2008-08-15

An Online Distributed Algorithm for Inferring Policy Routing Configurations

Epstein, Samuel
Boston University Computer Science Department

An Online Distributed Algorithm for Inferring Policy Routing Configurations

Samuel Epstein  Ibrahim Matta  Karim Mattar
Computer Science
Boston University
Email: {samepst, matta, kmattar}@cs.bu.edu
Technical Report BUCS-TR-2008-017

Abstract—We present an online distributed algorithm, the Causation Logging Algorithm (CLA), in which Autonomous Systems (ASes) in the Internet individually report route oscillations/flaps they experience to a central Internet Routing Registry (IRR). The IRR aggregates these reports and may observe what we call causation chains where each node on the chain caused a route flap at the next node along the chain. A chain may also have a causation cycle. The type of an observed causation chain/cycle allows the IRR to infer the underlying policy routing configuration (i.e., the system of economic relationships and constraints on route/path preferences).

Our algorithm is based on a formal policy routing model that captures the propagation dynamics of route flaps under arbitrary changes in topology or path preferences. We derive invariant properties of causation chains/cycles for ASes which conform to economic relationships based on the popular Gao-Rexford model. The Gao-Rexford model is known to be safe in the sense that the system always converges to a stable set of paths under static conditions. Our CLA algorithm recovers the type/property of an observed causation chain of an underlying system and determines whether it conforms to the safe economic Gao-Rexford model. Causes for nonconformity can be diagnosed by comparing the properties of the causation chains with those predicted from different variants of the Gao-Rexford model.

I. INTRODUCTION

The Border Gateway Protocol (BGP) is currently the de-facto inter-domain routing protocol employed in the Internet. BGP allows Autonomous Systems (ASes), operated by different administrative domains (e.g., Internet Service Providers, companies, universities) to independently apply local policies for selecting routes and propagating routing information. Given the critical role and global scope of BGP, both its transient and steady-state performance have received significant attention, and problems related to delayed convergence [1] and potential instability [2], [3] (i.e., route oscillations/flaps) have been identified and studied.

Route flaps, in particular, could be highly disruptive given the associated cost of communication and processing overheads. Route flaps can be transient (i.e., short-term) due to temporary changes in topology or route/path preferences. Route flaps can also be persistent due to routing policies across ASes that are conflicting (i.e., policies can not be simultaneously satisfied) [4]. We refer to such “conflicting” policies as an unsafe (policy) configuration.

Economic constraints that are typical of commercial relationships between ASes in the Internet—henceforth referred to as Gao-Rexford model [5]—have been shown to make BGP safe. For example, in the Gao-Rexford model, economic relationships between ASes and constraints on path preferences are defined as follows: an AS classifies its neighboring ASes based on their economic relationship, as either customer, provider, or peer. And the path preferences are restricted in a hierarchical fashion, e.g., every AS prefers a path through a customer AS over a path through a peer or provider AS.

Our Contribution:

Our goal in this paper is to develop an online distributed algorithm for observing and recording route flaps to infer the nature of the underlying policy configuration. Such an algorithm should be efficient, and should maintain privacy, i.e., the routing policies of an AS should not be revealed to other ASes.

To that end, we extend the static model of BGP [4] to capture the propagation dynamics of route updates under arbitrary changes in topology or path preferences. We call this extended model, Dynamic Policy Routing (DPR) model. We introduce the notion of causation chains, where informally, the route flap at a node on the chain causes a route flap at the next node along the chain. We also define causation cycles of different types; for example, a causation cycle is simple if it starts and terminates at the same node. (We give more formal definitions of these concepts later in the paper.)

The goal is then to find properties of these causation chains/cycles invariant to arbitrary changes in topology or path preferences. We start with the Gao-Rexford model and characterize economic relationships between adjacent ASes in causation chains (similar to the way paths are characterized [5]). For example, we prove that all causation chains in the Gao-Rexford model do not contain a provider-to-customer-to-provider sequence, referred to as “valley-free”, thus generalizing the result in [6] to time-varying topologies and path preferences. Thus, observing these properties allow us to infer whether or not the policy configuration conforms to the Gao-Rexford model and hence its safety.

In order to diagnose nonconformity, we relax the conditions of Gao-Rexford and consider six “violations” or variants of
the model. For example, a less-restricted variant may allow an AS to prefer a path through its peer over a path through its customer—there may be legitimate economic reasons for these more relaxed routing policies [7], [8]. We again characterize the resulting causation chains/cycles for each Gao-Rexford variant. Causes for nonconformity can be diagnosed by comparing the properties of the observed causation chains against those predicted by each variant.

The above machinery allows us to develop a distributed inference algorithm, called the Causation Logging Algorithm (CLA), that can be used to test and diagnose conformance to the Gao-Rexford model. CLA requires ASes to record and report to a central repository, e.g., a trusted Internet Routing Registry (IRR) [9], their locally-observed causation tuples, that consist of itself and a neighboring node that caused its route flap. These tuples/messages are of small size and can be aggregated by the IRR to identify causation chains/cycles, which based on their type, the IRR can determine whether or not the underlying system conforms to Gao-Rexford. CLA has the following main features:

- It is efficient in the sense of small message / communication overhead, as well as low processing overhead since only observed/realized route flaps are used—this is in contrast to the static checking of all possible conflicts in routing policies (whether or not they are actually realized), which is known to be NP-complete [10].
- It does not require an AS to reveal its private routing policies to other ASes since only locally-observed route flaps are reported to a trusted IRR—this is in contrast to exchanging route flaps among ASes in extended “history” messages [11], where an AS reveals its path preferences to every other AS as it adopts new paths and abandons old paths during route flaps.
- It is online and distributed, diagnosing actual behavior of ASes. Therefore inferring the conformity (or lack thereof) to Gao-Rexford policy configurations is done incrementally based only on observed route flaps. This is in contrast to offline methods which use information from BGP tables [12]. Furthermore, not all ASes have to report their causation tuples, though in these cases, the inference result from the IRR applies only to those segments of the network that adopt our algorithm.

Paper Outline:

The rest of the paper is organized as follows: Section II introduces our dynamic DPR model. Section III formalizes the Gao-Rexford model in the context of DPR and shows that certain properties of causation chains and cycles are invariant to dynamic changes in the underlying topology or route/path preferences. Section IV considers several variants/violations of the Gao-Rexford model. We show the different types of causation chains/cycles that may result under each Gao-Rexford variant. In Section V, we present our distributed online algorithm, the Causation Logging Algorithm, that uses observed causation sequences to test and diagnose conformance to the Gao-Rexford model. Section VI concludes the paper.

II. DYNAMIC POLICY ROUTING MODEL

The Dynamic Policy Routing (DPR) model is used to capture the dynamics of BGP inter-domain routing. Each Autonomous System (AS) is represented by a node in a graph. AS path preferences are represented by a ranking relation. The following section describes DPR, extending the notation of [4]. The major addition of DPR is that it models time-varying topologies and path preferences.

A. Basics of DPR

Definition 1 (Time). Time is represented by a non-negative, discrete index \( t \) such that: \( t = [0, \infty) \).

Definition 2 (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 \( t \) is represented by \( (u, v)^t \notin E \).

There exists a distinguished destination node, represented as \( \text{root} \), where \( \text{root} \in V \).

Definition 3 (Paths). Paths are sequences of nodes: \( \langle u_1, u_2, \ldots, u_k \rangle \). The empty path is denoted by \( \langle \rangle \). All paths should end with the \( \text{root} \) node. The goal of every node is to find a path to \( \text{root} \). A concatenation of a node \( u \) with a path \( Q \) is represented as: \( P = \langle u \mid Q \rangle \). A path originating from \( u \) is represented by \( P^u \). The set of paths originating from \( u \) is represented by \( \mathcal{P}^u \).

Definition 4 (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:

\[
P^u \succeq^t Q^u \Leftrightarrow P^u \succeq^t Q^u \text{ and } Q^u \nprec^t P^u
\]

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 \( \text{root} \). 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 5 (DPR Instance). A Dynamic Policy Routing (DPR) instance consists of a graph and a path preference relation \( D = (\succeq, G) \).

Definition 6 (States). At each time index \( t \), every node \( u \) has a path to \( \text{root} \), represented by \( P^u \succeq \pi(u, t) \). The available path choices of a node, via all possible neighbors \( v \), are represented by \( \text{Choices}(u, t) \) where:

\[
\text{Choices}(u, t) = \langle \rangle \cup \{ \langle u, \pi(v, t) \rangle; (u, v)^t \in E \}
\]

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

\[
\text{Best}(u, t) = \max_{\succeq^t} \text{Choices}(u, t)
\]
The states of nodes at each time $t$ is their best path of the previous round. For all nodes $u \in V$:

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

The path used by node $u$ at time $t$, $\pi(u, t)$, was its best path at time $t-1$, $\text{Best}(u, t-1)$. This best path was determined using the ranking $\geq t-1$. The $\rho$ notation is used to represent the next-hop neighbor of a current path:

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

**Definition 7 (Realized Paths).** A path $P^u$ is realized iff there exists a time $t$ such that $\pi(u, t) = P^u$.

**Theorem 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 $\emptyset \succ P^u$ so $P^u \neq \text{Best}(u, t-1)$ which is a contradiction. ■

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

**B. Causation in DPR**

**Definition 8 (Path Rank Changes).** The following definitions describe the relative change in the rankings of selected paths for a node:

$$\begin{align*}
\text{RankDec}(u, t) & \iff \pi(u, t) \succ t \pi(u, t + 1) \\
\text{RankInc}(u, t) & \iff \pi(u, t) \prec t \pi(u, t + 1) \\
\text{RankSame}(u, t) & \iff \pi(u, t) = \pi(u, t + 1)
\end{align*}$$

The relative change in rankings are with respect to the current time index’s path ranking $\succeq t$.

**Definition 9 (Causation Function).** During the course of the DPR model, a node $u$ may change its current path at a given time $t$. The causation function represents $u$’s neighboring node $v$ responsible for $u$’s path change. Causation function is the base construct from which causation chains will be built. A causation function $C$ maps each node $u$ at a given time $t$ to a neighboring node $v$:

$$C(u, t) = v$$

**III. GAO-REXFORD MODEL**

This section will show that if a DPR instance conforms to a set of economic relationships known as the Gao-Rexford model [5], then its dynamic behavior can be characterized, regardless of changes in topology or path preferences. In particular, we will show that all causation chains have the
property known as “valley-free” and all causation cycles are simple. Any deviation from these properties would imply that some nodes are operating in a non-economic fashion, forming the basis for the Causation Logging Algorithm in Section V.

Gao and Rexford [5] proposed restrictions on path rankings that reflected the economic relationships in the internet:

- Every node is customer, peer, or provider to its neighboring nodes.
- A node cannot be a provider to itself. There are no customer-provider cycles.
- For all times, each node prefers a path through a customer over a path through a peer or a provider. Likewise, each node prefers a path through a peer over a path through a provider.
- Each node provides transit service only to its customers. Paths are forbidden to have “valleys” as shown later in the section.

For simplicity, we will further restrict our attention to “strict” Gao-Rexford conditions. The strict Gao-Rexford conditions restrict the standard Gao-Rexford conditions by forbidding a node from being both a (direct or indirect) provider and a (direct or indirect) peer to another node. Figure 2 shows a direct comparison between the standard and strict Gao-Rexford models.  

![Fig. 2. Strict and Standard Economic Relationships. In the strict version, node u is no longer an indirect provider and peer to node z.](image)

Gao and Rexford [5] showed that standard (and thus the strict as well) Gao-Rexford conditions are sufficient to guarantee stability in a static graph. Thus both the strict and standard Gao-Rexford conditions are safe.

### A. Modeling Strict Gao-Rexford using DPR

This section shows that every causation chain (including cycles) of a strict Gao-Rexford DPR is valley-free and every causation cycle is simple. The restrictions of the strict Gao-Rexford model enable equivalence classes of peers, as seen in figure 3. Thus, the economic relationships between nodes can be represented using a pre-order relation.

**Definition 12** (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 tight economic relation is defined by:

$$u \succ_v w \text{ iff } u \succeq v \text{ and } u \not\preceq v$$

and an equivalence relation is defined by:

$$u =_v w \text{ iff } u \succeq v \text{ and } u \succeq w$$

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 transitive properties of the economic operator $\succeq$ can be modeled using pre-order conditions:

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

Note the following transitive relationships hold:

$$x \succ y \text{ and } y \succeq z \text{ implies } x \succ z$$
$$x \succeq y \text{ and } y \succ z \text{ implies } x \succeq z$$

**Definition 13** (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 $P^u$:

- Customer($P^u$) $\iff$ $u \succ \text{NextHop}(P^u)$
- Peer($P^u$) $\iff$ $u = \text{NextHop}(P^u)$
- Provider($P^u$) $\iff$ $u \prec \text{NextHop}(P^u)$

**Definition 14** (Valley-Free Sequences). Both paths and causation chains can be characterized by the economic relationships between their adjacent nodes. A sequence $(u_0 \ldots u_k)$ has a valley with respect to an economic operator $\succeq$ if there exists an $i$ such that $0 < i < k$ and:

$$u_{i-1} \succeq u_i \not\succeq u_{i+1}$$

The four cases of valleys can be seen in figure 4. Likewise, the sequence is valley-free if for all $i$, $0 < i < k$:

$$u_{i-1} \succeq u_i \Rightarrow u_i \succeq u_{i+1}$$

and equivalently

$$u_i \succeq u_{i+1} \Rightarrow u_{i-1} \prec u_i$$
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.

**Definition 15** (Strict Gao-Rexford Instances). An economic DPR instance \((\succeq_s, \succeq, G)\) satisfies the strict Gao-Rexford conditions if:
1) All paths which are not valley-free are forbidden.
   \[\text{HasValley}(P) \Rightarrow \langle \rangle > P\]
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 \(P_2^u\):
   \[\text{Customer}(P_1^u) \text{ and not } \text{Customer}(P_2^u) \Rightarrow P_1^u > P_2^u\]
   \[\text{Peer}(P_1^u) \text{ and } \text{Provider}(P_2^u) \Rightarrow P_1^u > P_2^u\]

**B. Causation Chains in the Gao-Rexford Model**

This section characterizes causation chains for strict Gao-Rexford DPR instances. For convenience of notation, we will drop the time index of certain terms with respect to a given chain \(Y = \langle y_0 y_1 \ldots y_k \rangle^t\), namely:

\[
\begin{align*}
\pi(y_i) &= \pi(y_i, t + i) \\
\pi_{\text{next}}(y_i) &= \pi(y_i, t + i + 1) \\
\rho(y_i) &= \rho(y_i, t + i) \\
\rho_{\text{next}}(y_i) &= \rho(y_i, t + i + 1) \\
\text{RankDec}(y_i) &\iff \text{RankDec}(y_i, t + i) \\
\text{RankSame}(y_i) &\iff \text{RankSame}(y_i, t + i) \\
\text{RankInc}(y_i) &\iff \text{RankInc}(y_i, t + i)
\end{align*}
\]

**Theorem 2.** Every causation chain of a strict Gao-Rexford DPR instance \((\succeq_s, \succeq, G)\) 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 \(y_{i-1} \succeq_s y_i \succeq y_{i+1}\). 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 in strict Gao-Rexford DPR instances, this results in a contradiction.

Since \(C(y_i) = y_{i-1}\), either the first or second condition of causation from table I holds for \(y_i\) at time \(t + i\).

**Case: \(y_i\), First Causation Condition**
If the first condition holds for \(y_i\), then: \(\rho(y_i) = y_{i-1}\) and \(\text{RankDec}(y_i)\), as shown in figure 6.

**Case: \(y_i\), Second Causation Condition**
If the second condition of table I holds for \(y_i\), then: \(\rho_{\text{next}}(y_i) = y_{i-1}\) and \(\text{RankInc}(y_i)\) as seen in figure 8.

Therefore \(\pi(y_i) >^{t+i} \pi_{\text{next}}(y_i)\). Let \(v = \rho_{\text{next}}(y_i)\). It cannot be that \(v \succeq_s y_i\). Otherwise, since \(\pi(y_i)\) is a customer path and \(\pi_{\text{next}}(y_i)\) is not (since \(\rho_{\text{next}}(y_i) = y_{i-1} \succeq_s y_i\)), by the conditions of strict Gao-Rexford instances: \(\pi(y_i) >^{t+i} \pi_{\text{next}}(y_i)\), causing a contradiction, as shown in figure 9. Thus \(\rho_{\text{next}}(y_i) \succeq_s y_i\) and \(v \succeq_s y_i\).
Fig. 9. Contradiction: $\text{RankDec}(y_i)$

So for both cases, at no time in the causation chain did $y_i$ have a customer path:

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

**Case: $y_{i+1}$ First Causation Condition**

If the first condition of causation holds for $y_{i+1}$, then $\rho(y_{i+1}) = y_i$. By lemma 1: $\pi(y_{i+1}) = \langle y_{i+1}, \pi(y_i) \rangle$. $\pi(y_{i+1})$ is a valley path since $y_{i+1} \succeq y_i \preceq \rho(y_i)$. Since all valley paths are forbidden, $\pi(y_{i+1})$ can never be realized, causing a contradiction.

**Case: $y_{i+1}$ Second Causation Condition**

Similar arguments can be used if the second condition of causation holds for $y_{i+1}$: $\rho_{\text{next}}(y_{i+1}) = y_i$. Thus by lemma 1: $\pi_{\text{next}}(y_{i+1}) = \langle y_{i+1}, \pi_{\text{next}}(y_i) \rangle$. $\pi_{\text{next}}(y_{i+1})$ 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.

**C. Causation Cycles in the Gao-Rexford Model**

This section describes properties of causation cycles in strict Gao-Rexford DPR instances. Figure 5 represents a causation cycle, where node $y_0$ loses a path to root and reroutes through $y_2$.

**Lemma 2.** Given a causation cycle $Y = \langle y_0, \ldots, y_k \rangle^t$ of a strict Gao-Rexford DPR instance $(\succeq_G, \preceq_G, G)$, every node in $Y$ is a provider to the primary node $y_0$.

**Proof:** Let $y_i \in Y$, where $0 < i < k$. By theorem 2, $Y$ is valley-free and either $y_{i-1} \preceq_G y_i$ or $y_i \preceq_G y_{i+1}$. If the first case is true, then by the definition of valley-free paths $y_{j-1} \preceq_G y_j$ for all $0 < j < i$, and by the transitive nature of economic relationships, $y_0 \preceq_G y_i$. If the second case is true, then by the definition of valley-free paths $y_j \preceq_G y_{j+1}$ for all $i < j < k$, and by the transitive nature of economic relationships, $y_0 \preceq_G y_k$. Thus every node $y_i$ is a provider to $y_0 = y_k$.

**Theorem 3.** Every causation cycle $Y = \langle y_0, \ldots, y_k \rangle^t$ of a strict Gao-Rexford DPR instance is simple.

**Proof:** Assume not. Then there exists a causation chain $Y_1 = \langle y_0, y_1, \ldots, y_{t-i} \rangle^t$ where $y_0 = y_k$. From lemma 2, $y_0 \preceq_G y_i$. However a new causation cycle $Y_{i+1}$ can be constructed such that:

$$Y_{i+1} = \langle y_1, y_2, \ldots, y_{k-1}, y_k, y_1 \rangle$$

Thus by lemma 2, $y_0 \preceq_G y_k = y_0$ which is a contradiction.

**IV. Variants of the Gao-Rexford Model**

The previous section described properties of causation chains/cycles in DPR instances which conform to the strict Gao-Rexford model. Along with the standard Gao-Rexford model, this section defines the properties of causation chains/cycles of six other violations/variants. The results of this section will be used in Section V to diagnosis nonconformity to the Gao-Rexford model. All the variants of the Gao-Rexford model can be defined by three binary parameters:

1) Strict or standard economic relationships
2) Allowance of sibling relationships
3) Preference of customer paths over peer paths

The first parameter was discussed in the previous section. We will describe binary parameters 2 and 3.

**A. Parameter 2: Allowance of Sibling Relationships**

For Gao-Rexford DPR instances with sibling relationships, the set of forbidden paths is reduced to allow transiting between peers. This peer transiting behavior is also known as sibling relationships [13]. Instead of “valley-free” paths, we refer to allowed paths as being “canyon-free”.

**Definition 16 (Canyon).** We define a canyon to be a sequence of three nodes $\langle a, b, c \rangle$ satisfying at least one of the following conditions:

1) $a \succeq_G b \preceq_G c$
2) $a \preceq_G b \succeq_G c$

**Fig. 5. Causation cycle $Y = \langle y_0, y_1, y_2 \rangle^t$ with causation conditions 1, 1, 2, and 2, respectively. 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 $t + 2$ and $y_2$ to switch to a path through $y_1$ at time $t + 3$. The cycle is closed with $y_0$ switching to a path via $y_2$ at time $t + 4$. Note the existence of a separate causation chain $Y\prime = \langle y_0 y_2 \rangle^t$.**
TABLE II
VARIANTS OF GAO-REXFORD MODELS

<table>
<thead>
<tr>
<th>Customer/Peer Paths</th>
<th>Sibling Relationships</th>
<th>Economic Relationships</th>
<th>Causation Chains</th>
<th>Vertical Cycles</th>
<th>Horizontal Cycles</th>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constrained</td>
<td>No</td>
<td>Strict</td>
<td>Valley-Free</td>
<td>Simple Only</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Constrained</td>
<td>No</td>
<td>Standard</td>
<td>Valley-Free</td>
<td>Simple Only</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Constrained</td>
<td>Yes</td>
<td>Standard</td>
<td>Canyon-Free</td>
<td>Simple Only</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Constrained</td>
<td>Yes</td>
<td>Standard</td>
<td>Ravine-Free</td>
<td>Simple Only</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Unconstrained</td>
<td>No</td>
<td>Strict</td>
<td>Ravine-Free</td>
<td>Simple Only</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Unconstrained</td>
<td>No</td>
<td>Standard</td>
<td>Ravine-Free</td>
<td>Simple Only</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Unconstrained</td>
<td>Yes</td>
<td>Strict</td>
<td>Ravine-Free</td>
<td>Simple Only</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Unconstrained</td>
<td>Yes</td>
<td>Standard</td>
<td>Ravine-Free</td>
<td>Simple Only</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Each canyon-free path is 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 10. Thus all paths with canyons in Gao-Rexford DPR instances with sibling relationships are forbidden.

B. Parameter 3: Preference of Customer Paths over Peer Paths

A Gao-Rexford model is unconstrained if peer paths can be ranked higher than customer paths. As it will be shown later in this section, all causation chains of unconstrained Gao-Rexford models are “Ravine-Free”.

Definition 17 (Ravine). We define a ravine to be a sequence of three nodes \( \langle a, b, c \rangle \) satisfying the condition:

\[
a \succneq b \preceq c
\]

Thus \( a \) is a provider to \( b \) and \( c \) is not a customer to \( b \). Ravines represent a subset of canyons, which represent a subset of valleys, as seen in figure 10.

C. Dynamic Behavior of Gao-Rexford Variants

The parameters discussed in the previous section describe different variants to the Gao-Rexford model. Each variant results in slightly different types of causation chains and cycles. In order to illustrate these differences, we partition causation cycles in to two categories: horizontal and vertical cycles.

Definition 18 (Horizontal Cycle). A causation cycle is horizontal if all adjacent nodes in the cycle are peers.

Definition 19 (Vertical Cycle). A causation cycle is vertical if there is at least one customer/provider relationship between adjacent nodes of the cycle. An example of a simple vertical cycle can be found in figure 5. An example of a non-simple vertical cycle can be found in figure 11.

Table II describes the properties of causation chains and cycles of all Gao-Rexford model and its variants. The vertical/horizontal cycles columns describe the types of causation cycles: “Yes” indicates both simple and non-simple cycles, whereas “Simple Only” indicates the appearance of only simple cycles. The proofs for table II can be found in Appendices B and A.

The first and second rows represent the strict and standard versions of the Gao-Rexford model described in Section III. They represent the only variants which are guaranteed to be safe. They represent conformity to the Gao-Rexford model. The six other variants describe violations to the model, and thus such instances can be potentially unsafe.

V. DETECTING GAO-REXFORD CONFORMITY

A. Causation Logging Algorithm

By examining the causation chains/cycles and economic relationships of a DPR instance, one can determine whether the DPR instance conforms to the behavior predicted by the Gao-Rexford model. For any period of routing instability, a simple distributive logging algorithm can be implemented by each Autonomous System. The logs can be combined together at a central Internet Routing Repository (IRR) which can detect any non-economic (Gao-Rexford) behavior. The results are completely independent of dynamic changes in topology or path preferences.

Figure 12 describes the Causation Logging Algorithm, which represents the computation and logging of the causation function during routing. Lines 2 and 3 represent standard routing behavior, which determines node \( u \)’s chosen route for the next time period. Lines 4 through 9 represent the logging of the causation tuples for the given time period. Each causation tuple consists of:

\[
\{v, t - 1\} - \text{causation neighbor } v \text{ and previous time } t - 1
\]

\[
\$\{v, u\} - v \text{’s economic relationship with current node } u
\]

\[
\{u, t\} - \text{node } u \text{ and time } t
\]
Fig. 11. A non-simple vertical cycle. Nodes a and b are peers. Node c is a customer of a and b and the root node is customer of every other node. Nodes a, b, and c always prefer a path through b, c, and a over a direct path to root, respectively. Paths going through all nodes a, b, and c are forbidden. This DPR instance corresponds to an unsafe “bad gadget” described in [4]. The corresponding causation chain represents a non-simple cycle: $Y = (b \ a\ c\ b\ a)^t$.

Fig. 12. Causation Logging Algorithm

```
1: function PROCESS(u, t)
2:  Best(u, t) ← max_{s \in \mathcal{S}} Choices(u, t)
3:  π(u, t + 1) ← Best(u, t)
4:  if RankInc(u, t) then
5:     v ← ρ(u, t + 1)
6:  else if RankDec(u, t) then
7:     v ← ρ(u, t)
8:  if v ≠ ∅ then
9:     Log(\{v, t - 1\}, \$(v, u), \{u, t\})
```

B. Central IRR

The causation tuples from every node over a specific time span is sent to a central IRR. The combined set of causation tuples $\mathcal{T}$ can be represented as:

$$\left(\{v, s\}, \$(v, u), \{u, t\}\right) \in \mathcal{T}$$

From $\mathcal{T}$, the IRR creates a causation digraph $G^c = (V^c, E^c)$. The vertices, $V^c$, consist of the node/time pairs in $\mathcal{T}$:

$$\left(\{v, s\}, \$(v, u), \{u, t\}\right) \in \mathcal{T} \Rightarrow \{v, s\} \in V^c \text{ and } \{u, t\} \in V^c$$

The edges, $E^c$, have the exact same form as the tuples of $\mathcal{T}$, representing directed edges, annotated with economic relationships:

$$\left(\{v, s\}, \$(v, u), \{u, t\}\right) \in \mathcal{T} \Rightarrow \left(\{v, s\}, \$(v, u), \{u, t\}\right) \in E^c$$

Every path $H$ in $G^c$ represents a causation chain in the corresponding DPR model. The path $H$ also represents a causation cycle in the DPR instance if $H$ contains two elements which share a DPR node:

$$\{u, s\} \in H \text{ and } \{u, t\} \in H$$

Figure 13 shows the causation tuples and corresponding causation graph of the DPR instance shown in figure 5. The economic annotations of the edges in the causation graph are represented by their slope: edges with horizontal slope are between peers, while edges with diagonal slopes are between customers and providers. The paths in the causation graph show that the DPR instance had a simple vertical causation cycle $(y_0 y_1 y_2 y_0)^t$.

Using the economic annotations of the edges, the IRR can analyze the causation chains and cycles represented by the paths of $G^c$. The valleys, canyons, and ravines for each causation chain can be determined. Causation cycles can be classified as being vertical, horizontal, and/or simple. As shown in figure 14, the IRR can determine whether a DPR instance conforms to the Gao-Rexford model. Given non-conformity, the causation chains/cycles are compared against the results of table II to eliminate possible reasons for the violation. If a causation chain does not adhere to a particular violation/variant—for example, a non-simple vertical cycle in the (constrained, sibling, strict) variant—then the cause for nonconformity cannot be due to that violation/variant. The outputs of figure 14 are sent to the offending nodes: the members of the ravine, canyon, valley or cycle.

The running time for determining ravines, canyons and valleys is linear in the number of edges: $O(E^c)$. As shown in [14], the running time for finding the number of elementary cycles is on the order of: $O((V^c + E^c)(N + 1))$, where $N$ is the number causation cycles found. This is more tractable than determining if a static BGP configuration is stable, which
and proved their
to their neighbor’s current local times. A path change causes a node to log the following causation tuple:
\[
\{v, t_v\} - \text{causation neighbor } v \text{ and its local time } t_v
\]
\[
\$\{v, u\} - v’s \text{ economic relationship with node } u
\]
\[
\{u, t_u\} - \text{node } u \text{ and its local time } t_u
\]
The tuples are sent to the IRR, which computes Gao-Rexford conformity in the same fashion as the original algorithm.

VI. SUMMARY AND FUTURE WORK
We introduced a Dynamic Policy Routing (DPR) model, which extends the static model of the Border Gateway Protocol (BGP), to capture the propagation dynamics of route flaps due to arbitrary changes in topology or path preferences.

We introduced the concept of causation chains and cycles, and proved their invariant properties for the Gao-Rexford policy configuration model and seven other variants. The Gao-Rexford model imposes economic constraints on the relationships between neighboring Autonomous Systems (ASes) and on the individual path preferences of each AS, to ensure safe/stable behavior.

Based on these causation properties, we developed an online distributed algorithm, the Causation Logging Algorithm, wherein each AS reports its locally-observed causation events to a central Internet Routing Repository (IRR). The IRR aggregates these reports to form a view of dependencies between the route updates at various ASes. Based on the type/property of causation chains/cycles that are detected, the IRR can infer if the underlying policy configuration conforms to one of the economic Gao-Rexford models and if not, which variants the configuration does not adhere to.

The inference problem we considered in this paper, takes as input the causation reports/tuples from each AS, where a tuple consists of the reporting AS-identifier, the neighboring AS-identifier that caused it to route-flap, and the economic relationship between the two ASes. The output of the inference algorithm is whether the underlying system of constraints on economic relationships and path preferences conforms to the Gao-Rexford model.

Future work includes expanding the set of Gao-Rexford variants, beyond the eight variants considered in this paper. This will allow us to diagnose a larger class of (less constrained) routing policies. In addition, our DPR model captures finer grained transient dynamics that could be explored. We believe that DPR can be used derive stricter conditions for BGP divergence than the ones derived using the static model [4]. This would allow for faster and more efficient detection of anomalous/non-conforming policy configurations.

REFERENCES
APPENDIX A
PROOFS FOR STRICT VARIANTS OF GAO-REXFORD MODELS

The following appendix section shows the proofs of table II for Gao-Rexford model variants which have strict economic relationships. We use the notation $D($Strict$)$ to specify that DPR instance $D$ has strict economic relationships described in Section III. To fully characterize DPR instances, we formally define DPR parameters 2, and 3 described in Section IV. The abbreviated notation used in Section III-B will be employed for brevity.

A. Parameter 2: Sibling Relationships

A DPR instance without sibling relationships, $D($NoSib$)$ only permits valley-free paths. Thus if a path contains a valley (from definition 14), it is forbidden. A DPR instance with sibling relationships, $D($Sib$)$, extends the set of permitted paths, to those that are canyon-free. Thus if a path contains a canyon (from definition 16), it is forbidden.

B. Parameter 3: Preference of customer over peer paths

In a constrained DPR instance, $D($Const$)$, customer paths are preferred over peer/provider paths and peer paths are preferred over provider paths. Thus given two paths $P_{1}^{u}$ and $P_{2}^{u}$:

Customer($P_{1}^{u}$) and not Customer($P_{2}^{u}$) $\Rightarrow$ $P_{1}^{u}$ $\succ$ $P_{2}^{u}$

Peer($P_{1}^{u}$) and Provider($P_{2}^{u}$) $\Rightarrow$ $P_{1}^{u}$ $\succ$ $P_{2}^{u}$

In an unconstrained DPR instance, $D($Unconst$)$, customer and peer paths are preferred over provider paths. However peer paths can be preferred over customer paths. Thus given two paths $P_{1}^{u}$ and $P_{2}^{u}$:

Customer($P_{1}^{u}$) and Provider($P_{2}^{u}$) $\Rightarrow$ $P_{1}^{u}$ $\succ$ $P_{2}^{u}$

Peer($P_{1}^{u}$) and Provider($P_{2}^{u}$) $\Rightarrow$ $P_{1}^{u}$ $\succ$ $P_{2}^{u}$

C. Model variant $D($Const,NoSib,Strict$)$

Theorem 4. Every causation chain in model variant $D($Const,NoSib,Strict$)$ is valley-free.

Proof: This follows directly from theorem 2.

Theorem 5. Every causation cycle in model variant $D($Const,NoSib,Strict$)$ is vertical and simple.

Proof: This follows from lemma 2 and theorem 3.

Theorem 6. Model variant $D($Const,NoSib,Strict$)$ is safe.

Proof: Model variant $D($Const,NoSib,Strict$)$ is a stricter version of the Gao-Rexford model, which was proven in [5] to be safe.

D. Model variant $D($Const,Sib,Strict$)$

Theorem 7. Every causation chain in model variant $D($Const,Sib,Strict$)$ is canyon-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 at least one of the two conditions hold:

(a) $y_{i-1} \preceq y_{i} \prec y_{i+1}$

(b) $y_{i-1} \succeq y_{i} \preceq y_{i+1}$

Case (a): $y_{i-1} \preceq y_{i} \prec y_{i+1}$

If case (a) holds, then it can be shown that both $\rho(y_{i}) \succeq y_{i}$ and $\rho_{next}(y_{i}) \succeq y_{i}$. 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 $\rho_{next}(y_{i}) \prec y_{i}$, since this would imply that $y_{i}$ switched from a non-customer path through $y_{i-1}$ to a customer path, since $y_{i} \preceq y_{i-1} \rho(y_{i})$ and $y_{i} \succeq y_{i-1} \rho_{next}(y_{i})$. This would imply RankInc($y_{i}$), causing a contradiction. Thus $\rho(y_{i}) \succeq y_{i}$ and $\rho_{next}(y_{i}) \succeq y_{i}$. If causation condition 2 holds for $y_{i}$, then $y_{i-1} = \rho_{next}(y_{i})$ and RankInc($y_{i}$). It cannot be the case that $\rho(y_{i}) \prec y_{i}$, since this would imply that $y_{i}$ switched from a customer path to a non-customer path through $y_{i-1}$, since $y_{i} \succeq y_{i-1} \rho(y_{i})$ and $y_{i} \preceq y_{i-1} \rho_{next}(y_{i})$. This would imply RankDec($y_{i}$), causing a contradiction. Thus for both cases, $\rho(y_{i}) \succeq y_{i}$ and $\rho_{next}(y_{i}) \succeq y_{i}$.

Thus given the results above, we can prove that $y_{i+1}$ had a realized canyon path. If causation condition 1 holds for $y_{i+1}$, then $\pi(y_{i+1}) = (y_{i+1} \pi(y_{i}))$. Since $y_{i+1} \succeq y_{i}$ and $y_{i} \succeq y_{i-1} \rho(y_{i})$, then $\pi(y_{i+1})$ is a realized canyon path, causing a contradiction. If causation condition 2 holds for $y_{i+1}$, then $\pi_{next}(y_{i+1}) = (y_{i+1} \pi_{next}(y_{i}))$. Since $y_{i+1} \succeq y_{i}$ and $y_{i} \succeq y_{i-1} \rho_{next}(y_{i})$, then $\pi_{next}(y_{i+1})$ is a realized canyon path, causing a total contradiction.

Case (b): $y_{i-1} \succeq y_{i} \preceq y_{i+1}$

If case (b) holds, then it can be shown that both $\rho(y_{i}) \succeq y_{i}$ and $\rho_{next}(y_{i}) \succeq y_{i}$. 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 $\rho_{next}(y_{i}) \prec y_{i}$, since this would imply that $y_{i}$ switched from a non-provider path through $y_{i-1}$ to a non-provider path, since $y_{i} \succeq y_{i-1} \rho(y_{i})$ and $y_{i} \succeq y_{i-1} \rho_{next}(y_{i})$. This would imply RankInc($y_{i}$), causing a contradiction. Thus $\rho(y_{i}) \succeq y_{i}$ and $\rho_{next}(y_{i}) \succeq y_{i}$. If causation condition 2 holds for $y_{i}$, then $y_{i-1} = \rho_{next}(y_{i})$ and RankInc($y_{i}$). It cannot be the case that $\rho(y_{i}) \prec y_{i}$, since this would imply that $y_{i}$ switched from a non-provider path to a provider path through $y_{i-1}$, since $y_{i} \succeq y_{i-1} \rho(y_{i})$ and $y_{i} \prec y_{i-1} \rho_{next}(y_{i})$. This would imply RankDec($y_{i}$), causing a contradiction. Thus for both cases, $\rho(y_{i}) \succeq y_{i}$ and $\rho_{next}(y_{i}) \prec y_{i}$.

Thus given the results above, we can prove that $y_{i+1}$ had a realized canyon path. If causation condition 1 holds for $y_{i+1}$, then $\pi(y_{i+1}) = (y_{i+1} \pi(y_{i}))$. Since $y_{i+1} \succeq y_{i}$ and $y_{i} \succeq y_{i-1} \rho(y_{i})$, then $\pi_{next}(y_{i+1})$ is a realized canyon path, causing a total contradiction. Thus the proof stands.

Theorem 8. Every vertical causation cycle $Y = \langle y_{0} \ldots y_{k} \rangle^{t}$
in DPR model variant $D(\text{Const,Sib,Strict})$ 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 $y_i \not\succeq y_{i-1}$. Note that $i \neq k$, otherwise $y_0 = y_1 = y_2 \ldots = y_{k-1} \neq y_k$, implying $y_0 \neq y_k$, which is a contradiction. Either $y_i \succeq y_{i-1}$ or $y_i \prec y_{i-1}$. It cannot be that $y_{i-1} \succeq y_i$, otherwise by the definition of canyon-free paths $y_0 = y_1 \succeq y_i \succeq y_{i+1} \ldots \succeq y_k$, implying $y_0 \succeq y_k$ 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 canyon, which is a contradiction. So $i = 1$ and $y_0 = y_1$.

Let $j$ be the first index $1 < j < k$ where $y_j \prec y_i$. Note that $j$ has to exist otherwise $y_0 \prec y_1 \preceq y_2 \ldots \preceq y_k$, implying $y_0 \prec y_k$ which is a contradiction. By the definition of canyon-free paths, $y_{j-1} \succeq y_h$ for all $j < h \leq k$. So $y_{j-1} \succeq y_k = y_0$. Therefore $Y$ must be simple, otherwise $\langle y_{j-1} y_j y_1 \rangle$ must be a causation chain of $D$. However since $y_{j-1} \succeq y_0$ and $y_0 \prec y_1$, the causation chain is a canyon, contradicting theorem 7, and thus proving the theorem.

**Theorem 9.** DPR model variant $D(\text{Const,Sib,Strict})$ admits simple and non-simple horizontal causation cycles.

Proof: From the example shown in figure 15. This example is identical to the “bad gadget” described in [4].

**Theorem 10.** DPR model variant $D(\text{Const,Sib,Strict})$ is potentially unsafe.

Proof: From the example shown in figure 15, no stable assignment exists.

**E. Model variant $D(\text{UnConst,NoSib,Strict})$**

**Theorem 11.** Every causation chain in model variant $D(\text{UnConst,NoSib,Strict})$ is ravine-free.

Proof: Assume not. Then there exists a causation chain $Y = \langle y_0 y_1 \ldots y_k \rangle$ and an index $i$ such that $0 < i < k$ and $y_i \prec y_j \not\preceq y_{i+1}$. The same reasoning as case (b) from the proof of theorem 7 can be used:

It can be shown that both $\rho(y_i) \succeq y_i$ and $\rho_{\text{next}}(y_i) \succeq y_i$. 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 $\text{RankDec}(y_i)$. 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 $y_i \prec y_{\text{next}}(y_i) = y_{i-1}$ and $y_i \preceq y_{\text{next}}(y_i)$. This would imply $\text{RankInc}(y_i)$, causing a contradiction. Thus $\rho(y_i) \succeq y_i$ and $\rho_{\text{next}}(y_i) \preceq y_i$. If causation condition 2 holds for $y_i$, then $y_{i-1} = \rho_{\text{next}}(y_i)$ and $\text{RankInc}(y_i)$. 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 $y_i \preceq \rho(y_i)$ and $y_i \prec y_{\text{next}}(y_i)$. This would imply $\text{RankDec}(y_i)$, causing a contradiction. Thus for both cases, $\rho(y_i) \succeq y_i$ and $\rho_{\text{next}}(y_i) \preceq y_i$.

Thus given the results above, we can prove that $y_{i+1}$ had a realized ravine path. If causation condition 1 holds for $y_{i+1}$, then $\pi(y_{i+1}) = \langle y_{i+1} \pi(y_i) \rangle$. Since $y_{i+1} \preceq y_i$ and $y_i \prec \rho(y_i)$, then $\pi(y_{i+1})$ is a realized ravine path, 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 $y_{i+1} \preceq y_i$ and $y_i \prec \rho_{\text{next}}(y_i)$, then $\pi_{\text{next}}(y_{i+1})$ is a realized ravine path, causing a total contradiction. Thus the proof stands.

**Theorem 12.** Every vertical causation cycle in DPR model variant $D(\text{UnConst,NoSib,Strict})$ is simple.

Proof: Assume not. Let vertical causation cycle $Y = \langle y_0 y_1 \ldots y_k \rangle$ be non-simple. The first part of this proof derives the economic relationships of $y_1$ to $y_0$ and $y_{k-1}$ to $y_k$.

Since $Y$ is a vertical causation cycle, there exists a minimal index $i$, $0 < i < k$ such that $y_i \not\preceq y_{i-1}$, as shown in figure 17. Note that $i \neq k$, otherwise $y_0 = y_1 = y_2 \ldots = y_{k-1} \neq y_k$, implying $y_0 \not\preceq y_k$, which is a contradiction. Either $y_{i-1} \prec y_i$ or $y_{i-1} \succeq y_i$. It cannot be that $y_{i-1} \prec y_i$, otherwise by the definition of ravine-free paths $y_0 = y_1 \succeq y_i \succeq y_{i+1} \ldots \succeq y_k$, implying $y_0 \succeq y_k$ which is a contradiction. Therefore $y_{i-1} \prec y_i$. So if $i = 1$, then $y_0 \prec y_1$, otherwise $y_0 = y_1$. So it can be inferred that $y_0 \preceq y_1$.

Let $j$ be the first index $1 < j < k$ where $y_j \prec y_i$. Note that $j$ has to exist otherwise $y_0 = y_1 \preceq y_2 \ldots \preceq y_k$, implying $y_0 \not\preceq y_k$, which is a contradiction. By the definition of ravine-free paths, $y_{j-1} \succeq y_j$ for all $j < h < k$. Therefore $y_{j-1} \succeq y_k = y_0$. Since $Y$ is a non-simple causation cycle, $\langle y_{j-1} y_0 y_1 \rangle$ is a causation cycle of $D$. However this chain...
contradiction.

admits simple and non-simple horizontal causation cycles.

Fig. 17. Vertical Causation cycle in DPR instance $D$(UnConst, NoSib, Strict).

is a ravine since $y_{k-1} \succ y_0$ and $y_0 \preceq y_1$, causing a contradiction.


Proof: From the example shown in figure 16. This example is identical to the "bad gadget" described in [4].

Theorem 14. DPR model variant $D$(UnConst, NoSib, Strict) is potentially unsafe.

Proof: From the example shown in figure 16, no stable assignment exists.

F. Model variant $D$(UnConst, Sib, Strict)

Theorem 15. Every causation chain in model variant $D$(UnConst, Sib, Strict) is ravine-free.

Proof: This can be proven using the exact same reasoning as the proof for theorem 11.

Theorem 16. Every vertical causation cycle in DPR model variant $D$(UnConst, Sib, Strict) is simple.

Proof: This can be proven using the exact same reasoning as the proof for theorem 12.

Theorem 17. DPR model variant $D$(UnConst, NoSib, Strict) admits simple and non-simple horizontal causation cycles.

Proof: From the example shown in figure 16.

Theorem 18. DPR model variant $D$(UnConst, NoSib, Strict) is potentially unsafe.

Proof: From the example shown in figure 16.

APPENDIX B

PROOFS FOR STANDARD VARIANTS OF GAO-REXFORD MODELS

The following appendix section shows the proofs of table II for Gao-Rexford model variants which have standard economic relationships. We use the notation $D$(Stand) to specify that DPR instance $D$ follows standard economic relationships. The following subsection formally defines standard economic relationships. The other two binary parameters are described in appendix A. The abbreviated notation used in Section III-B will be employed for brevity.

A. Standard Economic Relationships

If a DPR instance has standard economic relationships $D$(Stand) = $(\succeq, \succ, G)$, then it contains the economic operator $\succeq$. A tight economic relation is defined by:

$$u \succeq v \text{ iff } u \succeq v \text{ and } u \not\succeq v$$

and no relation is defined by:

$$u \parallel v \text{ iff } u \not\succeq v \text{ and } u \not\equiv v$$

Standard 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 $u \succeq v$.
- If $u$ is a peer to $v$, then $u \parallel v$.

The transitive 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 standard economic operator is that peering relationships are not transitive in the standard variant. Whereas peering is represented by the equivalence relation $=\parallel$ in the strict variant, peering is represented by no relation $\parallel$ in the standard variant. Thus as shown in figure 18, strict economic relationships form equivalence classes with the peering relation $=\parallel$, which are not present in standard economic relationships. This enables a node to be both an indirect peer and provider to another node in the standard variant. However it should be noted that provider-to-customer relationships are transitive in both variants.
Case (c): $y_0 \prec_s y_1$:
Assume $y_0 \prec_s y_1$. Thus $Y$ is vertical. The cases can be further partitioned by $y_{k-1}$’s economic relationship with $y_k$. If $y_{k-1} \prec_s y_k$, then by the definition of valley-free sequences, $y_{i-1} \prec_s y_i$ for all $0 < i \leq k$. Thus $y_0 \prec_s y_k = y_0$, which is a contradiction. Therefore $y_{k-1} \not\prec_s y_k$. If $Y$ is non-simple, then $\langle y_{k-1} y_k y_1 \rangle$ would be a realized causation chain of $D$. Since $y_{k-1} \not\prec_s y_0 = y_k$ and $y_0 \prec_s y_1$, the causation chain is a valley, causing a contradiction. Therefore $Y$ is simple and vertical.

Theorem 21. Model variant $D$(Const,NoSib,Stand) is safe.

Proof: This follows from the results of [5].

C. Model variant $D$(Const,Sib,Stand)

Theorem 22. Every causation chain in model variant $D$(Const,Sib,Stand) is canyon-free.

Proof: The same reasoning used in the proof of theorem 7 can be used.

Theorem 23. Every vertical causation cycle in DPR model variant $D$(Const,Sib,Stand) is simple.

Proof: Let $Y = \langle y_0 y_1 \ldots y_k \rangle^t$ be a vertical causation cycle of model variant $D$(Const,Sib,Stand). Since $Y$ is a vertical causation cycle, there exists a minimal index $i$, $0 < i < k$ such that $y_{i-1} \succ y_i$ or $y_{i-1} \approx_s y_i$.

If $y_{i-1} \prec_s y_i$, since $Y$ is canyon-free, $y_{j-1} \succ y_j$ for all $i \leq j \leq k$. Thus $i > 1$, otherwise $y_{i-1} \succ y_k = y_0$, which is a contradiction. So $y_0 \not\prec y_1$ and $y_i \succ y_k$. So $Y$ must be simple, otherwise the causation chain $\langle y_{k-1} y_0 y_1 \rangle$ would exist. This cannot happen since $y_{k-1} \succ y_0$ and $y_0 \prec_s y_1$, representing a canyon, and thus a contradiction.

If $y_{i-1} \not\prec_s y_i$, then $i = 1$. Otherwise if $i > 1$, then $y_0 \prec_{y_{i-1}} y_{i-1}$, representing a canyon, which is a contradiction. It cannot be that $y_{k-1} \not\prec y_k$, otherwise by the definition of canyon-free sequences, $y_{j-1} \not\prec y_j$ for all $0 < j \leq k$, thus $y_0 \not\prec y_k = y_0$, which is a contradiction. Therefore $y_{k-1} \not\prec y_k$. So $Y$ must be simple, otherwise the causation chain $\langle y_{k-1} y_0 y_1 \rangle$ would exist. This cannot happen since $y_{k-1} \not\prec y_0$ and $y_0 \prec_s y_1$, representing a canyon, and thus a contradiction.

Theorem 24. DPR model variant $D$(Const,Sib,Stand) admits simple and non-simple horizontal causation cycles.

Proof: From the example shown in figure 15.

Theorem 25. Model variant $D$(Const,Sib,Stand) is potentially unsafe.
Path preferences:

Node a: \(\{a \ b \ \text{root}\}\)  
\(\{a \ \text{root}\}\)

Node b: \(\{b \ c \ \text{root}\}\)  
\(\{b \ \text{root}\}\)

Node c: \(\{c \ a \ \text{root}\}\)  
\(\{c \ \text{root}\}\)

Fig. 19. Non simple vertical cycle for DPR model variant: \(D(\text{UnConst,NoSib,Stand})\). Paths not listed in the path preferences are forbidden.

Proof: From the example shown in figure 15, no stable assignment exists.

D. Model variant \(D(\text{UnConst,NoSib,Stand})\)

Theorem 26. Every causation chain in model variant \(D(\text{UnConst,NoSib,Stand})\) is ravine-free.

Proof: This follows from the same reasoning used in the proof of theorem 11.

Theorem 27. DPR model variant \(D(\text{UnConst,NoSib,Stand})\) admits simple and non-simple vertical and horizontal causation cycles.

Proof: An example of a non-simple horizontal cycle can be seen in figure 16. An example of a non-simple vertical cycle can be seen in figure 19.

Theorem 28. Model variant \(D(\text{UnConst,NoSib,Stand})\) is potentially unsafe.

Proof: This follows from the example in figure 19.

E. Model variant \(D(\text{UnConst,Sib,Stand})\)

Theorem 29. Every causation chain in model variant \(D(\text{UnConst,Sib,Stand})\) is ravine-free.

Proof: This follows from the same reasoning used in the proof of theorem 11.

Theorem 30. DPR model variant \(D(\text{UnConst,Sib,Stand})\) admits simple and non-simple vertical and horizontal causation cycles.

Proof: An example of a non-simple horizontal cycle can be seen in figure 16. An example of a non-simple vertical cycle can be seen in figure 19.

Theorem 31. Model variant \(D(\text{UnConst,Sib,Stand})\) is potentially unsafe.

Proof: This follows from the example in figure 19.

Appendix C

Asynchronicity with DPR

This section describes how the DPR model can simulate asynchronicity. We assume that we have a regular DPR instance \(D = (\mathcal{S}, G)\) 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 \(D' = (\mathcal{S}', G')\) 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. 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 \(D'\):

\[u \in V \Rightarrow u \in V'\]

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 20.

\[\begin{cases} 
(u, x_{M}^{uv})^t \in E' \\
(x_{i+1}^{uv}, x_{i}^{uv})^t \in E' \text{ for all } 1 < i < M \\
(x_{2}^{uv}, v)^t \in E' 
\end{cases} \text{ iff } (u, v)^t \in E\]

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

\[(u, x_{i}^{uv})^t \in E' \text{ iff } (u, v)^t \in E \text{ and } L(u, v, t) = i\]

\[(u, v)^t \in E' \text{ iff } (u, v)^t \in E \text{ and } L(u, v, t) = 1\]
An example of a delay of one and three between nodes $u$ and $v$ can be seen in figures 21 and 22.

### B. Path Preferences of Asynchronous DPR Instances

The path preferences of the asynchronous DPR: $D' = (V', E')$ 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 $u \in V'$:

$$P^u_1 \succeq_P^u P^u_2 \text{ iff } \text{RemoveTransit}(P^u_1) \succeq^t \text{RemoveTransit}(P^u_2)$$

Each transit node $x^u_{i+1}$ prefers a path through its source node $u$ than through its transit neighbor toward the source: $x^u_{i+1}$. Paths containing sequences in the opposite direction of the “communication link” (from $x^u_i$ to $x^u_{i-1}$) are forbidden.

### C. 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 23.

In order to remedy this situation, redundant links between the source node $u$ and the transit nodes are established, as shown in figure 24. This enables path consistency during changes of communication delays. Thus the proper transformation of links from synchronous $D$ to asynchronous $D'$ can be represented as:

$$\langle u, x^u_{i+1} \rangle^t \in E' \text{ iff } \langle u, v \rangle^t \in E \text{ and } L(u, v, t) \leq i$$

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

### D. 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 $\langle u, x^u_{3} x^u_{2} v \rangle^t$ in the asynchronous DPR instance $D'$.

### E. Gao-Rexford and Asynchronous DPR Instances

Asynchronous DPR instances can follow the Gao-Rexford conditions described in Section III. Transit nodes have no economic relationships with the other nodes. The domain of the economic operator $\succeq_S$ 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 Remove-
Transit can be employed to characterize customer, peer, and provider paths.

From this construction, the proofs of Sections III, IV and Appendices A, B are unchanged except for the application of the RemoveTransit operator.