A RANDOMIZED SOLUTION TO BGP DIVERGENCE

SELMA YILMAZ IBRAHIM MATTA
Computer Science Department
Boston University
Boston, MA 02215, USA
{selma,matta}@cs.bu.edu

ABSTRACT
The Border Gateway Protocol (BGP) is an interdomain routing protocol that allows each Autonomous System (AS) to define its own routing policies independently and use them to select the best routes. By means of policies, ASes are able to prevent some traffic from accessing their resources, or direct their traffic to a preferred route. However, this flexibility comes at the expense of a possibility of divergence behavior because of mutually conflicting policies. Since BGP is not guaranteed to converge even in the absence of network topology changes, it is not *safe*. In this paper, we propose a randomized approach to providing safety in BGP. The proposed algorithm dynamically detects policy conflicts, and tries to eliminate the conflict by changing the local preference of the paths involved. Both the detection and elimination of policy conflicts are performed locally, *i.e.* by using only local information. Randomization is introduced to prevent synchronous updates of the local preferences of the paths involved in the same conflict.

KEY WORDS
Inter-domain Routing; Border Gateway Protocol (BGP); Convergence Analysis.

1 Introduction

The Internet consists of thousands of ASes that operate independently and exchange routing information to coordinate the delivery of IP traffic. On its path from source to destination, an IP packet traverses routers and links that belong to different ASes. The sequence of ASes traversed by an IP packet is determined by routing policies. ASes use policies to prevent some traffic from accessing their resources, or to direct their traffic to a preferred route. The routing policies are realized through the Border Gateway Protocol (BGP) [7]. BGP allows each AS to select the best routes by applying local policies, and to propagate routing information without revealing local policies to the other ASes. However, Varadhan *et al.* [8] show that a group of ASes may independently define mutually conflicting BGP policies that lead to persistent BGP oscillations. In this state, ASes exchange BGP routing messages indefinitely without ever converging on a set of stable routes. Such divergence behavior may introduce a large amount of instability into the global routing system, which may significantly degrade the performance of the Internet.

The set of routing policies are called *safe* if they can never lead to BGP divergence. There have been recent studies on guaranteeing safety of BGP [3, 2, 6]. Govindan *et al.* [3] propose a static solution which involves keeping policies in a repository called Internet Route Registry and verifying that they do not contain policy conflicts that could lead to protocol divergence. However, Griffin and Willfong [5] show that such kind of verification is computationally very expensive. Also, most ASes do not want to reveal their policies, or keep the information in the registry up-to-date.

To avoid the global coordination required in [3], Gao and Rexford [2] propose another static solution which exploits the commercial relationships between ASes, namely peer-peer, and provider-customer. A pair of ASes have a *provider-customer* relationship if one offers Internet connectivity to the other and have a *peer-peer* relationship if they are providing connectivity among their customers. To ensure the stability of the BGP system, each AS is supposed to follow policy configuration guidelines which suggest preferring routes heard from customers to the routes heard from providers and peers. While this solution guarantees stability, it may unnecessarily disallow the use of many routes. The solution still requires usage of Route Registry database, only this time to keep hierarchical relationships between ASes, which is more public and deducible compared to the entire set of routing policies. Static periodic checks are necessary to verify the validity of a route announcement [1] and to ensure that local-preference values of routes are consistent with the desired relationships.

Griffin and Willfong [6] suggest a dynamic mechanism to detect and suppress BGP oscillations that arise because of policy conflicts. The idea is to extend BGP to carry additional information called *history of updates* with each routing update message. It allows each router to describe the sequence of events that led to the adoption of a path. Since a history of updates with loops is an indication of protocol divergence, divergence can be detected as it happens, and prevented by discarding such updates. However, with this approach, histories can grow very long, which makes processing and sending updates very expensive. Also, history may reveal private information about preferences of ASes over the routes since this information maybe carried implicitly with history.

In this paper, we propose a new dynamic mechanism to detect and suppress BGP oscillations. The motivation behind this work is to eliminate the drawbacks of current approaches mentioned above. This new dynamic algorithm allows us to detect policy conflicts using only local information, and to adjust local preference values through a randomized approach. We eliminate the use of potentially expensive and revealing histories, since the algorithm uses only local information to detect cycles. We also eliminate the need for any off-line phase, and hence the use of Internet Route Registry database either for policies, or relationships between ASes. The new algorithm is also more tolerant to the routes involved in a cycle than current approaches: Instead of immediately invalidating such routes, we reduce their preferences with some probability, and invalidate a route only if it gets involved in a cycle repeatedly, which provides a broader range of routes and hence allows for more flexible route selection.

The rest of the paper is organized as follows. Section 2 reviews related work to provide the necessary background, and the current approaches against which we compare our approach. Our randomized algorithm is described in Section 3. The performance evaluation methodology is presented in Section 4, and results are reported in Section 7. Section 6 concludes the paper.

2. Background and Related Work

To study safety of BGP, Griffin and Willfong [6] propose a simple model called *Stable Paths Problem* (SPP), and suggest that BGP is a distributed algorithm for solving SPP. SPP consists of an undirected graph with a single destination. Each node in the graph has a set of permitted paths and a ranking function to set an order of preference on the paths. Permitted paths are the routes learned from peers, and allowed by the local policy of the node. A solution of an SPP is an assignment of permitted paths to the nodes such that the path assigned to a node is the highest ranked path extending any of the paths chosen at its neighbors. The formal definition of SPP can be summarized as follows: A network is represented as a simple, undirected, connected graph $G = (V, E)$, where $V = \{0, 1, \cdots, n\}$ is the set of nodes connected by edges from E. For any node u, $peers(u) = \{w | \{u, w\} \in E\}$ represents the set of $peers$ for u. It is assumed that there is a single destination, which is node 0, to which all other nodes are trying to find paths. A $path$ P in G is a sequence of nodes $(v_k, v_{k-1}, \cdots, v_1, v_0)$, such that $(v_i, v_{i-1}) \in E$, for all $i, 1 \leq i \leq k$. Each path has implicit directionality associated with it. P's direction is from v_k to v_0. Assuming that $\{u, v\}$ is an edge in E, and v is the first node of path P, then $(u, v)P$ denotes the path that starts at node u, traverses edge (u, v), and follows path P.

An empty path, ϵ, indicates that a router cannot reach the destination. For each $v \in V - \{0\}$, the set $\mathcal{P}^v$ denotes the *permitted paths* from v to the destination. Let $\mathcal{P} = \{\mathcal{P}^v | v \in V - \{0\}\}$ denotes the set of all permitted paths. For each $v \in V - \{0\}$, there is a *ranking function* λ^v, defined over $\mathcal{P}^v$, which defines how node v ranks its permitted paths. If P_1 and $P_2 \in \mathcal{P}^v$ and $\lambda^v(P_1) < \lambda^v(P_2)$, then P_2 is said to be *preferred over* P_1. Let $\Lambda = \{\lambda^v | v \in V - \{0\}\}$ be the set of all ranking functions. An instance of an SPP $S = (G, \mathcal{P}, \Lambda)$, is a graph with the permitted paths and ranking function at each node with the following restrictions imposed on Λ and $\mathcal{P}$: For each $v \in V - \{0\}$

(1) *Empty path is permitted*: $\epsilon \in \mathcal{P}^v$.

(2) *Empty path is the lowest ranked path*: $\lambda^v(\epsilon) = 0$.

(3) *Strictness*: If $\lambda^v(P_1) = \lambda^v(P_2)$, then $P_1 = P_2$ or there is a peer u such that $P_1 = (v\ u)P_1'$ and $P_2 = (v\ u)P_2'$.

(4) *Simplicity*: If path $P \in \mathcal{P}^v$, then P does not have repeated nodes.

For a given node u, let W be a subset of the permitted paths $\mathcal{P}^u$ such that each path in W has a distinct next-hop. The *maximal path* in W, $max(u, W)$, is defined to be the highest ranked path in W. π is defined to be a function called *path assignment*, which maps each node $u \in V$ to a permitted path $\pi(u) \in \mathcal{P}^u$. π defines the path chosen by each node to reach the destination. $choices(u, \pi)$ is a set of paths, defined to be all $P \in \mathcal{P}^u$ such that either $P = (u\ 0)$ and $\{u, 0\} \in E$ or $P = (u\ v)\pi(v)$ for some $\{u, v\} \in E$. The path assignment π is called *stable at node u* if $\pi(u) = max(u, choices(u, \pi))$. The path assignment π is called *stable* if it is stable at every node $u \in V$.

An SPP instance $S = (G, \mathcal{P}, \Lambda)$ is *solvable* if there exists a stable path assignment π for S. Every such assignment is called a *solution* for S and written as $(P_1, P_2, \cdots, P_n)$, where $\pi(u) = P_u$. An instance of SPP may have no solution, or one or more solutions. Examples are GOOD GADGET with a single solution, which is ((1 3 0), (2 0), (3 0), (4 3 0)), and BAD GADGET with no solution as shown in Figure 1. Possible paths for each node are shown next to each one of them in a way that the highest ranked path is placed at the top.

Griffin and Willfong [6] define *Simple Path Vector Protocol (SPVP)* as a distributed algorithm for solving SPP. SPVP is an abstraction of BGP. With this abstraction, messages are simply paths and $rib(u)$ denotes the current path that node u is using to reach the $destination$ and $rib\_in(u \Leftarrow w)$ denotes the table where the most recent path received from each peer $w \in peers(u)$ are kept. Then the set of paths available at node u is $choices(u) = \{(u\ w)P \in \mathcal{P}^u | P = rib\_in(u \Leftarrow w)\}$ and the best path at u is $best(u) = max(u, choices(u))$. The best path is the highest ranked path for node u among the paths received from its peers. The network state of the system is the collection of $rib(u)$, $rib\_in(u \Leftarrow w)$ and the state of all communication links. A network state is *stable* if all communication links are empty. If SPVP converges, the resulting state is the solution of SPP. If SPP has no solution, then SPVP diverges.

Griffin *et al.* [4] introduce the notion of a *dispute digraph*, while developing sufficient conditions that will guarantee safety of an SPVP specification. If a dispute graph does not have any cycle, which is called *dispute cycle*, then the corresponding SPVP specification is *safe* and the corresponding SPP is solvable. Cycles in the dispute graph represent circular dependencies that cannot be satisfied simultaneously. For any instance of SPP $S = (G, \mathcal{P}, \Lambda)$, a directed graph called *dispute digraph*, $\mathcal{DD}(S)$, can be constructed as follows: The nodes of $\mathcal{DD}(S)$ are composed of permitted paths of S and the arcs represent certain relationships between the policies of peers. There are two types of arcs, *transmission arcs* and *dispute arcs*. Assuming nodes u and v are peers, there is a transmission arc $P \cdots > (u, v)P$ if P is permitted at node v, and $(u, v)P$ is permitted at node u. Assuming Q is a permitted path at node v and $(u, v)P$ is a permitted path at node u, there is a dispute arc $Q- > (u, v)P$ if and only if the following are true: (1) $(u, v)P$ is a permitted path at node u, (2) Q and P are permitted paths at node v, (3) Path $(u, v)Q$ is not permitted at node u, or $\lambda^u((u, v)Q) < \lambda^u((u, v)P)$, (4) $\lambda^v(P) \leq \lambda^v(Q)$. Dispute digraphs of GOOD GADGET and BAD GADGET are shown in Figure 1. Since GOOD GADGET is safe, the corresponding dispute digraph is acyclic, whereas the dispute digraph of BAD GADGET has a cycle.

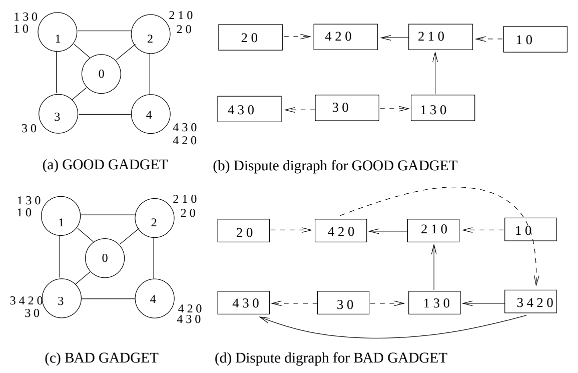

Figure 1. Examples of Stable Paths Problems and the corresponding dispute digraphs.

2.1 Review of the Evaluated Algorithms

2.1.1 Safe Path Vector Protocol: Griffin and Willfong [6] introduce an algorithm for dynamically detecting and eliminating cycles that arise because of policy conflicts. The idea is adding a new attribute called *path history* to the messages. Path histories are dynamically computed sequences of *path change events*. A path change event is a pair *e=(s,P)*, where $s \in \{+, -\}$ is the sign of the event and P is the path. Assuming P_{old} and P_{new} are permitted paths at node u and there has been a transition from $rib(u) = P_{old}$ to $rib(u) = P_{new}$. If we assume that u ranks P_{old} lower than P_{new}, then the corresponding path change event will be $(+, P_{new})$, which means u went up to path P_{new}. If we assume that u ranks P_{new} lower than P_{old}, the corresponding path change event will be $(-, P_{old})$, which means u went down from path P_{old}. Path history is either an empty history or a sequence $e_k\ e_{k-1}\ e_{k-2} \cdots e_1$, where each e_i is a path change event and e_k is the most recent event. A history may have a *cycle* if there exists i, j, with $1 \leq i < j \leq k$, such that $e_i = (s_1, P)$ and $e_j = (s_2, P)$. Figure 2 shows the distributed algorithm which is computing histories dynamically. The algorithm uses a function called *hist(u)* to calculate a new path history for P_{new} whenever the best path at node u changes from P_{old} to P_{new}. The exact procedure is shown in Figure 2. A message m in the algorithm is a pair (P, h) where P is a path and h is a history. For any message $m = (P, h)$, $path(m) = P$ and $history(m) = h$. Each node also uses an additional data structure to keep *bad paths*, $B(u)$. *Bad paths* are the paths that have been invalidated because their adoption led to a cycle in the history. Therefore, the paths in the bad paths set are banned forever even if in the future, they are advertised by peers and permitted by local policies. Definition of $best(u)$ and $choices(u)$ are updated to exclude the paths in the set $B(u)$ as follows:
$choices_B(u) = \{(u\ w)P \in \mathcal{P}^u - B(u))|P \in rib\_in(u \Leftarrow w)\}$ and $best_B(u) = max(u, choices_B(u))$.

```
process spvp[$u$]
  receive $m$ from $w$ $\longrightarrow$
    $rib\_in(u \Leftarrow w) = m$
    if $path(rib(u)) \neq$ best$_B(u)$ then
      $P_{old}$ = path(rib($u$))
      $P_{new}$ = best$_B(u)$
      $h_{new}$ = hist($u$)
      if $h_{new}$ contains a cycle then
        B($u$) = B($u$) $\bigcup \{P_{new}\}$
        $P_{new}$ = best$_B(u)$
        if $P_{new} \neq P_{old}$ then
          $h_{new} = (-,\ P_{old})$
      if $P_{new} \neq P_{old}$ then
        $rib(u) = (P_{new}, h_{new})$
        for each $v \in peers(u)$ do
          send rib($u$) to $v$
```

(a) SPVP process at node u. The program to the right of the $\longrightarrow$ is executed in one atomic step

$hist(u)$	condition
(+, $P_{new})h$	if $\lambda^u(P_{old}) < \lambda^u(P_{new})$, $P_{new} = (u\ w)P_2$, and $rib\_in(u \Leftarrow w) = (P_2, h)$,
(-, $P_{old})h$	if $\lambda^u(P_{new}) < \lambda^u(P_{old})$, $P_{old} = (u\ w)P_1$, and $rib\_in(u \Leftarrow w) = (P_2, h)$,
$(s, Q)h$	if $\lambda^u(P_{old}) = \lambda^u(P_{new})$, $P_{old} = (u\ w)P_1$, and $P_{new} = (u\ w)P_2$, and $rib\_in(u \Leftarrow w) = (P_2, h)$, $h = e_k\ e_{k-1} \ldots e_1$, s=sign($e_k$), and $Q = \begin{cases} P_{old} & \text{if } s = - \\ P_{new} & \text{if } s = + \end{cases}$

(b) Auxiliary function $hist(u)$ for SPVP algorithm. The most recent path adopted by w is P_2 and the associated history h explains why w did so. The path that w has just abondoned is P_1.

Figure 2. SPVP process at node u and function $hist(u)$.

2.1.2 Stable Internet Routing without Global Coordination (Gao&Rexford Algorithm): Gao and Rexford [2] propose a set of guidelines guaranteeing safety of BGP when they are followed. The approach exploits commercial relationships between autonomous systems in the Internet: The neighboring ASes have either a *customer-provider* or *peer-peer* relationship. Considering a node u, the set $neighbors(u)$ is partitioned into the following sets: $customers(u)$, $peers(u)$, and $providers(u)$. Paths are classified depending on the relationship between the first two nodes of the path. A path $(u\ v)P$ is a customer path if $v \in customers(u)$, a peer path if $v \in peers(u)$, or a provider path if $v \in providers(u)$. Gao *et al.* [1] propose that a stable paths problem with the following properties is safe: (1) *Acyclic provider-customer digraph*: The directed graph induced by the customer-provider relationships is acyclic; (2) *No valley*: A path between two nodes should not traverse an intermediate node that is lower in the hierarchy; (3) *No steps*: A peer-peer edge $\{u, v\}$ is a step in path $P_1(u\ v)P_2$ if either the last edge of P_1 is a peer-peer or provider-customer edge, or the first edge in P_2 is a customer-provider edge; (4) *Customer paths are preferable to peer and provider paths*. The idea of this algorithm is to use a database called *route registry* to store the relationships between each AS pair for each destination, and then to check if the aforementioned properties are satisfied. The route registry can identify the sequence of ASes invading these properties and force them to use a restrictive policy.

3 Randomized Algorithm

We are proposing an alternative algorithm for dynamic loop detection where only *local histories* are used. If there is a policy conflict, each node involved in this conflict will observe a *route flap*, and therefore will be able to locally detect which one of its paths is involved in a cycle. To break a cycle, it maybe sufficient to invalidate only one of the paths involved or drop the local preference of only one of such paths. However, since the proposed algorithm is distributed and based only on local information, there maybe synchronous invalidation or reduction of the preferences of the paths involved in the conflict. To prevent unnecessary path invalidation, or rank change, we suggest a randomized approach: Upon detection of an involvement in a cycle, the local preference of the path P_i is reduced with a probability inversely proportional to its preference rank, i.e. the probability decreases as $2^{-rank(P_i)}$. Figure 3 shows the exact algorithm. $rank(Path)$ is the index of $Path$ at node u in the order of decreasing local preference value.

With our randomized approach, because of the probabilistic drop of local preferences, a cycle may not be eliminated even though it is observed several times. This happens, for example, if the local preference of none of the paths involved in a cycle has been lowered. Another possibility is that each one of the paths in the cycle is the least preferred path in the corresponding node. That is why the lowering local preference of these paths would not change the relative rank of paths, hence the cycle would remain. To be able to deal with such persistent cycles and/or paths that get involved in many different conflicts, our algorithm makes use of counters to keep track of the number of times each path gets involved in a cycle. $times(Path, u)$ is the value of the counter showing how many times $Path$ has been adopted by node u. When a path is adopted and later abandoned as many times as some predetermined value, $max\_threshold$, it is invalidated and put in the set of *bad paths*, B. The paths in B are excluded from further consideration in the best path selection process, even if they are advertised by peers and permitted by local policies. On the other hand, $min\_threshold$ specifies how many route flaps later a node decides that there is a policy conflict. If a counter of a path exceeds this value, then a probabilistic dropping of its rank is started.

4 Evaluation Method

For a given SPP $S = (G, \mathcal{P}, \Lambda)$, we would like to see how efficient the algorithms are at removing all possible policy conflicts. To be able to do this, we run the algorithms repeatedly until all possible conflicts are resolved and the system is safe. Safety is tested by constructing the dispute digraph of S with the current set of permissible paths, $\mathcal{P}$, and checking it for cycles. At each run, to eliminate a dispute cycle, conflicting paths are either invalidated or their ranks have been updated depending on the algorithm. The pseudo-code of our evaluation method is shown in Figure 4. The input is just a graph, G, instead of an SPP specification. We construct SPP by finding the set of possible paths at each node of G, and assigning local preference values as shown. Therefore, at step 10, we have an SPP specification, with graph G, set of all permitted paths being the set of all possible paths and the ranking function is as shown in lines 3-9. After step 10, since each algorithm handles conflicts in a different way, the exact details of the evaluation method is different for each algorithm as discussed below.

Gao&Rexford algorithm has no run-time component. Therefore, we only construct the corresponding SPP, S. The $\mathcal{P}$ component of S is the set of all possible paths at this point. Each path in $\mathcal{P}$ is checked for guidelines reviewed in Section 2.1.2. Paths involving steps or valleys are invalidated, as well as customer-provider paths. Since following these guidelines guarantees safety, the resulting SPP with updated set of permitted paths is safe.

For SPVP, each independent conflict that is observed for the current state of SPP S is taken care of at each iteration (lines 10-14) until the system is safe. A cycle which does not contain any smaller cycles is called *independent*. The idea behind finding independent cycles and eliminating them in a single run is an attempt to model SPVP better in this static context. SPVP is a distributed, dynamic algorithm. Path histories are carried by path updates, which provide the main mechanism of detection and elimination of conflicts. Since we are not using the dynamic algorithm for evaluation, we try to model such dynamic behavior as closely as possible. In a dynamic environment, while the actual algorithm is running, the shorter cycles would be detected earlier than the longer cycles, just because the nodes whose paths are involved in shorter cycles are located closer to each other. Since SPVP is distributed, in a dynamic environment, many small independent cycles can be detected and taken care of simultaneously. Therefore, in our evaluation method, we break all independent cycles in one iteration. However, a question remains: which path(s) involved in a particular cycle should be invalidated and labeled as bad path in our static evaluation of SPVP. In a dynamic environment, the first node detecting the cycle would give up its path, and update the corresponding history accordingly. Therefore, none of the other nodes whose paths involved in this particular cycle would attempt to break this cycle again. Which node will be the first one to notice the cycle depends on the order of messages propagated. Therefore, in our static evaluation, to break a particular cycle, we *arbitrarily* choose a path involved in the cycle and exclude it from the set of permitted paths.

For our randomized approach, the evaluation is very similar to SPVP. However, with our randomized algorithm, a particular cycle would be observed in the form of route flaps. All the nodes whose paths are involved would try to break the cycle simultaneously without any way of knowing about the other nodes. However, since the algorithm is randomized, some nodes will end up lowering the local preference of their path involved in the cycle and some nodes will do nothing. The best case of our randomized algorithm happens when only one node lowers the local preference of its path involved in the cycle and doing so results in breaking the cycle. The worst case of our algorithm happens when none of the nodes lower the local preference of their paths involved in the cycle and this behavior repeats $max\_threshold$ times. As a result, each node is forced to add its path to B. We have a set of results for both of these cases. For the best case, we *arbitrarily* choose a path in the cycle, and reduce its local preference, without using any probabilities. For the worst case, we do not lower local preferences for any path involved in the cycle. Therefore, after $max\_threshold$ times seeing the same cycle, all the paths involved in the cycle are added to the set of bad paths. To obtain the expected performance of our algorithm in practice, we obtain another set of results, by probabilistically reducing local preference of each path involved in the cycle. The set of results showing the best and worst cases of our randomized algorithm are shown in Figure 5 and the set of results showing the expected behavior are shown in Figures 6, and 7.

```
process randomized[$u$]
    receive $m$ from $w$ $\longrightarrow$
        rib_in$(u \Leftarrow w) = m$
        if rib($u$)$\neq$ best$_B$($u$) then
            $P_{old}$=rib($u$)
            $P_{new}$ = best$_B$($u$)
            if ($P_{new} \neq P_{old}$) and ($P_{new} \neq \epsilon$)    then
                times($P_{new}$,$u$)++
            if times($P_{new}, u$) $\geq max\_threshold$    then
                B($u$) = B($u$) $\bigcup \{P_{new}\}$
                $P_{new}$ = best$_B$($u$)
            else if times($P_{new}, u$) $\geq min\_threshold$    then
                if $P_{new}$ is not least preferred path    then
                  localpref($P_{new}$)=localpref($P$)
                  with probability=$1/2^{rank(P_{new})}$,
                  where rank($P$)=rank($P_{new}$)-1
                else
                  localpref($P_{new}$)=localpref($P_{new}$)/2
                  with probability=$1/2^{rank(P_{new})}$
        if $P_{new} \neq P_{old}$    then
            rib($u$) =$P_{new}$
            for each $v \in peers(u)$    do
              send rib($u$)    to $v$

Note: The code to the right of the $\longrightarrow$
    is assumed to be executed in one atomic step .
```

Figure 3. Randomized Algorithm at node u.

```
line  process evaluator(graph $G$)
  1     construct SPP $S = (G, \mathcal{P}, \Lambda)$
  2         find all paths from each node to destination
            // set local preferences of the paths
  3         for each path $(u, v)P$
  4             if $v \in customers(u)$ then
  5                 set local preference of the path to 500
  6             if $v \in peers(u)$ then
  7                 set local preference of the path to 400
  8             if $v \in providers(u)$ then
  9                 set local preference of the path to 300
 10     while $S = (G, \mathcal{P}, \Lambda)$ is not safe //there is a cycle in $DD(S)$
 11         fi nd all cycles in $DD(S)$
 12         eliminate cycles that are not independent
 13         for each resulting independent cycle
 14             break the cycle //update set of all permitted paths, $\mathcal{P}$
```

Figure 4. The pseudo-code used to evaluate the algorithms.

5 Performance Metrics and Results

We used *dispute wheels* for evaluation. A dispute wheel of size n is a graph with n nodes, and one destination, node 0. Each node has 2 paths that are either the direct path or the path through the clockwise neighbor, where the latter is more preferred. The more formal definition of dispute wheels can be found in [4]. Griffin *et al.* [4] show that for every dispute wheel, there is a cycle in the corresponding dispute digraph, which in turn suggests policy conflicts that cannot be satisfied simultaneously. That is why when we use a graph that contains a dispute wheel, we can be sure that there is a policy conflict. The results in Figure 5 are obtained using a dispute wheel of size 250. To be able to deal with the huge number of possible paths, at the beginning of the evaluation, the possible paths for each node are restricted to the direct path and the path through the clockwise neighbor. For Figures 6, and 7, we have used smaller graphs of size 5, 10, 15, 20, and 25. Although with these smaller graphs, we were able to include extra paths in addition to the direct path or the path through the clockwise neighbor in the set of possible paths, we still needed to exclude some paths (customer-provider paths in our case) at the beginning of the evaluation just to keep the set more manageable. For our randomized algorithm, $max\_threshold$ and $min\_threshold$ are set to 6 and 2, respectively.

Number of tries is a measure of how many times we go through the while loop in line 10 of Figure 4. The metric is used to measure how quickly the system reaches safety. Figures 5 and 6 show the results. This metric is not meaningful for the Gao&Rexford algorithm, which does not have a run-time component. As expected, our randomized algorithm takes more tries to break cycles than SPVP, since SPVP eliminates conflicts by immediately excluding one of the paths involved. This is a sure way of eliminating both the current conflict and future ones that might involve this path. Our randomized approach tries to eliminate conflicts by reducing the local preferences probabilistically. However, sometimes lowering the preference of a path may not be enough to break a cycle: The preference might need to be further reduced. It is also possible that after breaking a particular conflict, the same path may get involved in another conflict because of its updated preference rank. It is also the case that lowering the preference of a path may take a few tries just because of the probabilistic nature of our algorithm. Therefore, it is obvious that resolving conflicts with our randomized algorithm may take longer than SPVP. The worst-case value of our randomized approach in Figure 5 is 6 because we set $max\_threshold$ to 6. The worst-case behavior can be improved for this metric by choosing a smaller value for $max\_threshold$. However, this may lead to more path elimination and hence badly affect reachability, which is the basic trade-off in this context. As the size of the network grows, the number of tries for our randomized algorithm approaches that of SPVP.

The percentage of the paths that are excluded from the set of possible paths is used to see if an algorithm eliminates too many paths and hence puts strain on routing reachability. Both Figures 5 and 7(a) show that the Gao&Rexford algorithm has the worst performance and may give up 96% of possible paths. Figure 5 shows that although the worst-case performance of our randomized algorithm is as bad as the Gao&Rexford algorithm, the best-case performance is as good as SPVP. Figure 7(a) shows that even though the expected performance of our randomized approach is worse than SPVP, it is still much better than the Gao&Rexford algorithm and excludes approximately 60% less paths. With SPVP, the number of paths that are eliminated equals the number of cycles observed, since SPVP breaks each cycle by putting exactly one path away. The number of cycles increases with increasing size of the graph, but at a slower rate. This is the main reason why in Figure 7(a) the metric value of SPVP decreases as the number of nodes increases. For our randomized approach, the number of cycles observed and dealt with in a single iteration is much higher than SPVP, since the randomized approach does not put away paths as soon as a cycle is observed, the cycle may not be broken or the path whose local preference value is updated may get involved in a different cycle in the next iteration. Another observation is that when the paths in a cycle are all the least preferred paths, then by lowering the local preference of the least preferred paths, the randomized approach does not change the ranks of the paths and the cycle will remain for up to $max\_threshold$ iterations. At this point all of the paths involved in the cycle will be eliminated and put in B. As a result, we observe that the randomized algorithm has a higher volume of path exclusions than SPVP.

Rearranging the ranks of the permitted paths is the basic mechanism for the randomized approach to resolve a conflict. To look deeper into the behavior of our randomized approach, we try to answer the question whether the algorithm causes too many nodes to rearrange the ranks of their paths and so giving up their preferences. For this purpose, we use the metric called *percentage of loss of preferences*, which is defined as $(total_1 - (total_2 + total_3))/total_1$, where $total_1 = \sum_{p \in \mathcal{P}_1} localpreference(p)$, $total_2 = \sum_{p \in \mathcal{P}_2} localpreference(p)$, and $total_3 = \sum_{p \in \mathcal{B}} originallocalpreference(p))$, denote the total value of local preferences for the sets $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{B}$, respectively; $\mathcal{P}_1$ is the set of permitted paths for the SPP to begin with, $\mathcal{P}_2$ is the set of permitted paths for the resulting SPP, and $\mathcal{B}$ is the set of all paths eliminated by the algorithm. For the paths that are in $\mathcal{B}$, the whole value of the original local preferences is counted as loss. The results are shown in Figures 5 and 7(b). Since we have already seen that the Gao&Rexford algorithm eliminates most of the paths, it is not surprising to see that it has the worst performance for this metric too. The same is true for the randomized algorithm at its worst-case behavior. Figure 7(b) shows that the expected performance of the randomized approach is not always as good as SPVP, but still about 35% better than the Gao&Rexford algorithm.

6. Conclusion and Future Directions

Our proposed randomized algorithm is designed to realize safety of BGP, while eliminating the drawbacks of current approaches. It is a dynamic algorithm which eliminates any kind of static checking, or route registry database as in the Gao&Rexford algorithm. It does also eliminate the need to carry potentially long and revealing histories as in SPVP. Instead, the cycles are detected locally, i.e. by using only local information. Our randomized algorithm attempts to resolve policy conflicts by adjusting the ranks of a few paths. Thus ASes would not need to lose their paths, and possibly end up not being able to reach the destination. The performance of our randomized algorithm is expected to be as close as SPVP in practice, while its worst-case performance is no worse than the Gao&Rexford algorithm.

We are currently evaluating the algorithms using detailed packet-level simulations using the SSF simulator, *www.ssfnet.org*. We are improving our randomized algorithm by restoring the local preferences of the paths whose local preferences were dropped as soon as stability is reached. This improvement will help us eliminate some unnecessary rank changes. We are also aware of the fact that not all route flaps are caused by policy conflicts. Therefore, we are further improving our algorithm by differentiating between transient and permanent route flaps. We are also working on mathematically analyzing our algorithm.

Metrics	Gao&Rexford	SPVP	Randomized best case	Randomized worst case
Number of Tries		1 ∓ 0.0	2 ∓ 0.0	6
%of Excluded Paths	50	$0.2 \mp 1.24E{-}17$	$0.2 \mp 1.24E{-}17$	50
% of Loss of Preference Value	55.55	$0.22 \mp 3.73E{-}17$	0.44 ∓ 0.05	55.55

Figure 5. Results for 250-node input graph.

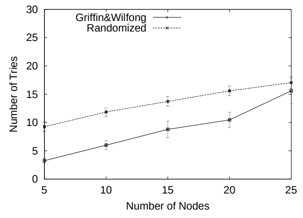

Figure 6. Number of Tries.

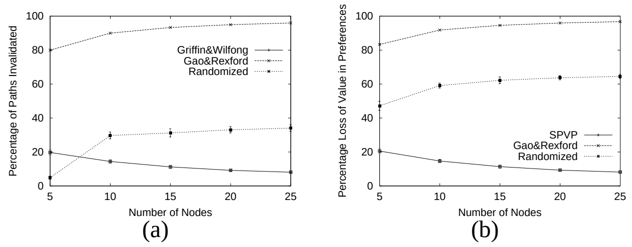

Figure 7. (a) Percentage of the paths that are invalidated (excluded) to break all possible cycles (b) Percentage of Loss of Preferences

Acknowledgment

This work was supported in part by NSF grants ANI-0095988, ANI-9986397, EIA-0202067, and ITR ANI-0205294.

References

[1] L. Gao, T. Griffin, and J. Rexford. “Inherently Safe Backup Routing with BGP ”. In *Proc. IEEE INFOCOM 2001*, April 2001.

[2] L. Gao and J. Rexford. “Stable Internet Routing without Global Coordination ”. In *Proc. ACM SIGMETRICS*, June 2000.

[3] R. Govindan, C. Alaettinoglu, G. Eddy, D. Kessens, S. Kumar, and W. Lee. “An Architecture for Stable, Analyzable Internet Routing ”. *IEEE Network*, 13(1):29-35, 1999.

[4] T. Griffin, F. Shepherd, and G. Willfong. “Policy Disputes in Path-Vector Protocols”. In *Proc. IEEE ICNP 1999*, 1999.

[5] T. Griffin and G. Willfong. “An Analysis of BGP Convergence Properties”. In *Proc. ACM SIGCOMM*, September 1999.

[6] T. Griffin and G. Willfong. “A Safe Path Vector Protocol”. In *Proc. IEEE INFOCOM 2000*, March 2000.

[7] Y. Rekhter and T. Li. “A Border Gateway Protocol”. In *RFC 1771*, March 1995.

[8] K. Varadhan, R. Govindan, and D. Estrin. “Persistent Route Oscillations in Inter-Domain Routing ”. *Computer Networks*, 32:1-16, 2000.