose length is finite, it implies that the routing instance is safe (*i.e.*, convergent). This allows us to explore the correspondence between the length of causation chains in an RDMP instance and the safety of the selected path preferences across nodes.

Consider the policy digraph of BAD GADGET in Figure 3·6. Consider the hypothetical toy example where the path preferences of exactly *two* nodes in BAD GADGET *must* be swapped. The resulting possible policy digraphs are outlined in Figure 4·1. The optimal solution B, where the length of the longest causation chain (*i.e.*, path) is minimized, is obtained by swapping the path

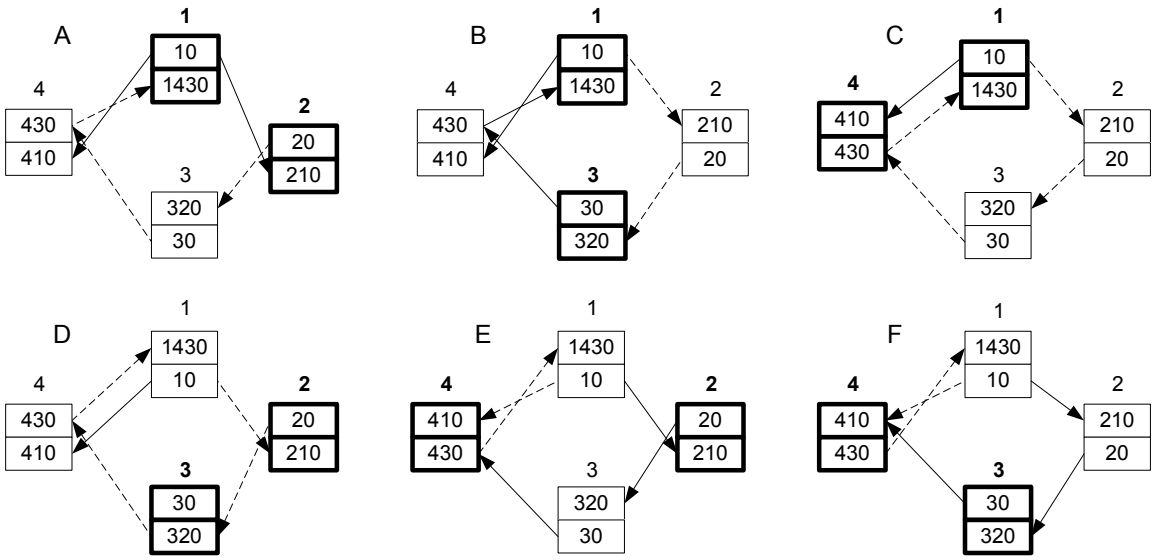


Figure 4·1: All possible policy digraphs where the path preferences of exactly two nodes are swapped. The bolded nodes represent the ones that performed a swap in their path preferences. The dashed arrows represent the subpath edges that are on the longest walk in the policy digraph. The optimal solution is B where nodes 1 and 3 swapped their path preferences and the length of the policy digraph is 2. The lengths of E and F are infinite since they have cycles.

preferences of nodes 1 and 3. *We show that finding a policy configuration which minimizes the length (*i.e.*, the size of the longest walk with possibly repeated nodes) of the policy digraph is NP-Hard.*

Even though RDMP as described in this thesis requires a centralized solver which cannot be easily realized in practice, we view RDMP as a complement to SPP. While SPP is concerned with the stable assignment of paths irrespective of the resulting routing dynamics, RDMP is concerned with optimizing the routing dynamics (by changing the path preferences) irrespective of the paths assigned. *By routing dynamics we mean the lengths of the causation chains which capture the propagation of path changes across nodes in the network.* If the routing dynamics in RDMP are optimized in such a way that the length of all causation chains (*i.e.*, paths) in the policy digraph are finite, then the policy routing instance is stable and all propagations of path changes are bounded (*i.e.*, a stable unique path assignment exists as proved in [Griffin et al., 2002]). We envision that the formulation of RDMP will allow us to explore problems where the dynamics of policy routing

can be examined. In particular, an immediate extension involves deriving policy guidelines that limit the control overhead due to the propagation of path changes in the network. Alternatively, one may develop distributed protocols that exchange diagnostic information to explicitly minimize (or reduce) the routing dynamics by having nodes dynamically adapt their routing policies. Such control-plane optimizations to reduce the overhead of routing dynamics (or the propagation of route updates) could also serve as the basis for performing better traffic engineering. In particular, the routing policies can be adapted to not only minimize the routing dynamics but may incorporate other traffic engineering objectives to increase the predictability of traffic loads and aid operators in capacity planning / dimensioning of their network [Feamster and Rexford, 2007, Quoitin et al., 2005, Uhlig and Bonaventure, 2004].

We also consider more realistic restrictions on RDMP. In particular, we consider the case where nodes abide by the Gao-Rexford guidelines that guarantee safety (*i.e.,* convergence). *We show that finding a policy configuration which minimizes the length of the policy digraph when nodes abide by the Gao-Rexford guidelines can be solved in polynomial time.* This result provides insight into possible approaches for developing distributed protocols to minimize the dynamics of policy routing in the Internet today. Such solutions, however, are outside the scope of this thesis.

While RDMP, as described here, minimizes the length of the longest path in the policy digraph, many other optimizations are possible such as the (weighted) average or median path length, the number of cycles in the graph, *etc*. Nodes in the policy digraph (representing paths) could also be labelled with weights to capture many possible metrics such as traffic load, relative path preference, probability of using that path, *etc*. This thesis focuses on formalizing an unlabelled policy digraph and on minimizing the length of the longest path (*i.e.,* a standard min-max optimization problem). Solving particular RDMP instances with other primary objectives (*e.g.,* traffic engineering requirements) is outside the scope of this thesis.

The rest of this chapter is organized as follows. Section 4.2 formalizes RDMP and proves it is NP-Hard. Section 4.3 formalizes Economic RDMP (*i.e.,* RDMP instances that abide by the Gao-Rexford guidelines) and proves that it can be solved in polynomial time.

4.2 Routing Dynamics Minimization Problem

In this section we formalize the Routing Dynamics Minimization Problem (RDMP). RDMP is an optimization problem—its goal is to change the path preferences of a subset of nodes to minimize the worst case routing dynamics regardless of changes to the underlying topology (*e.g.,* link failures). While SPP is concerned with the stable assignment of paths irrespective of the resulting routing dynamics, RDMP is concerned with minimizing the routing dynamics (by changing the path preferences) irrespective of which paths are assigned.

4.2.1 Formal Definition

Let a set of nodes in a network be denoted by V. Let Ω be a set of path preferences for this set of nodes V. In other words, each $\succ \in \Omega$ represents the path preferences across every node $v \in V$. Each path preference $\succ \in \Omega$ has a corresponding policy digraph $O(\succ)$, representing all possible causation chains over the set of nodes V. The set Ω reflects the degree of flexibility in changing the nodes' preferences. An example Ω is where only the path preferences of a single node $v \in V$ can be changed and the path preferences of all other nodes are fixed. RDMP is thus formally defined by (V, Ω). The goal of RDMP is to find the path preferences $\succ^* \in \Omega$ which will minimize the worst case routing dynamics represented by the policy digraph:

$$\succ^* \quad = \quad \arg\min_{\succ \in \Omega} \text{Length}(O(\succ))$$

Note that $\text{Length}(O(\succ))$ represents the length of the longest path (*i.e.,* causation chain) in $O(\succ)$. Thus RDMP represents a framework to tailor preferences of nodes in a network to minimize the worst case routing dynamics regardless of underlying time-varying topology.

4.2.2 Sample Instance

RDMP formalizes the example in Figure 4·1 where exactly two nodes must swap their path preferences. In that example, $|\Omega| = 6$ which each $\succ \in \Omega$ resulting in a particular policy digraph. The solution to this problem is configuration B where $\text{Length}(O(\succ^*)) = 2$. In general Ω can be used to formalize other conditions (or restrictions) on the path preferences of nodes. For example, Ω

can be used to encode popular constraints such as the Gao-Rexford conditions [Gao and Rexford, 2001] where AS path preferences abide by the commercial / economic agreements.

4.2.3 Complexity

Theorem 4. *RDMP is NP-Hard.*

Proof. The RDMP problem is given by a set of nodes in a network V and an allowable set of path preferences Ω across all the nodes $v \in V$: (V, Ω).

Overview of reduction from MAXSAT*:*

To prove that RDMP is NP-Hard, a reduction from MAXSAT is required. The MAXSAT problem consists of a set of variables $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ and a collection of clauses $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$, consisting of disjunctions of three literals (a literal X is a variable x or its negation $\overline{x}$). The goal of MAXSAT is to determine a truth assignment in $\mathcal{X}$ which maximizes the number of satisfied clauses. The MAXSAT problem is known to be NP-Hard. Without loss in generality we assume that each clause C consists of literals corresponding to exactly three variables.

An example of the MAXSAT problem with three clauses and three literals is as follows:

$$\begin{aligned} C_1 &= X_1 \cup X_2 \cup \overline{X}_3 \\ C_2 &= \bar{X}_1 \cup \overline{X}_2 \cup X_3 \\ C_3 &= X_1 \cup X_2 \cup X_3 \end{aligned}$$

An assignment $\mathcal{X} = \{x_1, x_2, x_3\} = \{1, 0, 0\}$ results in all three clauses being satisfied.

Given an algorithm A which solves the RDMP problem, there is a polynomial time algorithm B which uses A to solve MAXSAT*. This reduction implies that RDMP is NP-Hard.*

We can assume that algorithm A receives as input a set of nodes V and a set of path preferences Ω. Algorithm A then returns path preferences $\succ^* \in \Omega$ whose corresponding policy digraph has a minimal length. Algorithm B takes as input an instance I of MAXSAT and constructs a set of nodes V and a set of path preferences Ω. Algorithm B then utilizes A to return the optimal path preferences $\succ^* \in \Omega$ which can then be converted to a solution to the instance I of MAXSAT.

Algorithm B will construct a series of "structures" dependent on the input I. Each structure represents one or more nodes in the set V that will be sent to algorithm A. Each structure also represents the path preferences for those particular nodes. Thus for each node $v \in V$ represented by a structure, a set of path preferences Ω_v is defined. Algorithm B constructs multiple interconnected structures to create a final "gadget". This gadget is converted into a resultant set of path preferences Ω which represents the permutation of all possible path preferences Ω_v across the nodes in V. Finally, algorithm A is used to solve the RDMP problem on the input tuple (V, Ω). This output

from algorithm A is used to obtain a solution to the instance I of MAXSAT. First we must define the set of required structures. Instead of describing the preferences Ω_v directly, we describe the resultant set of policy digraphs.

Continuation structure:

The continuation structure shown in Figure 4·2 has 2 input links and 2 output links where any input link can be connected to any output link. In other words, input link e_1 can be connected to output links e_3 or e_4. While the example shows two input links and two output links, in general the continuation structure can have m input links and k output links.

The continuation structure maps to a single policy digraph. In particular it represents a single network node n that has fixed path preferences where $|\Omega_n| = 1$. The policy digraph is a single stacked pnode n. For example the connection e_1 to e_3 in the continuation structure is mapped to a causation chain from subpath edge e_1 to subpath edge e_3. This is possible because the policy digraph has implicit policy edges from each pnode to the one below it within a stacked pnode. In general there are causation chains from subpath edges e_1 or e_2 to subpath edges e_3 or e_4. The continuation structure will be used in the construction of the overall gadget to obtain a solution to the instance I of MAXSAT as we will see.

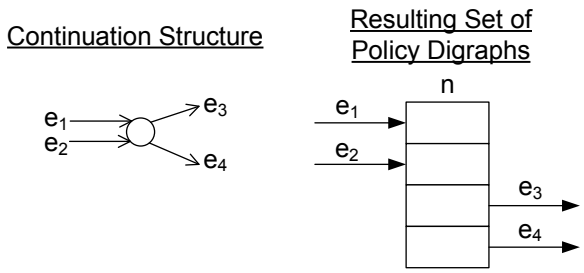


Figure 4·2: Continuation structure and its resulting policy digraph.

Switch structure:

The switch structure is outlined in Figure 4·3. This structure maps to a single network node n. Node n has four available paths and two available path preferences, with $|\Omega_n| = 2$. For the first path preference $\succ_1 \in \Omega_n$, there is a path from subpath edge e_1 to subpath edge e_3 in its policy digraph. Similarly, for the second path preference $\succ_2 \in \Omega_n$, there is a path from subpath edge e_2 to subpath edge e_4. If algorithm A returns a solution containing policy preference $\succ_1$, then we say A chooses to connect structure links e_1 to e_3 of the switch. Similarly for the choice of $\succ_2$ it represents a connection between e_2 and e_4.

Variable structure:

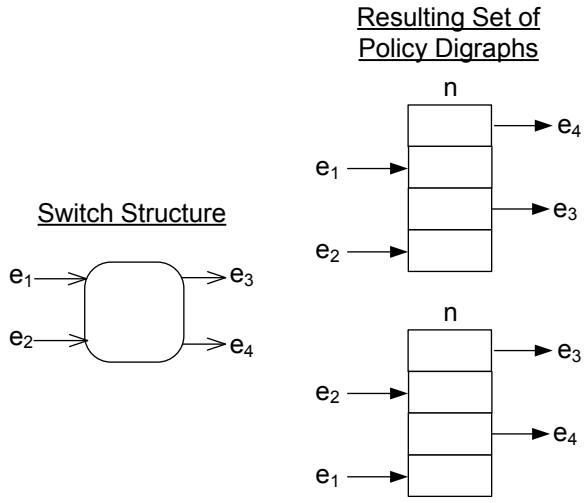


Figure 4·3: Switch structure and its resulting policy digraph for each $\succ \in \Omega$.

For each variable $X \in \mathcal{X}$ of the MAXSAT instance I, algorithm B creates a variable structure, shown in Figure 4·4. This structure is composed of three continuation structures and a switch. If A connects link e_1 to link T we say that A sets the variable X to TRUE. Otherwise if A connects link e_1 to link $\bar{F}$ we say that A sets the variable X to FALSE. If the input instance I of MAXSAT has k variables, then exactly k variable structures will be created.

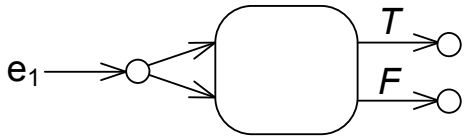


Figure 4·4: Variable structure.

Literal structure:

Each clause C consists of literals corresponding to three variables. For example $C = X \cup \overline{Y} \cup \overline{Z}$ contains three literals: one positive, X, and two negative, $\overline{Y}$ and $\overline{Z}$. For each literal X or $\overline{X}$ in a clause, B constructs a literal structure consisting of a single switch for the literal X that is connected to the corresponding variable structure x as shown in Figure 4·5 for a positive literal.

We say positive literal structures are satisfied if A assigns the variable structure to be TRUE. We say negative literal structures are satisfied if A assigns the variable structure to be FALSE. If a literal structure is satisfied, then we can safely assume A will choose not to connect e_1 to e_2. This is because in the construction of our overall gadget later on we will ensure that the causation chain leading up to e_1 will be large enough to discourage A from making such a choice. If a literal structure is not satisfied, we can assume A will connect e_1 to e_2 to avoid creating a cycle. For

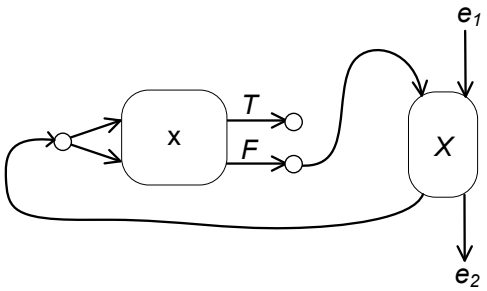


Figure 4·5: Literal structure for a positive literal. The variable structure x is on the left while the switch structure X is on the right. A negative literal structure would connect to the T link of the variable structure.

example if the literal structure in Figure 4·5 is not satisfied then the variable x will be assigned FALSE. If e_1 is not connected to e_2 there would be a cycle (*i.e.*, a path of infinite length). Hence, A will avoid the cycle (in an attempt to minimize the length of the policy digraph) by connecting e_1 to e_2.

Clause structure:

For each clause $C \in \mathcal{C}$ algorithm B constructs a clause structure consisting of three literal structures as shown in Figure 4·6. We say A satisfies a clause structure if and only if it satisfies at least one literal structure. It follows that if a clause is satisfied there is no connection between e_1 and e_2. If a clause is not satisfied then there is a connection between e_1 and e_2. This follows directly from the definition of a literal structure. If the input instance I of MAXSAT has m clauses, then exactly m clause structures will be created.

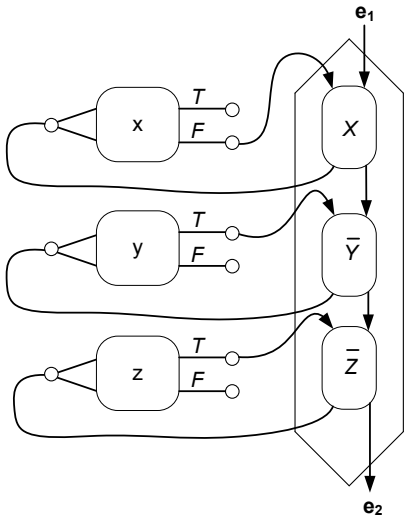


Figure 4·6: Clause structure corresponding to the clause $C = X \cup \overline{Y} \cup \overline{Z}$.

Overall gadget to solve MAXSAT*:*

All the links between the clause structures are connected together as shown in Figure 4·7. The boxes represent a path of pnodes of some length **L**. The length **L** is large enough to force the longest possible path (*i.e.,* causation chain) to contain such a box. Each clause structure that is satisfied implies there is no connection between e_1 and e_2 (as shown in Figure 4·6). Thus the longest path will not traverse the box and the clause structure associated with it. Instead the longest path will will traverse the continuation structures at the top. For every clause structure that is not satisfied, the length of the longest path will increase by **L** as it includes the box. The size of the longest possible path is equal to $sL + \epsilon$, where s is the number of unsatisfied clause structures and ϵ is a term not dependent on L. Thus for large enough L, ϵ can be made irrelevant to A's solution. Since algorithm A chooses path preferences that minimize the length of the longest possible path in the policy digraph, ($sL+\epsilon$), this is equivalent to A satisfying the largest number of clause structures in the gadget. Hence algorithm B can use algorithm A to maximize the number of satisfied clauses. This implies that RDMP is NP-Hard.

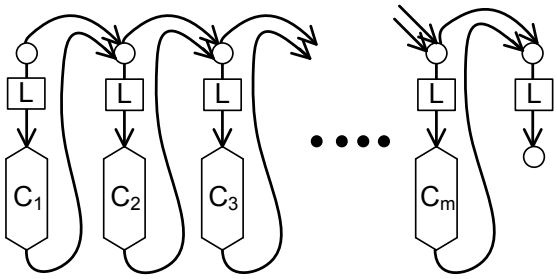


Figure 4·7: Gadget to solve MAXSAT.

□

4.3 Classes of RDMP

We have defined the overall space of RDMP instances and have shown the complexity of RDMP to be NP-Hard. One may, however, put more realistic restrictions to focus on particular RDMP instances that may provide insight into the structure of policy routing dynamics in the Internet today. In particular, a visual representation of a few sample classes of RDMP instances are outlined in Figure 4·8.

One may consider RDMP instances that are cycle-free. By cycle-free we mean RDMP instances defined by Ω where every possible policy digraph, $O(\succ)$, for every $\succ \in \Omega$ has finite length.

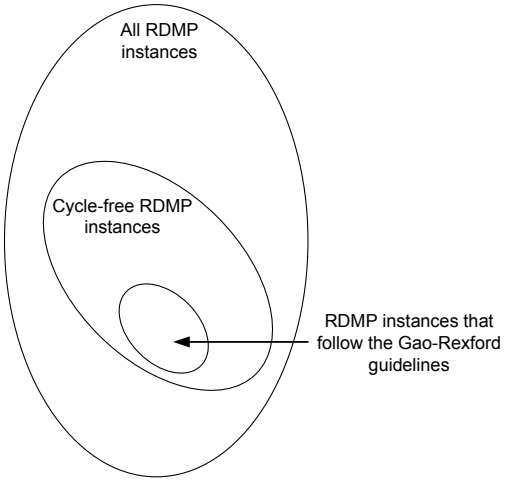


Figure 4·8: The space of RDMP instances.

This essentially implies that $O(\succ)$ for any random $\succ \in \Omega$ contains no cycles between pnodes. An example of a finite length policy digraph is shown in Figure 4·9 (Left) and an example of an infinite length policy digraph is shown in Figure 4·9 (Right).

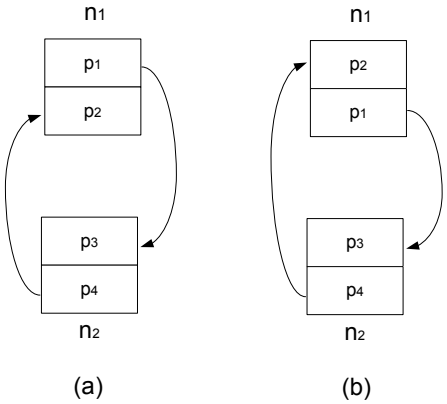


Figure 4·9: A policy digraph consisting of two stacked pnodes with a finite length of 2 (Left) and a policy digraph consisting of two stacked pnodes with infinite length (Right).

One may also consider instances where the restrictions on allowable path preferences abide by the Gao-Rexford guidelines [Gao and Rexford, 2001]. The Gao-Rexford guidelines consider the commercial / economic relationships between ASes to restrict the path preferences of nodes in such a way as to guarantee safety. One particular guideline, for example, is that ASes must

prefer customer paths over peer / provider paths. RDMP instances that follow the Gao-Rexford guidelines must also be cycle-free. Otherwise there exists a path preference $\succ \in \Omega$ that follows the Gao-Rexford guidelines but still contains a dispute wheel which would contradict the safety of the guidelines as proved in [Gao and Rexford, 2001].

4.3.1 Simple RDMP

RDMP instances without cycles guarantee routing safety for any solution. We are interested in further classifying RDMP instances by the level of constraints on the nodes. For example, the RDMP instance in Figure 4·1 describes a constraint where exactly two nodes in the network must swap their preferences. We define *Flexible* RDMP instances to have no constraints, where each node can freely choose to swap its non-forbidden paths independently.

Definition 31 (Flexible RDMP)**.** An RDMP instance (V, Ω) is flexible if there is a set of allowable paths $\mathcal{P}$ and Ω represents every possible ordering of $\mathcal{P}$. This implies that all non-forbidden paths within every stacked pnode are interchangeable.

Definition 32 (Simple RDMP)**.** An RDMP instance (V, Ω) is simple if it is flexible and cycle-free. Thus, the policy digraphs, $O(\succ)$, of every preference $\succ \in \Omega$ have finite length.

If an RDMP instance is *simple*, every node is flexible to change its policies and no cycles exist. Thus all simple RDMP instance represent safe policy routing dynamics (*i.e.,* instances that do not contain cycles or dispute wheels). Instances that are dispute wheel free represent easy instances of the Stable Path Problem, in that there is a single unique stable path assignment. Such instances are interesting as they represent the type of dynamics that we strive for: safe dynamics that converge. All simple RDMP instances are flexible, implying that there are no inter and intra node constraints to hinder the minimization of the routing dynamics.

The complexity of this class of problems are of particular interest as they constitute the simplest class of routing dynamics against which comparisons can be made. We show that minimizing the dynamics of simple RDMP instances can be done in polynomial time.

Theorem 5. *Let $\succ^*$ be a solution to a simple RDMP instance (V, Ω). Then there exists a longest causation chain of the policy graph $O(\succ^*)$ that consists only of subpath edges.*

Proof. Assume not. Let $\mathcal{Y}$ be the set of longest causation chains of $O(\succ^*)$. Based on the assumption, every causation chain $Y \in \mathcal{Y}$ must have at least one policy edge. We choose $Y \in \mathcal{Y}$ such that the number of subpath edges before its first policy edge is maximized over $\mathcal{Y}$. Let n represent the stacked pnode through which the first policy edge of Y traverses, as shown in Figure 4·10. Causation chain Y traverses stacked pnode n from subpath edge a to subpath edge d via one (or more) policy edges.

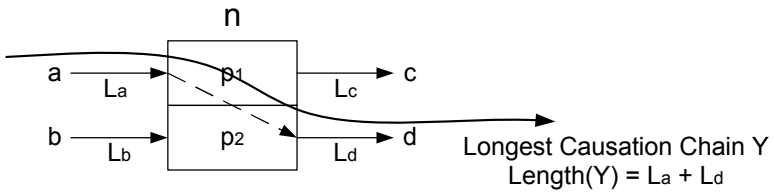


Figure 4·10: The stacked pnode n through which the longest causation chain Y traverses. Links a and b are incoming subpath edges, while links c and d are outgoing subpath edges. Causation chain Y traverses stacked pnode n from subpath edge a to subpath edge d.

Let L_a and L_b represent the lengths of the longest incoming causation chains via subpath edges a and b, respectively. Similarly, let L_c and L_d represent the lengths of the longest outgoing causation chains via subpath edges c and d, respectively. For causation chain Y to be the longest causation chain traversing stacked pnode n, the following inequalities must hold:

$$\begin{aligned} L_a + L_c &\leq L_a + L_d \\ L_b + L_d &\leq L_a + L_d \end{aligned}$$

Therefore $L_c \leq L_d$ and $L_b \leq L_a$. Furthermore it must be that $L_c < L_d$. Otherwise if $L_c = L_d$, then there is a maximum causation chain $Y' \in \mathcal{Y}$ that uses subpath edges a and c such that its first policy edge occurs at a later stacked pnode. This contradicts the assumption that Y contains the maximum number of subpath edges before the first policy edge is traversed.

Since Ω is simple, there exists another preference $\succ' \in \Omega$ identical to Y but with the paths of the stacked pnode n swapped as shown in Figure 4·11. This creates a maximum causation chain of length $L_b + L_c < L_a + L_d$. Thus $\succ^*$ is not optimal as the length of the longest causation chain was reduced. This causes a contradiction and completes the proof.

□

Theorem 6. *Simple RDMP instances can be solved in polynomial time.*

Proof. Let Ω be a simple RDMP instance for nodes V of a network. We describe a polynomial time algorithm to find a preference $\succ^* \in \Omega$ whose resulting policy digraph $O(\succ^*)$ has minimized length.

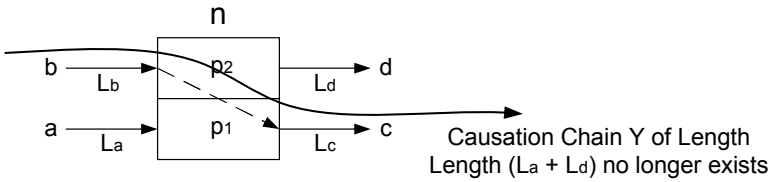


Figure 4·11: The stacked pnode n after pnodes p_1 and p_2 are flipped. Links a and b are incoming subpath edges, while links c and d are outgoing subpath edges. The longest causation chain Y of length $(L_a + L_d)$ no longer exists.

We start with a random preference $\succ \in \Omega$ and compute its corresponding policy digraph $O(\succ)$, as well as its longest causation chain Y^*. This can be done in polynomial time since $O(\succ)$ has no cycles. If Y^* consists solely of subpath edges then the algorithm stops and outputs $\succ$. This is an optimal solution based on the result of Theorem 5.

Otherwise we select a stacked pnode $n \in O(\succ)$ at which a policy edge of Y is traversed. This policy edge is from pnode p_1 to policy edge p_2. We perform a preference swap of pnodes p_1 and p_2 in n which Y^* traverses. This new policy graph has a corresponding preference $\succ' \in \Omega$ since the RDMP instance is simple. An example is shown in Figure 4·11 where pnodes p_1 and p_2 are swapped. The longest causation chain Y^* of length $L_a + L_d$ no longer exists since the swapping operation removes the policy edge between subpath edge a and subpath edge d. While it does create a new causation chain of length $L_b + L_c$, we know that $L_b + L_c < L_a + L_d$. Furthermore this swapping operation does not create any new causation chains of length $L_a + L_d$.

This operation will occur only $O(m^3)$ times, where m is the number of nodes in V. This is because after the swap occurs, nodes p_1 and p_2 will not be swapped again for a causation chain of length $L_a + L_d$. Thus nodes p_1 and p_2 will be swapped once for every maximal causation chain length $L_a + L_d$. So p_1 and p_2 will be swapped at most m times, since $L_a + L_d \leq m$. There are $O(m^2)$ pairs of pnodes, hence the swapping operation will occur only $O(m^3)$ times. Thus there is a polynomial time algorithm which always returns an optimal solution to a simple RDMP instance. □

4.3.2 Economic RDMP

Next we consider RDMP instances that abide by the Gao-Rexford guidelines that guarantee safety [Gao and Rexford, 2001].

Definition 33 (Economic RDMP)**.** An RDMP instance $(V, \Omega, \succeq_\$)$ is economic if there is a set of allowable valley-free paths $\mathcal{P}$ and Ω represents every possible ordering of $\mathcal{P}$ consistent with the Gao Rexford guidelines and the economic relationships of $\succeq_\$$.

Theorem 7. *Let $\succ^*$ be a solution to an Economic RDMP instance $(V, \Omega, \succeq_\$)$. Then there exists*

a longest causation chain of the policy digraph $O(\succ^*)$ that consists only of subpath edges and policy edges between pnodes representing paths of different types based on $\succeq_\$$.

Proof. This proof proceeds in a similar fashion to the proof in Theorem 5.

Assume not. Let $\mathcal{Y}$ be the set of longest causation chains of $O(\succ^*)$. Based on the assumption, every causation chain $Y \in \mathcal{Y}$ now has at least one policy edge between pnodes representing paths of the same type based on $\succeq_\$$. We again choose $Y \in \mathcal{Y}$ such that the number of subpath edges before this first policy edge is maximized over $\mathcal{Y}$. Causation chain Y must have a policy edge traversing at least one stacked pnode n as shown in Figure 4·12. The pnodes within the stacked pnode n represent paths of three distinct types: customer, provider and peer paths. In general there are two types of policy edges: policy edges between pnodes representing paths of the same type and policy edges between pnodes representing paths of different types. The policy edges we are considering here must be between pnodes representing paths of the same type. In Figure 4·12 the longest causation chain Y traverses stacked pnode n from subpath edge a to subpath edge d via a policy edge between two pnodes representing customer paths. Note that this is the first occurrence of this type of policy edge in Y.

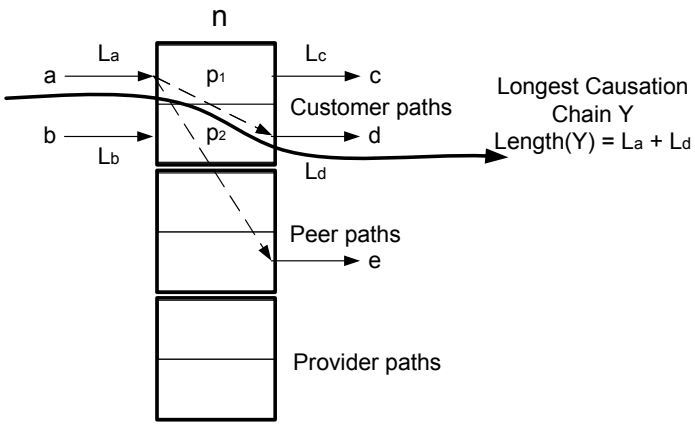


Figure 4·12: The stacked pnode v through which the longest causation chain Y traverses. Links a and b are incoming subpath edges, while links c and d are outgoing subpath edges. Causation chain Y traverses pnode v from subpath edge a to subpath edge d.

Using an argument similar to the one in Theorem 5, the same inequalities hold and there exists another preference $\succ' \in \Omega$ that is identical to Y but with the paths of the stacked pnode v swapped as shown in Figure 4·13. This creates a causation chain of length $L_b + L_c < L_d + L_a$. Thus $\succ^*$ is not optimal, causing a contradiction. Note that the swapped pnodes represent paths of the same type and hence can be swapped. Thus, all causation chains containing policy edges between pnodes representing paths of the same type can be eliminated. This only leaves causation chains consisting of subpath edges and policy edges between pnodes representing paths of different types.

Note that pnodes representing paths of different types cannot be swapped as such a swap would violate the Gao-Rexford guidelines since all customer paths must be preferred over peer paths and all peer paths must be preferred over provider paths.

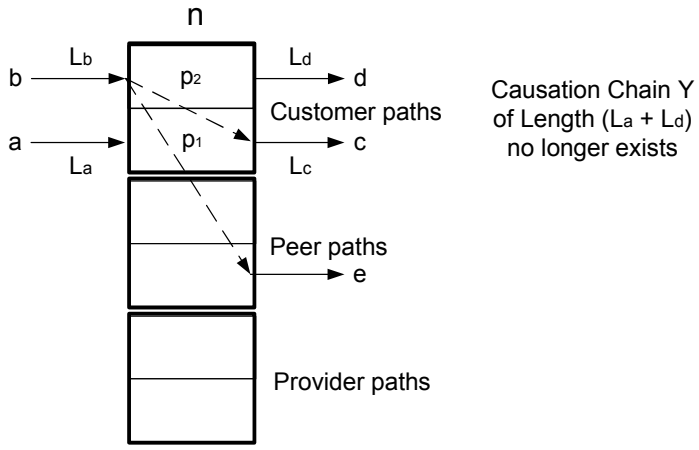


Figure 4·13: The stacked pnode v where two pnodes representing customer paths are swapped thus eliminating the longest causation chain Y. Links a and b are incoming subpath edges, while links c and d are outgoing subpath edges.

□

Theorem 8. *Economic RDMP instances can be solved in polynomial time.*

Proof. The proof is identical to the one presented for Theorem 6. The only difference is that path preference swaps can only occur between pnodes that represent paths of the same type in order to abide by the Gao-Rexford guidelines. Hence the only difference is with a node's flexibility in setting its routing policies. □

Chapter 5

Detecting Policy Conflicts

5.1 Overview

In this chapter we derive all the required theoretical results to develop our detector for policy conflicts. We also provide pseudocode for SAFETYPULSE—our token-based distributed algorithm for detecting policy conflicts in any dynamic network. SAFETYPULSE diagnoses and monitors the health of the network by detecting policy conflicts that could potentially lead to unbounded routing dynamics (*i.e.*, protocol divergence) in realtime. The formal proofs and algorithm specification can be found in [Mattar et al., b, Mattar et al., 2010b]. We focus here on motivating the problem and on presenting the main intuition behind our results.

Detecting policy conflicts has been a long-standing problem in policy routing [Varadhan et al., 1996]. Policy conflicts induce unhealthy (and unnecessary) routing dynamics that should (and could) be avoided or eliminated. The existence of policy conflicts indicates that routers may not be able to agree on any stable path assignment and BGP could potentially be divergent (*i.e.*, could lead to potentially unbounded routing dynamics). Such routing oscillations are hard to diagnose in realtime as path changes do occur in the network and it is often hard to classify a route flap as a result of transient routing dynamics or a persistent policy conflict. This is especially true if the conflict is among many nodes in the network that are distributed geographically and are managed by many independent entities. This reduces QoS predictability, increases delay variability, causes service disruption, and increases packet loss [Labovitz et al., 2000].

We utilize causation fences, a time-invariant structure which under certain conditions represents a dispute wheel, to 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.

Our theoretical results are outlined in Figure 5·1. We utilize these theoretical results to develop SAFETYPULSE—our token-based distributed algorithm that is both provably correct (as SPVP [Griffin and Wilfong, 2000]) and computationally efficient (as some of the light-weight heuristic approaches [Yilmaz and Matta, 2007, Cobb and Musunuri, 2004]). In particular, we identify the root cause of a causation cycle as either a transient route flap or a policy conflict. SAFETYPULSE has several characteristics, namely, it is computationally efficient (a constant factor reduction in message size and number of messages when compared to SPVP), provably correct, and backwards compatible. More specifically in terms of efficiency, SAFETYPULSE requires each node to append only 2 bits alongside each message update. Thus, for an n-node causation chain, an overhead of $2n$ bits is incurred, compared to xn bits in SPVP where $x >> 2$ represents the number of bits required to encode the history information that needs to be appended by each node. Also, SAFETYPULSE reduces the number of messages required to detect a policy conflict by at least a factor of 2 as we will see.

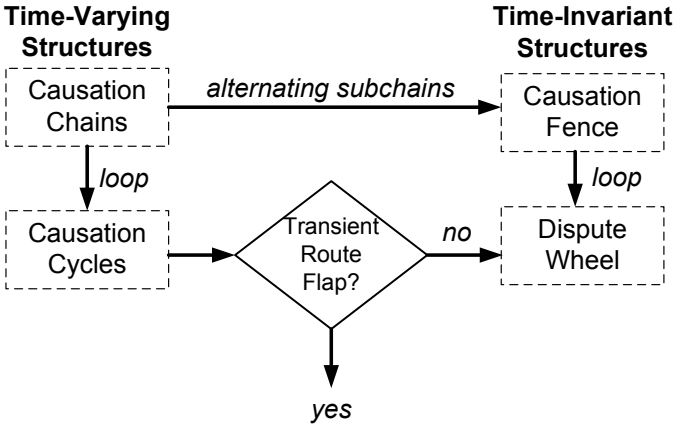


Figure 5·1: Overview of the theoretical results underlying our conflict detection algorithm.

One could also visualize our conflict detection algorithm using policy digraphs. Consider the following sample walk in the policy digraph of BAD GADGET in Figure 3·6:

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

It is easy to see that 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 exists. In the sample walk above, 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.

Our policy digraphs also provide intuition into the operation of 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 [Griffin and Wilfong, 2000] 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 routing update messages. To detect the cycle from our sample walk, SPVP encodes the exchanged path histories as:

$$\langle +20 \rangle \langle +320 \rangle \langle -430 \rangle \langle -1430 \rangle \langle +210 \rangle \langle -320 \rangle$$
$$\langle +430 \rangle \langle +1430 \rangle \langle -210 \rangle$$

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 as we can see from our sample walk.

SAFETYPULSE, our token-based distributed algorithm, leverages our theoretical results to construct the most generalized detector for policy conflicts. It is the most generalized detector because a node does not need to flap on the *same path* two (or more) times for a conflict to be detected as in SPVP [Griffin and Wilfong, 2000]. 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 which one of its paths is part of (or is involved in) the cycle of route updates propagating in the network. Such is the type of information that must be encoded in SAFETYPULSE's token.

One could also use policy digraphs to synthetically construct policy routing instances with more complex routing dynamics. Consider a walk in Figure $5{\cdot}2$ 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 and in turn causes it to get adopted. Since nodes A, B and C are involved in a policy conflict, path a_2 will get withdrawn after another cycle of updates between nodes A, B and C. The more preferred path a_3, however, may be adopted via another cycle of updates through nodes A, D and E. In this case if the network stabilizes, no policy conflict will be detected if the detection criteria is a single path being adopted and discarded. This highlights that it could potentially take a long time for a node to adopt and discard a path. It is also important to note that any changes in the underlying topology could stop the propagation of path changes (since some paths may not be available to continue inducing path changes across nodes) making policy conflicts harder to detect. That is precisely why the condition used by SAFETYPULSE to detect a policy conflict reduces the number of messages required by at least a factor of 2 when compared to SPVP.

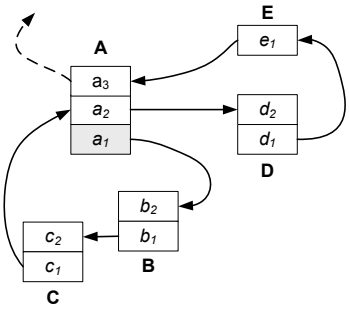


Figure 5·2: Sample policy digraph.

The rest of this chapter is organized as follows: Section 5.2 utilizes DPR to derive all our theoretical results that will serve as the foundation for our SAFETYPULSE algorithm in Section 5.3.

5.2 Detecting Dispute Wheels

Once a causation cycle $Y = \langle y_0\ y_1 \ldots y_k \rangle^t$ 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) 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 5·3. 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 9, or the occurrence of a transient route flap in Theorem 10. Finally, we outline how y_0 could theoretically infer the existence (or lack thereof) of dispute wheels.

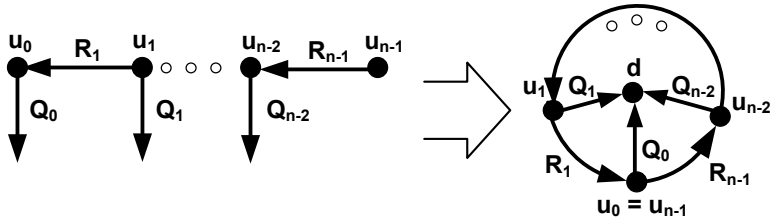


Figure 5·3: If $u_0 = u_{n-1}$ and $Q_0 \prec R_{n-1}Q_{n-2}$ then a causation fence is a dispute wheel.

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 5·3. 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 5·3, 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 5·3, 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 9 (Dispute Wheel Inference)**.** *Given a causation cycle Y, such that $Y = \langle y_0\ y_1\ \ldots\ y_k \rangle^t$ 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 10 (Route Flap Inference)**.** *Given a causation cycle Y, such that $Y = \langle y_0\ y_1\ \ldots\ y_k \rangle^t$ 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 9 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. □

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.

Remark 4. 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 it needs to determine path Q_0 that triggered the cycle and path $R_{n-1}Q_{n-2}$ that it had when the cycle was detected. Even if node y_0 performs other actions due to other causation chains propagating in the network, the relative ranking of paths Q_0 and $R_{n-1}Q_{n-2}$ is still sufficient to infer the existence of a dispute wheel.

5.3 SAFETYPULSE

Dispute wheels may result in protocol divergence. 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 [Gao and Rexford, 2001], 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.

5.3.1 Overview

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 9 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 5·4.

1. Sending out token with ProcessNode()

2. Computing time offset with SetTimeOffset()

3. Receiving token with DetectDisputeWheel()

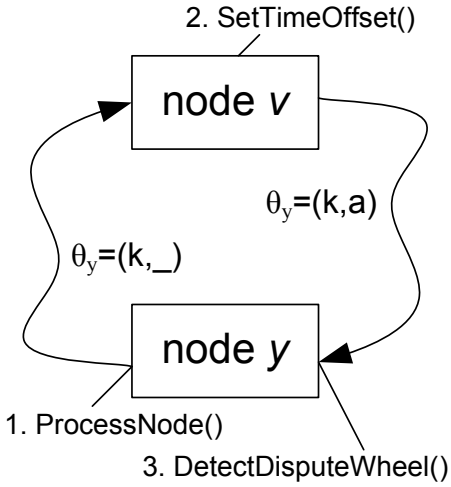


Figure 5·4: Overview of SAFETYPULSE algorithm.

5.3.2 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 $\theta_y = (k, \_)$, 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:

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

```
1: function PROCESSNODE(y, t)
2:     Best(y, t) ← max_≻ Choices(y, t)
3:     π(y, t + 1) ← Best(y, t)
4:     θ_y ← ∅
5:     if π(y, t) ≠ π(y, t + 1) then
6:         k ← new key
7:         Store(k, (π(y, t), π(y, t + 1)))
8:         θ_y ← (k, _)
9:         M(y, t + 1) ← M(Cause(y, t), t) + θ_y
```

Figure 5·5: SAFETYPULSE token creation and action storage.

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 5·5.

5.3.3 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 9 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 5·6.

5.3.4 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

```
1: function DETECTDISPUTEWHEEL(y, t)
2:     if θ_y ∈ M(Cause(y, t), t) then
3:         θ_y = (k, a)
4:         (P_1, P_2) ← Lookup(k)
5:         if a = 0 then
6:             P_test ← P_1
7:         else
8:             P_test ← P_2
9:         if Action(y, t) = StepDown then
10:            b ← 0
11:        else
12:            b ← 1
13:        if P_test ≺ π(y, t + b) then
14:            ReportDisputeWheel(P_test, π(y, t + b))
```

Figure 5·6: SAFETYPULSE token receival and dispute wheel detection.

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 5·7 shows how third-party nodes can fill in the time offset a.

```
1: function SETTIMEOFFSET(y, t)
2:     for all unclassified θ_v = (k, _) ∈ M(y, t + 1) do
3:         if Action(y, t) = StepUp then
4:             θ_v ← (k, 1)
5:         else if Action(y, t) = StepDown then
6:             θ_v ← (k, 0)
```

Figure 5·7: SAFETYPULSE time offset computation.

5.3.5 Complete Algorithm

The complete SAFETYPULSE algorithm is outlined in Figure 5·8. Each node y at time t simply executes the three algorithms described above.

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

Figure 5·8: SAFETYPULSE algorithm.

5.3.6 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, $b_0 b_1$. 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 $a = b_0 b_1$ as follows: set b_0 to 1 and flip b_1 if the action of v is a StepUp. When node y receives $a = b_0 b_1$ 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. It is important to note that the key k does not need to be appended to the token itself as each node can keep track of its index in the list of appended tokens. Then when a cycle is detected, the node can index its token appropriately using the key k that is stored locally. Thus the overhead added by each node can be exactly two bits.

5.3.7 Characteristics

SAFETYPULSE has the following characteristics:

- **Provably Correct**. SAFETYPULSE is based on a theoretical framework of policy routing dynamics and changes in network topology do not affect the correctness of detecting policy conflicts.
- **Efficient Space**. A small token of space complexity $O(1)$ (two bits) is appended to each routing update message irrespective of how the routing dynamics manifest in the network. Thus a causation chain consisting of n nodes incurs a message overhead of $2n$ bits.

- **Policy Freedom**. Since SAFETYPULSE is a dynamic detection algorithm, it does not require any restrictions on routing policies to be imposed.
- **Backwards Compatible**. SAFETYPULSE requires only a minor extension to BGP and is therefore backwards compatible. To detect policy conflicts, only the ASes along the causation chains / cycles to be diagnosed need to adopt the protocol.

Chapter 6

Properties of Safe Routing Dynamics

6.1 Overview

In this chapter we distill three properties of safe routing dynamics (*i.e.*, dynamics when the policies of all nodes adhere to the Gao-Rexford guidelines). These properties hold irrespective of changes to the underlying topology or path preferences and can be used to diagnose the health of the network, in particular its routing dynamics. To this end, we also develop a token-based distributed algorithm, INTERFERENCEBEAT, to check adherence to these properties. We discuss and model reasons why ASes violate the Gao-Rexford guidelines which lead to potentially unsafe dynamics where the properties no longer hold. We show that these dynamics can still be precisely characterized and can be used to enhance the diagnostic power of INTERFERENCEBEAT. Our distributed algorithm, INTERFERENCEBEAT, essentially diagnoses and monitors the health of the network by detecting invalid routing dynamics (*i.e.,* causation chains that do not adhere to the derived properties) in realtime. The formal proofs and protocol specification can be found in [Mattar et al., c, Epstein et al., 2009]. We focus here on motivating the problem and on presenting the main intuition behind our results.

6.1.1 What are the properties?

Non-Interference Property: *If an AS y is not at a higher tier-level than (provider to) any two of its neighbors x and z, then x and z cannot directly induce path changes in each other through y. This property holds regardless of changes in the underlying topology or path preferences.*

The notion of "inducing path changes" is synonymous with a continuous propagation of path changes across nodes, which we model in DPR as a causation chain. The basic premise of the non-

interference property comes from a theoretical result where we proved that any causation chain must not contain sequences such as a provider-to-customer-to-provider. The relative tier-level of ASes can be easily derived from the commercial relationships between ASes. For example, an AS x that is a provider to AS y is at a higher tier-level.

It is important to note that all the properties we derive, including the non-interference property, are with respect to a single causation chain propagating in the network. This causation chain is initiated by one and only one root cause such as a change in a link's availability or a change in a node's path preferences.

Figure 6·1 outlines all the Internet configurations where AS x cannot directly affect AS z through AS y. More specifically, non-interference holds if:

1. AS y is multi-homed with providers AS x and AS z.
2. AS y is a customer of AS x and a peer of AS z.
3. AS y is a peer of AS x and a customer of AS z.
4. AS y is a peer of both AS x and AS z.

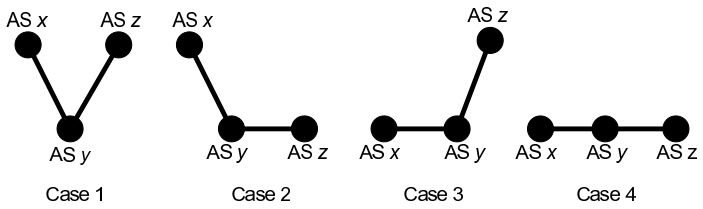


Figure 6·1: All Internet configurations where AS x cannot directly affect AS z. Horizontal edges represent peering links and diagonal edges represent customer-to-provider links.

Single Cycle Property: *In any cycle of routing update messages between ASes, every AS x affects its neighbor y at most once. This property holds regardless of changes in the underlying topology or path preferences.*

The notion of "cycle" is synonymous with a continuous propagation of path changes across nodes where at least one node is affected twice. We model such a cycle of path changes in DPR as a *causation cycle*. The single cycle property comes from a theoretical result where we proved that any causation cycle in safe policy routing occurs only once.

Multi-Tiered Cycle Property: *Every cycle of routing update messages between ASes must have at least two ASes in different tier-levels. This property holds regardless of changes in the underlying topology or path preferences.*

The multi-tiered cycle property comes from a theoretical result where we proved that no causation cycle in safe policy routing can occur exclusively between peering ASes.

6.1.2 Why do the properties not always hold?

Violations of safe policy routing (*i.e.,* the Gao-Rexford guidelines) result in unpredictable, black-box dynamics that are potentially unsafe. When policy violations occur, the properties no longer hold. The reasons for such violations are:

1. **Intentional:** representing legitimate policy configurations for backup links or complex agreements [Feamster et al., 2004].

2. **Unintentional:** representing misconfigurations or complex realtime interactions between routers that do not reflect the intentions of the administrators.

6.1.3 How do we check the properties?

Network administrators can locally check whether they are abiding by the Gao-Rexford guidelines where the dynamics are guaranteed to conform to the properties derived. This can be done by inspecting their local preferences and ensuring that all their import / export policies are set correctly. Local checks are inadequate, however, since not all nodes are necessarily compliant with the guidelines. Non-compliance by some nodes has global implications on the routing dynamics that cannot be easily checked locally. This creates the need to check adherence to the properties in realtime using a distributed algorithm such as INTERFERENCEBEAT.

Figure 6·2 illustrates "interference" between nodes 1 and 3. The interference is due to policy violations by node 2 which cannot be locally checked by node 3. Instead, node 3 will need to discover the interference by somehow detecting the causation chain propagating through nodes 1, 2 and 3.

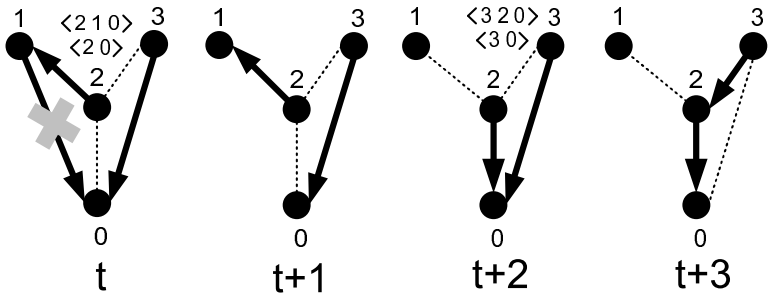


Figure 6·2: Sample dynamics where interference occurs. The list of path preferences for nodes 2 and 3 are organized such that the most preferred path is at the top. Paths not explicitly listed are forbidden. All nodes are trying to reach destination node 0.

Node 3 is abiding by the Gao-Rexford guidelines and initially uses the customer path $\langle 30 \rangle$ which is valley-free. Node 2, however, violates the guidelines by preferring a path through its provider $\langle 210 \rangle$ over a path through its customer $\langle 20 \rangle$. At time t, the link connecting node 1 to node 0 is lost, causing node 1 to have an empty path to node 0 at time $t + 1$. At time $t + 2$, node 2 switches from path $\langle 210 \rangle$ to $\langle 20 \rangle$. This action in turn causes node 3 to switch from path $\langle 30 \rangle$ to $\langle 320 \rangle$ at time $t + 3$. Even though node 3 abides by the Gao-Rexford guidelines, the forbidden interference occurs. The causation chain consists of a provider (node 1), followed by its customer (node 2), followed by another provider (node 3).

If node 2 does not violate the guidelines, the dynamics would manifest differently. For example, suppose that path $\langle 20 \rangle$ is forbidden, *forcing* node 2 to use its provider path $\langle 210 \rangle$. The loss of link connectivity between nodes 1 and 0 at time t causes node 2 to lose connectivity at time $t + 2$. Node 3 is unaffected. The causation chain solely consists of a provider (node 1) followed by its customer (node 2). Since this chain is valley-free, the dynamics conform to the properties we derived.

INTERFERENCEBEAT, our token-based distributed algorithm, checks if the properties hold or whether policy violations exist. This is accomplished by passing a small token alongside route updates to detect forbidden causation chains (including cycles) induced by policy violations. Once a forbidden causation chain is detected, the ASes involved need to collaborate to resolve the potential problem.

6.2 Causation in Economic DPR

This section characterizes causation chains and cycles for economic DPR instances. For convenience of notation, we drop the time index of certain terms with respect to a given chain $Y = \langle y_0\ y_1\ \ldots\ y_k \rangle^t$ as outline in Table 6.1.

Table 6.1: Notation

$\pi(y_i)$	=	$\pi(y_i, t+i)$
$\pi_{\text{next}}(y_i)$	=	$\pi(y_i, t+i+1)$
$\rho(y_i)$	=	$\rho(y_i, t+i)$
$\rho_{\text{next}}(y_i)$	=	$\rho(y_i, t+i+1)$
RankDec(y_i)	iff	RankDec$(y_i, t+i)$
RankSame(y_i)	iff	RankSame$(y_i, t+i)$
RankInc(y_i)	iff	RankInc$(y_i, t+i)$

Theorem 11. *Every causation chain of an economic DPR instance $(\succeq_\$, \succeq^t, G)$ is valley-free.*

Proof. Assume not. Then there exists a causation chain $Y = \langle y_0\ y_1\ \ldots\ y_k \rangle^t$ and an index i such that $0 < i < k$ and $y_{i-1} \succeq_\$ y_i \preceq_\$ y_{i+1}$. Thus y_{i-1} and y_{i+1} are peers or providers to y_i.

The first part of this proof shows that if this is the case, then at no time during the causation chain did y_i have a customer path. The second part of this proof shows that sometime during the causation chain y_{i+1} had a path through y_i. Therefore y_{i+1} had a realized valley path since y_i did not have a customer path and y_i is a customer of or peer to y_{i+1}. Since valley-paths are forbidden (not realizable) in economic DPR instances, this results in a contradiction. Since Cause$(y_i) = y_{i-1}$, either the first or second condition of causation from Table 3.2 holds for y_i at time $t+i$.

Case: y_i Causation Condition 1

If the first condition of Table 3.2 holds for y_i then: $\rho(y_i) = y_{i-1}$ and RankDec(y_i), as shown in Figure 6·3. Therefore $\pi(y_i) \succ^{t+i} \pi_{\text{next}}(y_i)$. Let $v = \rho_{\text{next}}(y_i)$. It cannot be that $v \prec_\$ y_i$. Otherwise, since $\pi_{\text{next}}(y_i)$ is a customer path and $\pi(y_i)$ is not a customer path (since $\rho(y_i) =$

$y_{i-1} \succeq_\$ y_i$), by the conditions of economic DPR instances: $\pi(y_i) \prec^{t+i} \pi_{\text{next}}(y_i)$, causing a contradiction as shown in Figure 6·4. Thus $v \succeq_\$ y_i$ and $\rho_{\text{next}}(y_i) \succeq_\$ y_i$.

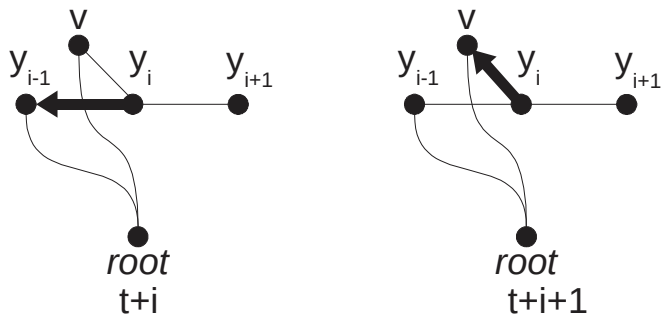


Figure 6·3: Causation condition 1: RankDec(y_i)

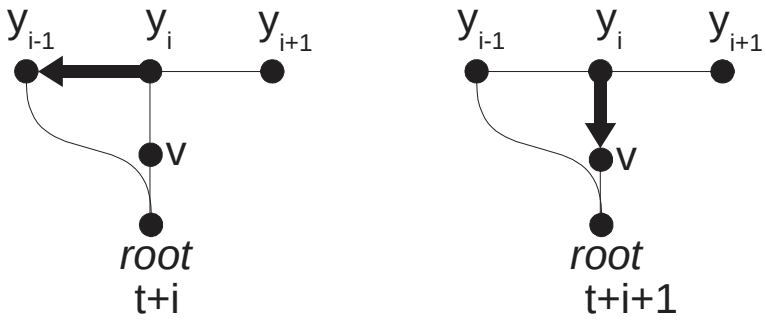


Figure 6·4: Contradiction: RankInc(y_i)

Case: $\mathbf{y_i}$ Causation Condition 2

If the second condition of Table 3.2 holds for y_i then: $\rho_{\text{next}}(y_i) = y_{i-1}$ and RankInc(y_i), as shown in Figure 6·5. Therefore $\pi(y_i) \prec^{t+i} \pi_{\text{next}}(y_i)$. Let $v = \rho(y_i)$. It cannot be that $v \prec_\$ y_i$. Otherwise, since $\pi(y_i)$ is a customer path and $\pi_{\text{next}}(y_i)$ is not (since $\rho_{\text{next}}(y_i) = y_{i-1} \succeq_\$ y_i$), by the conditions of economic DPR instances: $\pi(y_i) \succ^{t+i} \pi_{\text{next}}(y_i)$, causing a contradiction, as shown in Figure 6·6. Thus $\rho_{\text{next}}(y_i) \succeq_\$ y_i$ and $v \succeq_\$ y_i$. So for both cases, at no time in the causation chain did y_i have a customer path:

$$\rho(y_i) \succeq_\$ y_i \text{ and } \rho_{\text{next}}(y_i) \succeq_\$ y_i$$

Case: $\mathbf{y_{i+1}}$ Causation Condition 1

If the first causation condition of Table 3.2 holds for y_{i+1}, then $\rho(y_{i+1}) = y_i$. By Proposition

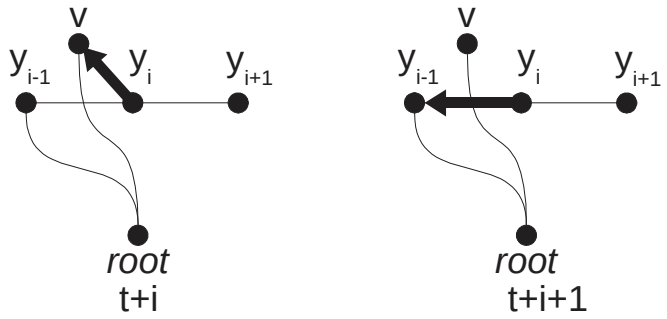


Figure 6·5: Causation condition 2: RankInc(y_i)

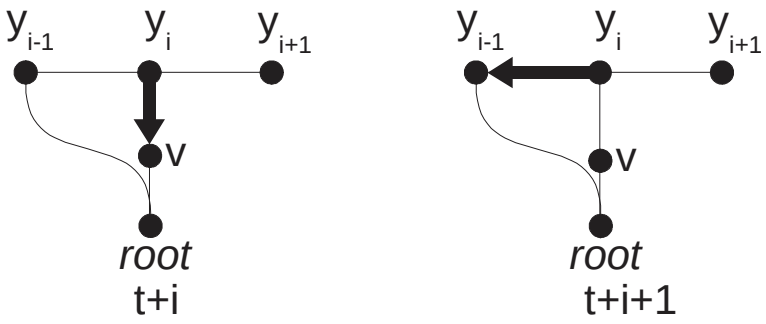


Figure 6·6: Contradiction: RankDec(y_i)

2: $\pi(y_{i+1}) = \langle y_{i+1}\ \pi(y_i)\rangle$. $\pi(y_{i+1})$ is a valley path since $y_{i+1} \succeq_\$ y_i \preceq_\$ \rho(y_i)$. Since all valley paths are forbidden (not realizable), $\pi(y_{i+1})$ can never be realized, causing a contradiction.

Case: $\mathbf{y_{i+1}}$ Causation Condition 2

Similar arguments can be used if the second causation condition of Table 3.2 holds for y_{i+1}: $\rho_{\text{next}}(y_{i+1}) = y_i$. Thus by Proposition 2: $\pi_{\text{next}}(y_{i+1}) = \langle y_{i+1}\ \pi_{\text{next}}(y_i)\rangle$. $\pi_{\text{next}}(y_{i+1})$ is a valley path since $y_{i+1} \succeq_\$ y_i \preceq_\$ \rho_{\text{next}}(y_i)$, and can never be realized. Thus in all cases a contradiction occurs, proving the theorem. □

Definition 34 (Horizontal Cycle)**.** A causation cycle is horizontal if all adjacent nodes in the cycle are peers.

Definition 35 (Vertical Cycle)**.** A causation cycle is vertical if there is at least one customer / provider relationship between adjacent nodes in the cycle.

Figure 3·19 represents a simple vertical causation cycle, where node y_0 loses a path to *root* and reroutes through y_2.

Lemma 6. *Given a causation cycle $Y = \langle y_0 \ \ldots \ y_k \rangle^t$ of an economic DPR instance $(\succeq_\$, \succeq^t, G)$, every node in Y is a provider to the first node y_0.*

Proof. Let $y_i \in Y$, where $0 < i < k$. By Theorem 11, Y is valley-free and either $y_{i-1} \preceq_\$ y_i$ or $y_i \succeq_\$ y_{i+1}$. If the first case is true, then by the definition of valley-free paths $y_{j-1} \prec_\$ y_j$ for all $0 < j < i$, and by the transitive nature of economic relationships, $y_0 \prec_\$ y_i$. If the second case is true, then by the definition of valley-free paths $y_j \succ_\$ y_{j+1}$ for all $i < j < k$, and by the transitive nature of economic relationships, $y_i \succ_\$ y_k$. Thus every node y_i is a provider to $y_0 = y_k$. □

Theorem 12. *Every causation cycle $Y = \langle y_0 \ \ldots \ y_k \rangle^t$ of an economic DPR instance is vertical and simple.*

Proof. Lemma 6 directly implies that every causation cycle in economic DPR instances is vertical. The second part regarding simple causation cycles is proved by contradiction. Assume there exists a non-simple causation cycle $Y_1 = \langle y_0 \ y_1 \ \ldots \ y_k \ y_1 \rangle^t$ where $y_0 = y_k$. From Lemma 6, $y_0 \prec_\$ y_1$. However a new causation cycle Y_2 exists where: $Y_2 = \langle y_1 \ y_2 \ \ldots \ y_{k-1} \ y_k \ y_1 \rangle^{t+1}$. Thus by Lemma 6, $y_1 \prec_\$ y_k = y_0$ which is a contradiction. □

The theoretical results in this section are the proofs for the three properties of safe policy routing dynamics introduced in Section 6.1. The non-interference principle comes from Theorem 11, which states that every causation chain in an economic DPR instance must be valley-free. The single and multi-tiered cycle properties come from Theorem 12, which states that every causation cycle in an economic DPR instance is vertical and simple.

6.3 InterferenceBeat

In this section, we outline a distributed algorithm, INTERFERENCEBEAT, that checks if the properties of safe policy routing dynamics are maintained or whether policy violations exist. This is accomplished by detecting forbidden causation chains (including cycles) induced by policy violations. Once a forbidden causation chain is detected, the ASes involved need to collaborate to resolve the potential problem.

6.3.1 Overview

In INTERFERENCEBEAT, each node appends a small token alongside the route update message. When a node y receives a route update from its neighbor v at time t, it also receives a token θ_{in}. If node y selects a new path then it broadcasts a new token θ_{out} alongside its own route update at time $t + 1$. Tokens are passed along causation chains. In general, a causation chain is started when a link flaps (*i.e.*, is lost or becomes available) or when a node changes its path preferences. A token consists of three parts, (i, r, n). The identifier of the causation chain is i. The economic relationship between y and its predecessor v on the causation chain is:

$$r \in \{\succ_{\$},\ \prec_{\$},\ =_{\$},\ \emptyset\}$$

For example, if v is a provider to y, then r is $\succ_{\$}$. The counter n keeps track of the number of times the token was passed along a customer-to-provider or a provider-to-customer link.

The PROCESS function outlined in Figure 6·7 performs basic routing tasks and handles the incoming and outgoing tokens. It is invoked in every node y at time t after receiving all routing update messages. In steps 2 and 3, node y chooses and adopts its best available path. If y's assigned path has changed in step 4 (*i.e.*, an action occurred), then node y's causing neighbor v is identified in step 5. The token received from neighbor v is recovered in step 6. In step 7, the CREATETOKEN function is called which returns the contents of the new token to be sent out by y at time $t + 1$. The CHECKPROPERTIES function is called in step 8. Node y stores information about the outgoing token in step 9, which is later used to detect cycles in the CHECKPROPERTIES function. The outgoing token is then disseminated to all y's neighbors in step 10.

The CREATETOKEN function is outlined in Figure 6·8. Step 2 retrieves the needed parts from the incoming token. If the identifier i_{in} is empty in step 3 then a new one is generated in step 4. Otherwise, in step 6, the outgoing identifier i_{out} is set to the incoming identifier i_{in}. In step 7, r_{out} is set to the economic relationship between v and y. In steps 8 through 11, the outgoing counter n_{out} is only incremented if nodes y and v are *not* peers. The outgoing token is returned in step 12.

The CHECKPROPERTIES function is outlined in Figure 6·9. Steps 2 and 3 retrieve the needed

1: **function** PROCESS(y, t)
2: Best$(y, t) \leftarrow \max_{\succeq t}$ Choices(y, t)
3: $\pi(y, t+1) \leftarrow$ Best(y, t)
4: **if** $\pi(y, t+1) \neq \pi(y, t)$ **then**
5: $v =$ Cause(y, t)
6: $\theta_{\text{in}} =$GETTOKENFROMNEIGHBOR(y, v, t)
7: θ_{out} = CREATETOKEN(y, v, θ_{in})
8: CHECKPROPERTIES($y, v, \theta_{\text{in}}, \theta_{\text{out}}$)
9: STORETOKEN($y, v, \theta_{\text{out}}$)
10: SENDTOKEN($y, t, \theta_{\text{out}}$)

Figure 6·7: PROCESS function.

1: **function** CREATETOKEN(y, v, θ_{in})
2: $(i_{\text{in}}, \_, n_{\text{in}}) = \theta_{\text{in}}$
3: **if** i_{in} is $\emptyset$ **then**
4: $(i_{\text{out}}, r_{\text{out}}, n_{\text{out}}) =$(NEWID(), $\emptyset$, 0)
5: **else**
6: $i_{\text{out}} = i_{\text{in}}$
7: r_{out} = ECONOMICRELATION(v, y)
8: **if** r_{out} is equal to $=_{\$}$ **then**
9: $n_{\text{out}} = n_{\text{in}}$
10: **else**
11: $n_{\text{out}} = n_{\text{in}} + 1$
12: **return** $(i_{\text{out}}, r_{\text{out}}, n_{\text{out}})$

Figure 6·8: CREATETOKEN function.

parts from the tokens. Step 4 checks for the existence of a valley causation chain. If one is found, then interference is reported, where the causing node v, the chain identifier i_{in} and the relationship r_{in} are identified. In step 6, node y determines if it has previously received a token with identifier i_{in}. If so, then a cycle is detected. Node y recovers the old information in step 7. If the token was previously received from the same neighbor v then a non-simple cycle is reported in step 9. Step 10 checks if the token previously received contained the same counter value. If so, then the token was only passed between peers since leaving node y and a horizontal cycle is reported in step 11.

```
1: function CHECKPROPERTIES(y, v, θ_in, θ_out)
2:     (i_in, r_in, _) = θ_in
3:     (_, _, n_out) = θ_out
4:     if (r_in is equal to ≻_$ or =_$) and (v ⪯_$ y) then
5:         REPORTINTERFERENCE(y, v, θ_in)
6:     if HASRECEIVEDTOKEN(y, i_in) then
7:         (v_old, n_old) = GETSTOREDTOKEN(y, i_in)
8:         if v_old is equal to v then
9:             REPORTNONSIMPLECYCLE(y, v, θ_in)
10:        if n_old is equal to n_out then
11:            REPORTHORIZONTALCYCLE(y, v, θ_in)
```

Figure 6·9: CHECKPROPERTIES function.

6.3.2 Sample Operation

Figure 6·10 shows the operation of INTERFERENCEBEAT on the DPR instance described in Figure 3·18, assuming y_0, y_1 and y_2 are all peers. At time $t + 1$, node y_0 initiates a new causation chain with identifier ID1 and sends a token to y_1. Since y_0 initiated the chain, the count is 0 and the relationship is $\emptyset$. Node y_1 takes an action and sends a new token to y_2. Since y_1 and y_0 are peers, the relationship is set to $=_\$$ and the count is still 0 as the token only traversed a peering link. Finally, since y_2 is a peer to its causing node y_1, interference is detected by y_2 upon receiving the token.

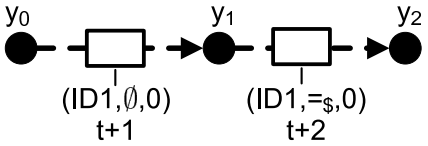


Figure 6·10: Sample operation of INTERFERENCEBEAT.

6.3.3 Characteristics

INTERFERENCEBEAT has the following characteristics:

- **Provably Correct**. INTERFERENCEBEAT is based on a theoretical framework of policy routing dynamics and changes in network topology or path preferences do not affect the

correctness of detecting policy violations.

- **Efficient Space**. A small token of space complexity $O(1)$ (a few bytes) is appended to each routing update message irrespective of how the routing dynamics manifest in the network.

- **Policy Freedom**. Since INTERFERENCEBEAT is a dynamic detection algorithm, it does not require any restrictions on routing policies to be imposed.

- **Backwards Compatible**. INTERFERENCEBEAT requires only a minor extension to BGP and is therefore backwards compatible. To detect policy violations, only the ASes along the causation chains to be diagnosed need to adopt the protocol. Thus neighboring ASes can use INTERFERENCEBEAT to detect misconfigurations.

6.3.4 Practical Considerations

INTERFERENCEBEAT could be implemented over BGP where the token is passed in the message options. When an AS initiates a new causation chain it must create a new identifier using the NEWID() function. This can be accomplished by hashing the AS number, router identifier, time and destination prefix. A fixed number of bits can be allocated to the identifier, with more bits reducing the probability of a hash collision.

In INTERFERENCEBEAT, if a cycle or valley is detected by a node y, only its causing neighbor node v can be immediately identified. In order to identify/notify other nodes along the chain, a back-propagating alert protocol may be used. Each node can leverage its stored tokens to find its previous causing neighbor. Note that a token only needs to be stored for the duration of the causation chain, thus the local storage requirements at a node are expected to be minimal.

In Appendix A we show that the synchronicity of DPR is not a hindrance and that it has sufficient expressive power to model asynchronicity. Hence, INTERFERENCEBEAT can be extended to a realtime setting.

6.4 Violations of the Economic DPR Model

We formally define four common policy violations, which represent different relaxations to the strict economic DPR model[1]. In the next sections we prove the invariant properties of the resultant causation chains and cycles for each violation. By invariant properties we mean the properties that hold irrespective of changes in path preferences or changes the underlying topology. The modelled dynamics induced by each violation can be compared against the dynamics observed by INTERFERENCEBEAT. If a violation cannot cause the observed behavior, then it can be ruled out.

6.4.1 Overview of Violations and their Induced Dynamics

To describe paths and causation chains in better detail we categorize valleys into four subtypes.

Definition 36 (Valley Types)**.** We extend definition 29 of valleys to four subtypes as shown in Table 6.2.

Table 6.2: Valley types given sequence $\langle a\ b\ c \rangle$.

Valley Type	Condition	Illustration
$\mathcal{A}$	$a \succ_\$ b \prec_\$ c$	
$\mathcal{B}$	$a \succ_\$ b =_\$ c$	
$\mathcal{C}$	$a =_\$ b \prec_\$ c$	
$\mathcal{D}$	$a =_\$ b =_\$ c$	

Table 6.3: Violations of the Economic DPR Model

Violation	Valley Types in Causation Chains: $\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$	Vertical Cycles	Horizontal Cycles	Potentially Unsafe?
0: None					simple	none	no
1: Non-Strict Economics					simple	none	no
2: Transiting					simple	non-simple, simple	yes
3: Peers Preferred					simple	non-simple, simple	yes
4: Providers Preferred					non-simple, simple	none	yes

[1]There are other relaxations that can be considered such as sibling relationships (*i.e.,* backup links) between ASes [Gao, 2001].

Violation 1: Non-Strict Economic Relationships

With non-strict economic relationships, a node can be both a (direct or indirect) provider and a (direct or indirect) peer to another node. Figure 6·11 shows a comparison between strict and non-strict economic relationships.

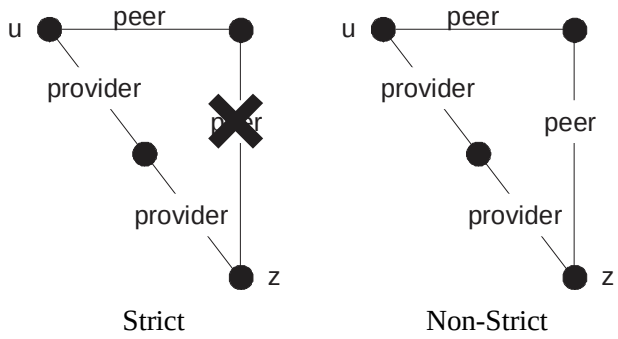


Figure 6·11: Strict and non-strict economic relationships. In the strict variant, node u cannot be an indirect provider and peer to node z. The crossed edge represents an edge that cannot exist in this variant.

Violation 2: Transiting Between Peers

Generally, an AS only carries traffic that is destined to (or originating from) one of its customers. However, due to misconfigurations or complex agreements between peers, an AS may transit traffic between its peers. Economic DPR instances with this violation have an enlarged set of realizable paths. Paths containing valleys of type $\mathcal{D}$ can be adopted by nodes. However, paths are forbidden (not realizable) if they contain valley types $\mathcal{A}$, $\mathcal{B}$, or $\mathcal{C}$. Paths are forbidden by having nodes configure their import / export policies for paths accordingly (*i.e.*, which paths are advertised to which neighbors and which paths are accepted from which neighbors). Therefore, every realizable path consists of a series of zero or more ascending customer-to-provider edges, followed by zero or more peer edges, followed by zero or more descending provider-to-customer edges, as shown in Figure 6·12.

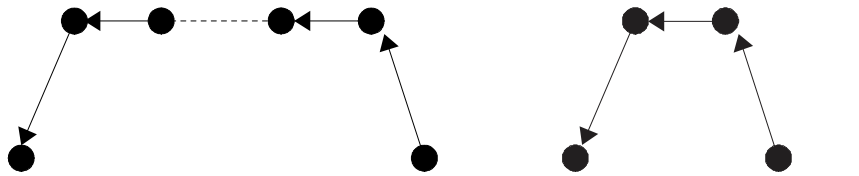


Figure 6·12: Allowable paths in economic DPR with and without violation 2. Nodes at the same level are peers. On the other hand, nodes at higher levels are providers for the nodes at lower levels that they are connected to.

Violation 3: Prefer Peer Paths Over Customer Paths

Whereas violation 2 is a relaxation on the set of realizable paths, violation 3 is a relaxation of the path preferences. Nodes in economic DPR instances with violation 3 can prefer peer paths over customer paths. Nodes, however, cannot prefer provider paths over peer/customer paths. Only valley-free paths are realizable.

Violation 4: Prefer Provider Paths Over Peer / Customer Paths

Nodes in economic DPR instances with violation 4 can prefer provider paths over peer/customer paths. Again, only valley-free paths are realizable. Again this is achieved by having each node configure its import / export policies for paths accordingly.

The four violations describe different variants of the economic DPR model. Each variant results in different types of causation chains and cycles. Table 6.3 summarizes the effects of each violation on the characteristics of causation chains and cycles. The first and second rows show the strict and non-strict economic DPR models. They are the only two variants guaranteed to be safe. The three other violations induce routing behavior which is potentially unsafe.

INTERFERENCEBEAT can be extended using the results of Table 6.3. Upon the detection of a valley in the causation chain, its type ($\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, or $\mathcal{D}$) can rule out possible causing violations. For example, if a valley of type $\mathcal{B}$ was detected using INTERFERENCEBEAT, then violations 1, 2, and 3 can be immediately ruled out as the possible causes for the observed behavior. Similar methods

can be used upon detection of non-simple or horizontal causation cycles.

The theoretical proofs for the dynamics induced by each violation are presented next.

6.4.2 Violation 1: Non-Strict Economic Relationships

First we consider the dynamics induced by violation 1, outlined in the second row of Table 6.3, where ASes can have non-strict economic relationships (*i.e.,* an AS can be both a provider and a peer to another AS). Similar to the case where there are no violations, we prove that vertical cycles must be simple, horizontal cycles are not possible, all causation chains are valley-free, and the resulting routing policy configuration is safe.

We start by formally defining non-strict economic relationships. If an economic DPR instance has non-strict economic relationships, then it contains the operator $\succeq_*$ where $D = (\succeq_*, \succeq, G)$. From $\succeq_*$, a tight economic relation is defined by:

$$u \succ_* v \text{ iff } u \succeq_* v \text{ and } u \not\preceq_* v$$

and no relation is defined by:

$$u \|_* v \text{ iff } u \not\succeq_* v \text{ and } u \not\preceq_* v$$

The customer, peer, and provider economic relationships can be derived from the operator $\succeq_*$:

- If u is a customer of v, then $u \prec_* v$.
- If u is a provider to v, then $u \succ_* v$.
- If u is a peer to v, then $u \|_* v$.

The properties of the economic operator $\succeq_*$ can be modeled using post-order conditions:

1. (reflexive) $x \succeq_* x$
2. (anti-symmetric) $x \succeq_* y$ and $y \succeq_* x$ implies $x = y$
3. (transitive) $x \succeq_* y$ and $y \succeq_* z$ implies $x \succeq_* z$

The key difference between a strict and non-strict economic operator is that peering relationships are not transitive in the non-strict variant. Whereas peering is represented by the equivalence relation $=_{\$}$ in the strict variant, peering is represented by no relation $\|_*$ in the non-strict variant. Strict economic relationships form equivalence classes with the peering relation $=_{\$}$ as shown in Figure 6·13. Such equivalence classes are not present in the non-strict variant. This enables a node to be both an indirect peer and provider to another node in the non-strict variant as shown in Figure 6·11. However it should be noted that provider-to-customer relationships are transitive in both variants.

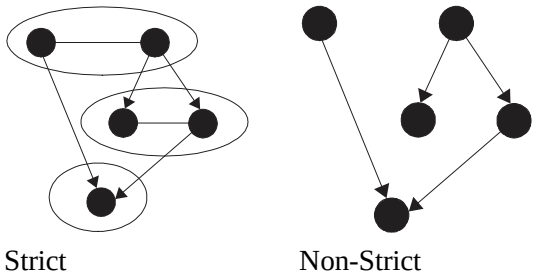


Figure 6·13: Strict and Non-Strict economic relationships. The circles over the nodes in the strict variant represent equivalent classes of peers.

For ease of exposition, the following notation is used to describe that node x is a peer or provider to node y:

$$x \succsim_* y \text{ iff } x \not\prec_* y$$

We define paths by the economic relationship between a path's starting node u and its next-hop. For all paths P^u:

$$\begin{aligned} \text{Customer}(P^u) &\Leftrightarrow u \succ_* \text{NextHop}(P^u) \\ \text{Peer}(P^u) &\Leftrightarrow u \|_* \text{NextHop}(P^u) \\ \text{Provider}(P^u) &\Leftrightarrow u \prec_* \text{NextHop}(P^u) \end{aligned}$$

Given a sequence of nodes $\langle a\ b\ c \rangle$, valley types are represented as follows:

Valley Type	Condition	Illustration
$\mathcal{A}$	$a \succ_* b \prec_* c$	
$\mathcal{B}$	$a \succ_* b \parallel_* c$	
$\mathcal{C}$	$a \parallel_* b \prec_* c$	
$\mathcal{D}$	$a \parallel_* b \parallel_* c$	

Theorem 13. *All causation chains of non-strict economic DPR instances are valley-free.*

Proof. This proof is identical to the one for Theorem 11, by replacing the $\succ_\$$ with $\succ_*$ and $\succeq_\$$ with $\succsim_*$. □

Theorem 14. *All causation cycles of non-strict economic DPR instances are vertical and simple.*

Proof. Let $Y = \langle y_0\ y_1\ \ldots\ y_k \rangle^t$ be a causation cycle, where $y_0 = y_k$. The cases for this proof can be partitioned by y_1's economic relationship with y_0:

Case (a): $\mathbf{y_0 \succ_* y_1}$
If $y_0 \succ_* y_1$, since Y is valley-free, $y_i \succ_* y_{i+1}$ for $0 \leq i < k$. However $y_0 \succ_* y_k = y_0$, causing a contradiction and eliminating this case.

Case (b): $\mathbf{y_0 \parallel_* y_1}$
If $y_0 \parallel_* y_1$, since Y is valley-free, $y_i \succ_* y_{i+1}$ for $1 \leq i < k$. Thus Y is vertical. Y has to be simple, otherwise $\langle y_{k-1}\ y_0\ y_1 \rangle$ would be a realized causation chain. Since $y_{k-1} \succ_* y_0$ and $y_0 \parallel_* y_1$, the causation chain is a valley, causing a contradiction. Therefore Y is simple and vertical.

Case (c): $\mathbf{y_0 \prec_* y_1}$
If $y_0 \prec_* y_1$ then Y is vertical. The cases can be further partitioned by y_{k-1}'s economic relationship with y_k. If $y_{k-1} \prec_* y_k$, then by the definition of valley-free sequences, $y_{i-1} \prec_* y_i$ for all $0 < i \leq k$. Thus $y_0 \prec_* y_k = y_0$, which is a contradiction. Therefore $y_{k-1} \succsim_* y_k$. If Y is non-simple, then $\langle y_{k-1}\ y_0\ y_1 \rangle$ would be a realized causation chain. Since $y_{k-1} \succsim_* y_0 = y_k$ and $y_0 \prec_* y_1$, the causation chain is a valley, causing a contradiction. Therefore Y is simple and vertical. □

Remark 1. DPR instances with non-strict economic relationships are safe. This follows directly from the results in [Gao and Rexford, 2001].

6.4.3 Violation 2: Transiting Between Peers

Next we consider the dynamics induced by violation 2, outlined in the third row of Table 6.3, where ASes can transit traffic between their peers. We prove that all vertical cycles must be simple, horizontal cycles can be simple or non-simple, only causation chains of valley type $\mathcal{D}$ are possible, and the resulting routing policy configuration is potentially unsafe.

Theorem 15. *Every causation chain in an economic DPR instance with violation 2 does not admit valley types $\mathcal{A}$, $\mathcal{B}$ or $\mathcal{C}$.*

Proof. Assume not. Then there exists a causation chain $Y = \langle y_0\ y_1\ \ldots\ y_k \rangle^t$ and an index i such that $0 < i < k$ and at least one of these two conditions hold:

(a) $y_{i-1} \succ_\$ y_i \preceq_\$ y_{i+1}$

(b) $y_{i-1} \succeq_\$ y_i \prec_\$ y_{i+1}$

Case (a): $y_{i-1} \succ_\$ y_i \preceq_\$ y_{i+1}$

If case (a) holds, then it can be shown that both $\rho(y_i) \succ_\$ y_i$ and $\rho_{\text{next}}(y_i) \succ_\$ y_i$. This can be seen by looking at the causation conditions of y_i. If causation condition 1 holds for y_i, then $y_{i-1} = \rho(y_i)$ and RankDec(y_i). It cannot be the case that $\rho_{\text{next}}(y_i) \preceq_\$ y_i$, since this would imply that y_i switched from a provider path through y_{i-1} to a non-provider path, since $y_i \prec_\$ \rho(y_i) = y_{i-1}$ and $y_i \succeq_\$ \rho_{\text{next}}(y_i)$. This would imply RankInc(y_i), causing a contradiction. Thus $\rho(y_i) \succ_\$ y_i$ and $\rho_{\text{next}}(y_i) \succ_\$ y_i$. If causation condition 2 holds for y_i, then $y_{i-1} = \rho_{\text{next}}(y_i)$ and RankInc(y_i). It cannot be the case that $\rho(y_i) \preceq_\$ y_i$, since this would imply that y_i switched from a non-provider path to a provider path through y_{i-1}, since $y_i \succeq_\$ \rho(y_i)$ and $y_i \prec_\$ \rho_{\text{next}}(y_i) = y_{i-1}$. This would imply RankDec(y_i), causing a contradiction. Thus for both cases, $\rho(y_i) \succ_\$ y_i$ and $\rho_{\text{next}}(y_i) \succ_\$ y_i$.

Thus given the results above, we can prove that y_{i+1} had a realized path with valley type $\mathcal{A}$ or $\mathcal{B}$. If causation condition 1 holds for y_{i+1}, then $\pi(y_{i+1}) = \langle y_{i+1}\ \pi(y_i) \rangle$. Since $y_{i+1} \succeq_\$ y_i$ and $y_i \prec_\$ \rho(y_i)$, then $\pi(y_{i+1})$ is a realized path with valley type $\mathcal{A}$ or $\mathcal{B}$, causing a contradiction. If causation condition 2 holds for y_{i+1}, then $\pi_{\text{next}}(y_{i+1}) = \langle y_{i+1}\ \pi_{\text{next}}(y_i) \rangle$. Since $y_{i+1} \succeq_\$ y_i$ and $y_i \prec_\$ \rho_{\text{next}}(y_i)$, then $\pi_{\text{next}}(y_{i+1})$ is a realized path with valley type $\mathcal{A}$ or $\mathcal{B}$, causing a contradiction.

Case (b): $y_{i-1} \succeq_\$ y_i \prec_\$ y_{i+1}$

If case (b) holds, then using an argument similar to case (a) it can be shown that both $\rho(y_i) \succeq_\$ y_i$ and $\rho_{\text{next}}(y_i) \succeq_\$ y_i$. We can then prove that y_{i+1} had a realized path with valley type $\mathcal{A}$ or $\mathcal{C}$, causing a contradiction. □

Theorem 16. *Every vertical causation cycle $Y = \langle y_0 \ \ldots \ y_k \rangle^t$ in an economic DPR instance with violation 2 is simple.*

Proof. This proof proceeds by determining y_1's economic relationship with y_0 and y_{k-1}'s economic relationship with $y_k = y_0$. Since Y is a vertical causation cycle, there exists a minimal index i, $0 < i < k$ such that $y_i \neq_\$ y_{i-1}$. Note that $i \neq k$, otherwise $y_0 =_\$ y_1 =_\$ \ldots =_\$ y_{k-1} \neq_\$ y_k$, implying $y_0 \neq_\$ y_k$, which is a contradiction. Either $y_i \succ_\$ y_{i-1}$ or $y_i \prec_\$ y_{i-1}$. It cannot be that $y_{i-1} \succ_\$ y_i$ since by Theorem 15, $y_0 =_\$ y_{i-1} \succ_\$ y_i \succ_\$ y_{i+1} \ldots \succ_\$ y_k$, implying $y_0 \succ_\$ y_k$ which is a contradiction. Therefore $y_{i-1} \prec_\$ y_i$. If $i > 1$, then $y_{i-2} =_\$ y_{i-1} \prec_\$ y_i$, representing a valley of type $\mathcal{C}$, which is a contradiction. So $i = 1$ and $y_0 \prec_\$ y_1$.

Let j be the first index, $1 < j < k$, where $y_{j-1} \succ_\$ y_j$. Note that j has to exist otherwise $y_0 \prec_\$ y_1 \preceq_\$ \ldots \preceq_\$ y_k$, implying $y_0 \prec_\$ y_k$ which is a contradiction. From Theorem 15, $y_{h-1} \succ_\$ y_h$ for all $j < h \leq k$. So $y_{k-1} \succ_\$ y_k = y_0$. Therefore Y must be simple, otherwise $\langle y_{k-1} \ y_0 \ y_1 \rangle$ must be a causation chain. However since $y_{k-1} \succ_\$ y_0$ and $y_0 \prec_\$ y_1$, Y contains a valley of type $\mathcal{A}$, contradicting Theorem 15, and thus proving the theorem. □

Theorem 17. *An economic DPR instance with violation 2 admits simple and non-simple horizontal causation cycles.*

Proof. This follows directly from the example shown in Figure 6·14 which is identical to the "Bad Gadget" instance described in [Griffin et al., 2002]. □

Theorem 18. *An economic DPR instance with violation 2 is potentially unsafe.*

Proof. Again from the example shown in Figure 6·14, no stable assignment exists. □

Path preferences:

Node a: $\langle a \ b \ \textit{root} \rangle$
$\langle a \ \textit{root} \rangle$

Node b: $\langle b \ c \ \textit{root} \rangle$
$\langle b \ \textit{root} \rangle$

Node c: $\langle c \ a \ \textit{root} \rangle$
$\langle c \ \textit{root} \rangle$

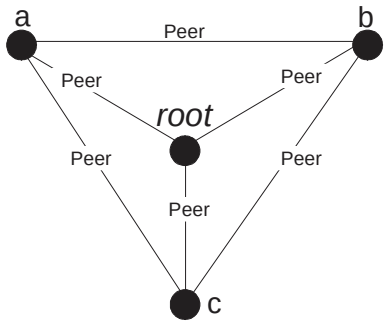


Figure 6·14: Non-simple horizontal cycle for an economic DPR instance with violation 2. Paths not listed in the path preferences are forbidden.

6.4.4 Violation 3: Prefer Peer Paths Over Customer Paths

Next we consider the dynamics induced by violation 3, outlined in the fourth row of Table 6.3, where ASes can prefer peer paths over customer paths. This violation induces dynamics that are similar in nature to those induced by violation 2. We prove that all vertical cycles must also be simple, horizontal cycles can be simple or non-simple, and the resulting routing policy configuration is potentially unsafe. The only difference is that, in addition to causation chains of valley type $\mathcal{D}$, we prove that causation chains of valley type $\mathcal{C}$ are now also possible.

Theorem 19. *Every causation chain in an economic DPR instance with violation 3 does not admit valley types $\mathcal{A}$ or $\mathcal{B}$.*

Proof. Assume not. Then there exists a causation chain $Y = \langle y_0\ y_1\ \ldots\ y_k \rangle^t$ and an index i such that $0 < i < k$ and $y_{i-1} \succ_\$ y_i \preceq_\$ y_{i+1}$. The same reasoning as case (a) from the proof of Theorem 15 can be used. By considering the causation conditions of y_i, it can be shown that both $\rho(y_i) \succ_\$ y_i$ and $\rho_{\text{next}}(y_i) \succ_\$ y_i$. We can then prove that y_{i+1} had a realized path with valley type $\mathcal{A}$ or $\mathcal{B}$, causing a contradiction. □

Path preferences:
Node a: $\langle a\ b\ \textit{root} \rangle$
$\langle a\ \textit{root} \rangle$
Node b: $\langle b\ c\ \textit{root} \rangle$
$\langle b\ \textit{root} \rangle$
Node c: $\langle c\ a\ \textit{root} \rangle$
$\langle c\ \textit{root} \rangle$

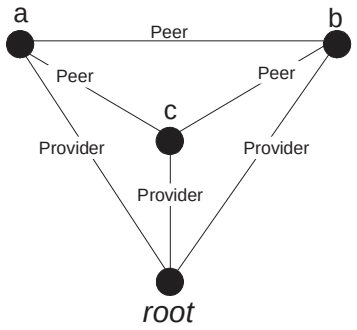


Figure 6·15: Non-simple horizontal cycle for an economic DPR instance with violation 3. Paths not listed in the path preferences are forbidden.

Theorem 20. *Every vertical causation cycle in an economic DPR instance with violation 3 is simple.*

Proof. Assume not. Let vertical causation cycle $Y = \langle y_0\ y_1\ \ldots\ y_k \rangle^t$ be non-simple. Since Y is a vertical causation cycle, there exists a minimal index i, $0 < i < k$ such that $y_i \neq_\$ y_{i-1}$. Following an argument similar to the one used to prove Theorem 16, we can prove that Y contains a valley of type $\mathcal{A}$ or $\mathcal{B}$, which is a contradiction. □

Theorem 21. *An economic DPR instance with violation 3 admits simple and non-simple horizontal causation cycles.*

Proof. This follows directly from the example shown in Figure 6·15 which is identical to the "Bad Gadget" instance described in [Griffin et al., 2002]. □

Theorem 22. *An economic DPR instance with violation 3 is potentially unsafe.*

Proof. From the example shown in Figure 6·15, no stable assignment exists. □

6.4.5 Violation 4: Prefer Provider Paths Over Peer / Customer Paths

Finally we consider the dynamics induced by violation 4, outlined in the last row of Table 6.3, where ASes can prefer provider paths over peer / customer paths. We prove that vertical cycles can be simple or non-simple, horizontal cycles do not occur, only causation chains of valley types $\mathcal{A}$ and $\mathcal{B}$ are possible, and the resulting routing policy configuration is potentially unsafe.

Theorem 23. *Every causation chain in an economic DPR instance with violation 4 does not admit valley types $\mathcal{C}$ or $\mathcal{D}$.*

Proof. Assume not. Then there exists a causation chain $Y = \langle y_0\ y_1\ \ldots\ y_k \rangle^t$ and an index i such that $0 < i < k$ and $y_{i-1} =_\$ y_i \preceq_\$ y_{i+1}$. The rest of the proof is similar to that of Theorem 15. First we show that both $\rho(y_i) \succeq_\$ y_i$ and $\rho_{\text{next}}(y_i) \succeq_\$ y_i$. Then we show that either $\pi(y_{i+1})$ or $\pi_{\text{next}}(y_{i+1})$ is a valley path of type $\mathcal{C}$ or $\mathcal{D}$, causing a contradiction. □

Theorem 24. *There are no horizontal cycles in economic DPR instances with violation 4.*

Proof. This follows directly from Theorem 23, which states that causation chains of type $\mathcal{D}$ do not exist. □

Theorem 25. *An economic DPR instance with violation 4 admits simple and non-simple vertical causation cycles.*

Proof. This follows directly from the example shown in Figure 6·16 which is identical to the "Bad Gadget" instance described in [Griffin et al., 2002]. □

Theorem 26. *An economic DPR instance with violation 4 is potentially unsafe.*

Proof. From the example shown in Figure 6·16, no stable assignment exists. □

Path preferences:

Node a: $\langle a\ b\ root \rangle$
$\langle a\ root \rangle$

Node b: $\langle b\ c\ root \rangle$
$\langle b\ root \rangle$

Node c: $\langle c\ a\ root \rangle$
$\langle c\ root \rangle$

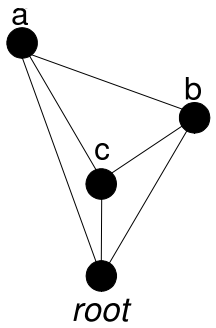


Figure 6·16: Non-simple vertical cycle for an economic DPR instance with violation 4. All edges are customer/provider links. Paths not listed in the path preferences are forbidden.

Chapter 7

Conclusion

We introduced the Dynamic Policy Routing (DPR) model which extends the Stable Paths Problem (SPP) [Griffin et al., 2002] with discrete synchronous time. DPR captures the propagation dynamics of path changes due to arbitrary changes in topology or path preferences. We introduced policy digraphs—a time-invariant structure which captures how routing update messages can propagate in the network.

Using our policy digraphs we formalized the Routing Dynamics Minimization Problem (RDMP) to solve a graph optimization problem. This optimization problem explicitly minimizes one possible metric, namely, the longest sequence of routing update messages in any dynamic network. This is done by changing the path preferences of nodes. We show that finding a policy configuration which minimizes the length of the policy digraph is NP-Hard. While RDMP is NP-Hard, we believe that it complements SPP and we envision that its formulation will allow us to explore problems where the dynamics of policy routing can be examined.

We characterized policy routing in the presence of policy conflicts to develop an efficient policy conflict detector. We introduced SAFETYPULSE—a distributed policy conflict detection algorithm.

We derived several invariant properties of routing dynamics in a safe (economic) policy configuration. We introduced INTERFERENCEBEAT—a distributed algorithm to detect and diagnose policy violations. INTERFERENCEBEAT was further enhanced by modeling common policy violations and characterizing the resulting dynamics.

Appendices

Appendix A

Asynchronicity with DPR

A.1 Overview

This section describes how the DPR model can simulate asynchronicity. We assume that we have a regular DPR instance $D = (\succeq, G)$ which we wish to augment with asynchronicity. There are several ways to represent asynchronicity. We will use link delays. This choice enables us to use the existing DPR model without adding new constructs. 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 the values $L(u, v, 4) = 100$ 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 DPR instance $D' = (\succeq', G')$ can be constructed to simulate D with the time delays.

For every pair of nodes in the original instance D, a set of $M - 1$ transit nodes will be added to D'. These transit nodes represent the "communication wire" between every two nodes. The dynamic nature of the links in DPR instances will be used to control the length of the "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.

A.2 Graph of Asynchronous DPR Instances

For every node u in the original DPR instance D, there is a corresponding node in the asynchronous DPR instance D':

$$u \in V \Rightarrow u \in V'$$

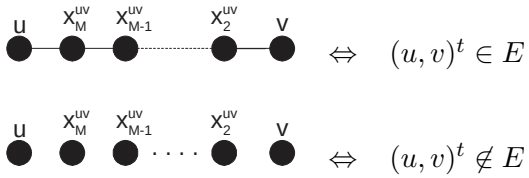


Figure A·1: Transit Nodes

For every two nodes u, v in D, there are $M-1$ transit nodes:

$$u, v \in V \Rightarrow x_i^{uv} \in V' \text{ for } 2 \leq i \leq M$$

Each transit node is connected to its neighbors. This connection forms the longest possible communication between nodes u and v. It toggles on/off with the connectivity of $(u,v)^t \in E$ for each time t, as shown in Figure A·1.

$$\left\{ \begin{array}{l} (u, x_M^{uv})^t \in E' \\ (x_{i+1}^{uv}, x_i^{uv}) \in E' \text{ for all } 1 < i < M \\ (x_2^{uv}, v)^t \in E' \end{array} \right\} \text{ iff } (u,v)^t \in E$$

The time delays $L(u,v,t)$ describe the "shortcut" available through the transit nodes at each time t:

$$(u,v)^t \in E' \quad \text{iff} \quad (u,v)^t \in E \text{ and } L(u,v,t) = 1$$
$$(u, x_i^{uv})^t \in E' \quad \text{iff} \quad (u,v)^t \in E \text{ and } L(u,v,t) = i$$

An example of a delay of one and three between nodes u and v can be seen in Figures A·2 and A·3.

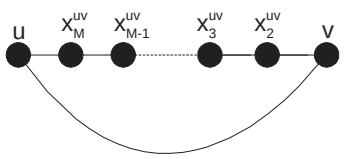


Figure A·2: Transit nodes simulating a delay of $L(u,v,t) = 1$.

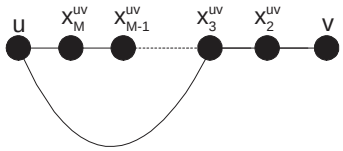


Figure A·3: Transit nodes simulating a delay of $L(u, v, t) = 3$.

A.3 Path Preferences of Asynchronous DPR Instances

The path preferences of the asynchronous DPR: $D' = (\succeq', G')$ discount the presence of transit nodes in paths. Let the operation RemoveTransit remove all transit nodes of a sequence. This operation allows us to derive the asynchronous path preferences from the original synchronous path preferences. Thus for all non-transit nodes $u \in V'$:

$$P_1^u \succeq'^t P_2^u \text{ iff RemoveTransit}(P_1^u) \succeq^t \text{RemoveTransit}(P_2^u)$$

Each transit node x_i^{uv} prefers a path through its source node u than through its transit neighbor toward the source: x_{i+1}^{uv}. Paths containing sequences in the opposite direction of the "communication link" (from x_i^{uv} to x_{i-1}^{uv}) are forbidden.

A.4 Redundant Connections

The transformation from synchronous to asynchronous DPR instances described above needs to be enhanced to avoid transient routing losses. This can occur during abrupt changes in connection delays as shown in Figure A·4.

In order to remedy this situation, redundant links between the source node u and the transit nodes are established, as shown in Figure A·5. This enables path consistency during changes of communication delays. Thus the proper transformation of links from synchronous D to asyn-

chronous D' can be represented as:

$$(u,v)^t \in E' \quad \text{iff} \quad (u,v)^t \in E \text{ and } L(u,v,t) = 1$$
$$(u,x_i^{uv})^t \in E' \quad \text{iff} \quad (u,v)^t \in E \text{ and } L(u,v,t) \leq i$$

A.5 Causation Chains in Asynchronous DPR Instances

The definition of causation chains is not changed for asynchronous DPR instances. Given delay $L(u,v,t) = 3$, a causation chain of $\langle u\ v\rangle^t$ in the original DPR instance D would correspond to a causation chain of $\langle u\ x_3^{uv}\ x_2^{uv}\ v\rangle^t$ in the asynchronous DPR instance D'.

A.6 Asynchronous Economic DPR Instances

Asynchronous economic DPR instances can follow the Gao-Rexford guidelines. Transit nodes have no economic relationships with the other nodes. The domain of the economic operator $\succeq_\$$ is only over non-transit nodes. Characterization of sequences (causation chains or paths) is accomplished by using the RemoveTransit operator. A path P in D' is valley-free if its corresponding transit-free path RemoveTransit(P) is valley-free. Similarly, a causation chain Y in D' is valley-free if its corresponding transit-free chain RemoveTransit(Y) is valley-free. Similar use of RemoveTransit can be employed to characterize customer, peer, and provider paths. From this construction, the proofs in this thesis are unchanged except for the application of the RemoveTransit operator.

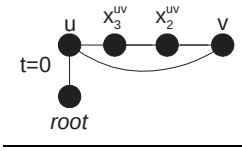
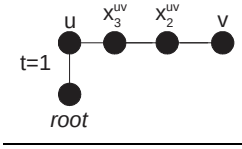
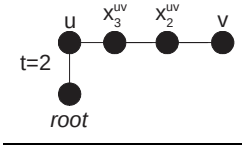
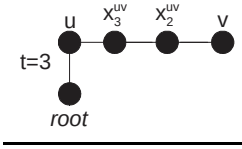
Graph	Node	Path
	u	$\langle u\ \textit{root}\rangle$
	x_3^{uv}	$\langle\rangle$
	x_2^{uv}	$\langle\rangle$
	v	$\langle\rangle$
	u	$\langle u\ \textit{root}\rangle$
	x_3^{uv}	$\langle x_3^{uv}\ u\ \textit{root}\rangle$
	x_2^{uv}	$\langle\rangle$
	v	$\langle v\ u\ \textit{root}\rangle$
	u	$\langle u\ \textit{root}\rangle$
	x_3^{uv}	$\langle x_3^{uv}\ u\ \textit{root}\rangle$
	x_2^{uv}	$\langle x_2^{uv}\ x_3^{uv}\ u\ \textit{root}\rangle$
	v	$\langle\rangle$
	u	$\langle u\ \textit{root}\rangle$
	x_3^{uv}	$\langle x_3^{uv}\ u\ \textit{root}\rangle$
	x_2^{uv}	$\langle x_2^{uv}\ x_3^{uv}\ u\ \textit{root}\rangle$
	v	$\langle v\ x_2^{uv}\ x_3^{uv}\ u\ \textit{root}\rangle$

Figure A·4: Node v has a transient path loss from node u. This is due to an increase in delay from $L(u, v, 0) = 1$ to $L(u, v, 1) = 3$. At time $t = 0$ only node u has a path to *root* and link (u, v) becomes unavailable. At time $t = 1$, node x_3^{uv} receives a route update from node u while node v has the best path $\langle x_3^{uv}\ u\ \textit{root}\rangle$ from the previous round. At time $t = 2$, node x_2^{uv} receives a route update from node x_3^{uv} while node v realizes that link (u, v) is unavailable and loses its path. At time $t = 3$, node v receives a new route update from node x_2^{uv} and updates its path to $\langle v\ x_2^{uv}\ x_3^{uv}\ u\ \textit{root}\rangle$.

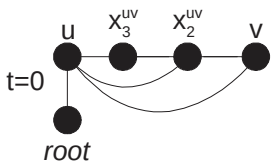


Figure A·5: The transient path loss at node v is prevented by having redundant connections. Node v will never have an empty path when link (u, v) becomes unavailable.

References

Ahronovitz, Knig, J.-C., and Saad, C. (2006). A Distributed Method for Dynamic Resolution of BGP Oscillations. In *IPDPS*.

Cittadini, L., Battista, G. D., Rimondini, M., and Vissicchio, S. (2009). Wheel + Ring = Reel: the Impact of Route Filtering on the Stability of Policy Routing. In *ICNP*.

Cobb, J. and Musunuri, R. (2004). Enforcing Convergence in Inter-Domain Routing. *GLOBECOM*, 3:1353–1358.

Davey, B. A. and Priestley, H. A. (2002). *Introduction to Lattices and Order*. Cambridge University Press.

Ee, C. T., Ramachandran, V., Chun, B., Lakshminarayanan, K., and Shenker, S. (2007). Resolving Inter-Domain Policy Disputes. In *SIGCOMM*, pages 157–168.

Epstein, S., Mattar, K., and Matta, I. (2009). Principles of Safe Policy Routing Dynamics. Technical Report BUCS-TR-2009-13, Computer Science Department, Boston University.

Fabrikant, A. and Papadimitriou, C. H. (2008). The Complexity of Game Dynamics: BGP Oscillations, Sink Equilibria, and Beyond. In *SODA*, pages 844–853.

Feamster, N. and Balakrishnan, H. (2004). Verifying the Correctness of Wide-Area Internet Routing. Technical report, MIT.

Feamster, N., Balakrishnan, H., and Rexford, J. (2004). Some Foundational Problems in Interdomain Routing. In *HotNets*.

Feamster, N., Johari, R., and Balakrishnan, H. (2005). Implications of Autonomy for the Expressiveness of Policy Routing. In *SIGCOMM*, pages 25–36.

Feamster, N. and Rexford, J. (2007). Network-wide prediction of BGP routes. *IEEE/ACM Transactions on Networking*, 15(2):253–266.

Feigenbaum, J., Ramachandran, V., and Schapira, M. (2006a). Incentive-compatible Interdomain Routing. In *EC*, pages 130–139.

Feigenbaum, J., Sami, R., and Shenker, S. (2006b). Mechanism Design for Policy Routing. *Distributed Computing*, 18:293–305.

Feigenbaum, J. and Shenker, S. (2002). Distributed Algorithmic Mechanism Design: Recent Results and Future Directions. In *DIALM 2002*, pages 1–13.

Feldmann, A., Maennel, O., Mao, Z., Berger, A., and Maggs, B. (2004). Locating Internet Routing Instabilities. In *SIGCOMM*.

Gao, L. (2001). On Inferring Autonomous System Relationships in the Internet. *IEEE/ACM Transactions on Networking*, 9(6):733–745.

Gao, L. and Rexford, J. (2001). Stable Internet Routing Without Global Coordination. *IEEE/ACM Transactions on Networking*, 9(6):681–692.

Govindan, R., Alaettinoglu, C., Eddy, G., Kessens, D., Kumar, S., and san Lee, W. (1999). An Architecture for Stable, Analyzable Internet Routing. *IEEE Network*, 13:29–35.

Griffin, T., Shepherd, F., and Wilfong, G. (2002). The Stable Paths Problem and Interdomain Routing. *IEEE/ACM Transactions on Networking*, 10(2):232–243.

Griffin, T. and Wilfong, G. T. (2000). A Safe Path Vector Protocol. In *INFOCOM*, pages 490–499.

Griffin, T. G. and Sobrinho, J. L. (2005). Metarouting. In *SIGCOMM*, pages 1–12.

Haxell, P. E. and Wilfong, G. T. (2008). A Fractional Model of the Border Gateway Protocol (BGP). In *SODA*, pages 193–199.

Labovitz, C., Ahuja, A., Bose, A., and Jahanian, F. (2000). An Experimental Study of Internet Routing Convergence. Technical Report MSR-TR-2000-08, Microsoft Research and EECS Department at University of Michigan.

Labovitz, C., Ahuja, A., Bose, A., and Jahanian, F. (2001). Delayed Internet Routing Convergence. *IEEE/ACM Transactions on Networking*, 9:293–306.

Levin, H., Schapira, M., and Zohar, A. (2008). Interdomain Routing and Games. In *STOC*, pages 57–66.

Mattar, K., Epstein, S., and Matta, I. On the Complexity Policy Routing Dynamics. *Under submission to INFOCOM 2011*.

Mattar, K., Epstein, S., and Matta, I. On the Detection of Policy Conflicts in Interdomain Routing. *Under submission to INFOCOM 2011*.

Mattar, K., Epstein, S., and Matta, I. Principles of Safe Policy Routing Dynamics. *Under submission to IEEE/ACM Transactions on Networking.*

Mattar, K., Epstein, S., and Matta, I. (2010a). On the Complexity Policy Routing Dynamics. Technical Report BUCS-TR-2010-033, Computer Science Department, Boston University.

Mattar, K., Epstein, S., and Matta, I. (2010b). On the Detection of Policy Conflicts in Interdomain Routing. Technical Report BUCS-TR-2010-009, Computer Science Department, Boston University.

Obradovic, D. (2002). Real-time Model and Convergence Time of BGP. In *INFOCOM*.

Pei, D., Zhang, B., Massey, D., and Zhang, L. (2006). An analysis of Convergence Delay in Path Vector Routing Protocols. *Computer Networks*, 50(3):398–421.

Quoitin, B., Pelsser, C., Bonaventure, O., and Uhlig, S. (2005). A Performance Evaluation of BGP-based Traffic Engineering. *International Journal of Network Management*, 15(3):177–191.

Sami, R., Shapira, M., and Zohar, A. (2009). Searching for Stability in Interdomain Routing. In *INFOCOM*.

Uhlig, S. and Bonaventure, O. (2004). Designing BGP-based Outbound Traffic Engineering Techniques for Stub ASes. *SIGCOMM Computer Communication Review*, 34(5):89–106.

Varadhan, K., Govindan, R., and Estrin, D. (1996). Persistent Route Oscillations in Inter-domain Routing. *Computer Networks*.

Wang, F., Gao, L., and Qiu, J. (2005). On Understanding of Transient Interdomain Routing Failures. In *ICNP*.

Wang, F., Qiu, J., Gao, L., and Wang, J. (2009). On Understanding Transient Interdomain Routing Failures. *IEEE/ACM Transactions on Networking*, 17(3):740–751.

Yilmaz, S. and Matta, I. (2007). An Adaptive Management Approach to Resolving Policy Conflicts. In *IFIP Networking*, Atlanta, Georgia.

Curriculum Vitae

Karim A. Mattar

Department of Computer Science
Boston University
111 Cummington Street, MCS 138
Boston, MA, 02215

(617) 833 - 8635
kmattar@cs.bu.edu
http://cs-people.bu.edu/kmattar
http://www.linkedin.com/in/kmattar

EDUCATION

PhD, Computer Science; 11/2010
Boston University, Boston, MA (1/2004 — 11/2010)
Advisor: Dr. Ibrahim Matta

- Thesis — Policy Routing Dynamics: Theory and Applications

Master of Art, Computer Science; 1/2008
Boston University, Boston, MA (1/2004 — 1/2008)
Advisors: Dr. Ibrahim Matta and Dr. Azer Bestavros

- Thesis — TCP over CDMA2000 Networks: A Cross-Layer Measurement Study
- **GPA: 3.76/4.0**

Bachelor of Science, Computer Systems Engineering; 5/2003
University of Massachusetts, Amherst, MA (9/1999 — 5/2003)

- Minor in Computer Science
- **GPA: 3.88/4.0**, Summa Cum Laude

RESEARCH INTERESTS

Internet routing, transport, measurement and clean-slate architectures; scheduling and resource allocation in cellular data networks; algorithms; machine learning; data mining and modeling; statistical inference; game theory

WORK

Research Fellow, Boston University, Boston, MA (1/2004 — 11/2010)
Experience Advisor: Dr. Ibrahim Matta. Projects include: (1) modeling the dynamics of policy routing protocols, (2) developing a clean-slate Internet architecture, (3) developing scheduling, resource allocation and transport solutions for cellular data networks
[Java, C/C++, Tcl, Perl, MATLAB, Datalog]

Teaching Fellow, Boston University, Boston, MA (1/2004 — 5/2010)

- *CS 111: Computer Science I* with Dr. Dave Sullivan **[Java]**
- *CS 320: Algorithms* with Dr. John Byers
- *CS 455/655: Computer Networks* with Dr. Ibrahim Matta **[Java, C/C++]**
- *CS 112: Data Structures and Algorithm Analysis* with Dr. William Mullally **[Java]**
- *CS 556: Advanced Computer Networks* with Dr. Ibrahim Matta **[Java, C/C++, Tcl]**

Network Consultant, Movik Inc., Littleton, MA (7/2008 — 3/2009)
Built a simulation testbed in the network simulator ns2 to evaluate the performance of applications on mobile devices in 3G/UMTS networks running various proxy-based solutions
[C/C++, Tcl, Perl, MATLAB]

Research Intern, Sprint Labs, Burlingame, CA (6/2006 — 12/2006)
Developed an application-level transport solution for Sprints CDMA2000 EV-DO cellular network and evaluated it using the network simulator ns2
[Java, C/C++, Tcl, MATLAB, Perl]

Research Intern, Sprint Labs, Burlingame, CA (5/2005 — 8/2005)
Evaluated the performance of TCP over Sprints CDMA2000 1xRTT cellular network using a custom probing and measurement application
[Java, C/C++, Tcl, MATLAB, Perl]

Research Assistant, UMass Amherst, Amherst, MA (5/2002 — 5/2003)
Advisor: Dr. Lixin Gao. Implemented a tool to aid in visualizing BGP routing information, relationships between autonomous systems and the Internet hierarchy
[Java]

Teaching Assistant, UMass Amherst, Amherst, MA (1/2003 — 5/2003)
ECE 242: Data Structures and Algorithms with Dr. Lixin Gao
[Java]

GRADUATE COURSES

Programming Languages, Computer Networks (I/II), Cryptography, Probability in Computing, Algorithms, Operating Systems, Databases, Performance Analysis, Machine Learning, and Information Theory

GRADUATE COURSE PROJECTS

- **Machine Learning**
  Developed code for training and testing a support vector machine classifier with a nonlinear kernel; implemented the markov chain monte carlo technique for approximate inference in bayesian networks; implemented the multi-class decision tree method
  [MATLAB]
- **Performance Analysis**
  Implemented a discrete-event simulator for a G/G/1 queue
  [MATLAB]
- **Computer Networks I/II**
  Implemented a client-server application to probe and measure end-to-end path performance; implemented a reliable transport protocol; implemented a distance vector routing protocol; evaluated the fairness of several transport protocols and simulated a low-rate denial-ofservice attack on TCP using the network simulator ns2
  [Java, C/C++, Tcl]
- **Databases**
  Implemented several buffer management policies for the backend server of PostgreSQL; designed and implemented a database system for a web-based picture sharing application
  [Java, JSP, HTML, SQL]
- **Operating Systems**
  Implemented a dynamic thread pool, with priorities to support quality of service, for a video server to disseminate streaming video to clients over the Internet; implemented a file system as a loadable module in the Linux Kernel
  [C/C++]

SKILLS

Platforms: Unix, Linux, Windows
Programming: Java, Perl, MATLAB, C/C++, Tcl, Datalog, Scheme, Bash, Shell
Web Technologies: HTML, CSS, Javascript, Java Servlets, XML, JSP
Databases: PostgreSQL
Languages: Fluent spoken/written English/Arabic; Intermediate French
Leadership: Strong interpersonal, communication and leadership skills

HONORS AND AWARDS

- Teaching Fellow Award, College of Arts and Sciences, Boston University (2009-2010)
- Travel Grant Award for the Passive and Active Measurements Conference, Belgium (2007)
- Presidential University Graduate Fellowship, Boston University (2004)
- Fidelity Technology Fellows Scholarship, 2 years (20012003)
- Biography published in the National Deans List (2003)
- Commonwealth College and Departmental Honors, UMass Amherst (2003)
- Deans List, 8 semesters, College of Engineering, UMass Amherst (1999 — 2003)
- Member of the National Society of Collegiate Scholars (2003)

THESES

TCP over CDMA2000 Networks: A Cross-Layer Measurement Study
Masters Thesis, Department of Computer Science, Boston University
Technical Report BUCS-TR-2007-016, Dec 14, 2007

PUBLICATIONS

Declarative Transport: A Customizable Transport Service for the Future Internet
Karim Mattar, Ibrahim Matta, John Day, Vatche Ishakian and Gonca Gursun
Networking Meets Databases (NetDB), Montana, USA, 2009

Principles of Safe Policy Routing Dynamics
Samuel Epstein, Karim Mattar and Ibrahim Matta
International Conference on Network Protocols (ICNP), New Jersey, USA, 2009

Networking is IPC: A Guiding Principle to a Better Internet
John Day, Ibrahim Matta and Karim Mattar
Re-Architecting the Internet (Re-Arch), Madrid, Spain, 2008

TCP over CDMA2000 Networks: A Cross-Layer Measurement Study
Karim Mattar, Ashwin Sridharan, Hui Zang, Ibrahim Matta and Azer Bestavros
Passive and Active Measurement Conference (PAM), Louvain-la-neuve, Belgium, 2007

TECHNICAL REPORTS

On the Complexity of Policy Routing Dynamics
Karim Mattar, Samuel Epstein and Ibrahim Matta
Technical Report BUCS-TR-2010-033, Sep 12, 2010

On the Detection of Policy Conicts in Interdomain Routing
Karim Mattar, Samuel Epstein and Ibrahim Matta
Technical Report BUCS-TR-2010-009, Apr 16, 2010

Principles of Safe Policy Routing Dynamics
Samuel Epstein, Karim Mattar and Ibrahim Matta
Technical Report BUCS-TR-2009-013, Apr 21, 2009

On the Performance and Robustness of Managing Reliable Transport Connections
Gonca Gursun, Ibrahim Matta and Karim Mattar
Technical Report BUCS-TR-2009-014, Apr 21, 2009

Foundational Theory for Understanding Policy Routing Dynamics
Karim Mattar, Samuel Epstein and Ibrahim Matta
Technical Report BUCS-TR-2009-001, Jan 30, 2009

Networking is IPC: A Guiding Principle to a Better Internet
John Day, Ibrahim Matta and Karim Mattar
Technical Report BUCS-TR-2008-019, Aug 15, 2008

An Online Distributed Algorithm for Inferring Policy Routing Configurations
Samuel Epstein, Ibrahim Matta and Karim Mattar
Technical Report BUCS-TR-2008-017, Aug 15, 2008

Declarative Transport: No more transport protocols to design only policies to specify
Karim Mattar, Ibrahim Matta, John Day, Vatche Ishakian and Gonca Gursun
Technical Report BUCS-TR-2008-014, Jul 12, 2008

TCP over CDMA2000 Networks: A Cross-Layer Measurement Study
Karim Mattar, Ashwin Sridharan, Hui Zang, Ibrahim Matta and Azer Bestavros
Technical Report BUCS-TR-2006-030, Oct 25, 2006

On the Interaction between TCP and the Wireless Channel in CDMA2000 Networks
Karim Mattar, Ashwin Sridharan, Hui Zang, Ibrahim Matta and Azer Bestavros
Technical Report BUCS-TR-2006-009, Jun 6, 2006

TALKS

Declarative Transport: A Customizable Transport Service for the Future Internet
Networking Meets Databases (NetDB), Montana, USA, 2009

Principles of Safe Policy Routing Dynamics *International Conference on Network Protocols (ICNP), New Jersey, USA, 2009*

TCP over CDMA2000 Networks: A Cross-Layer Measurement Study *Passive and Active Measurements Conference (PAM), Belgium, 2007*

On the Interaction between TCP and the Wireless Channel in CDMA2000 Networks *Workshop on Wireless Traffic Measurements and Modeling (WiTMeMo), Boston, MA, 2006*

SPONSORED RESEARCH

- NSF Grant for Building a Recursive Internet Architecture, $560K (5/2010 5/2013)
  Research Principal Investigator: Dr. Ibrahim Matta
- Movik Support for Application Performance over 3G/UMTS, $40K (7/2008 3/2009)
  Principal Investigator: Dr. Ibrahim Matta
- Sprint Support for Transport over Cellular Data Networks, $35K (9/2006 8/2007)
  Principal Investigator: Dr. Ibrahim Matta (co-PI: Dr. Azer Bestavros)
- Sprint Support for Characterizing Cellular Data Networks, $20K (5/2005 8/2005)
  Principal Investigator: Dr. Ibrahim Matta (co-PI: Dr. Azer Bestavros)

ACADEMIC SERVICES

Reviewer for

- IEEE International Conference on Network Protocols (2004, 2010)
- Transactions on Mobile Computing (2009)
- Passive and Active Measurements Conference (2008)
- ACM SIGMETRICS (2007)
- IEEE INFOCOM (2007, 2006)
- IEEE Symposium on Computers and Communications (2005)
- International Conference on Mobile and Ubiquitous Systems (2005)

Volunteer for

- IEEE International Conference on Network Protocols (2005)
- Fourth Workshop on Applications and Services in Wireless Networks (2004)

VISA STATUS

F-1 Student

REFERENCES

Available upon request