An Adaptive Management Approach to Resolving Policy Conflicts

SELMA YILMAZ IBRAHIM MATTA

Computer Science Department
Boston University
Boston, MA 02215, USA

{selma,matta}@cs.bu.edu

Technical Report BUCS-TR 2006-008

Abstract

The Border Gateway Protocol (BGP) is the current inter-domain routing protocol used to exchange reachability information between Autonomous Systems (ASes) in the Internet. BGP supports policy-based routing which allows each AS to independently define a set of local policies on which routes it accepts and advertises from/to other networks, as well as on which route it prefers when more than one route becomes available. However, independently chosen local policies may cause global conflicts, which result in protocol divergence. In this paper, we propose a new algorithm, called Adaptive Policy Management Scheme (APMS), to resolve policy conflicts in a distributed manner. Akin to distributed feedback control systems, each AS independently classifies the state of the network as either conflict-free or potentially conflicting by observing its local history only (namely, route flaps). Based on the degree of measured conflicts, each AS dynamically adjusts its own path preferences—increasing its preference for observably stable paths over flapping paths. APMS also includes a mechanism to distinguish route flaps due to topology changes, so as not to confuse them with those due to policy conflicts. A correctness and convergence analysis of APMS based on the sub-stability property of chosen paths is presented. Implementation in the SSF network simulator is performed, and simulation results for different performance metrics are presented. The metrics capture the dynamic performance (in terms of instantaneous throughput, delay, etc.) of APMS and other competing solutions, thus exposing the often neglected aspects of performance.

Key Words: Inter-domain Routing; Border Gateway Protocol (BGP); Feedback Control; Convergence Analysis; Simulation.

I. INTRODUCTION

The Border Gateway Protocol (BGP) plays a major role in the performance of the Internet, and is known to have properties that are far from ideal. BGP allows policy-based routing; each AS independently defines a set of local policies on which routes to accept and advertise from/to other networks, as well as on which route it prefers when more than one route becomes available. However, independently defined local policies may lead to policy conflicts. Policy conflicts occur when neighboring ASes have opposite interests over routes. For example, assume AS u and AS v are neighbors, and AS v has two permitted paths $p1$ and $p2$, where $p1$ is preferred over $p2$. If extensions of $p1$ and $p2$, *i.e.* $(u, v)p1$ and $(u, v)p2$, are permitted at AS u, and $(u, v)p2$ is preferred over $(u, v)p1$, then there is a policy conflict. When AS v improves its best path from $p2$ to $p1$, AS u will be forced to give up its more preferred path for the less preferred one. Any policy conflict can be resolved by changing the preference of the ASes over their paths, *i.e.* local policies.

Although not all policy conflicts are harmful, a group of ASes may define conflicting policies that cannot be satisfied simultaneously, causing BGP to *diverge*. Assume AS u, v, and z form such group. The scenario of divergence may take place as follows: When AS u improves its best path, it forces AS v to give up its best path for a less preferred path, which in turn gives AS z an opportunity to improve its best path, which forces AS u to give up its best path for a less preferred path, and so on. Each AS in such conflict repeatedly selects the same sequence of routes, never converging on any one set of routes. Therefore, route oscillations due to policy conflicts are *persistent*, and requires some kind of intervention to stop.

As the commercial infrastructure of the Internet continues to grow, so does the potential for developing persistent route oscillation because of the growth of policies both in size and complexity [1], [2].

Several studies [3], [2], [4] have examined the dynamic behavior of inter-domain routing and highlighted the negative impacts of unstable routes. Instabilities taking place across ASes may negatively impact end-to-end network performance and efficiency of the Internet. A network that has not yet reached convergence may drop packets or deliver packets out of order. Routers may experience severe CPU load and memory problems: Because of repeated advertising and withdrawing of routes, routers need to rerun the BGP decision process to select the best paths, and update routing and forwarding tables. Frequent changes in the routes that are advertised by the other domains also make traffic engineering through an AS very difficult. BGP is crucial for a healthy and efficient global routing, and it is imperative to guarantee convergence of BGP independent of the locally selected policies.

Contribution of This Paper: There have been a number of studies (reviewed in Section II) on guaranteeing safety, *i.e.* convergence, of BGP independent of the locally selected policies [5], [6], [7], [8], [9], [10], [11], [12]. In our previous work [13], we introduced the idea of dynamically detecting and suppressing BGP oscillations through probabilistic change of path ranks (preferences). The algorithm is designed to detect policy conflicts by using local histories only. This paper extends and completes our preliminary idea [13] in many ways: (1) we augment the algorithm of path rank change so that an AS might choose a less preferred but *observably stable* path over a more preferred but oscillating path, thus it becomes

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

natural for an AS to implicitly assign a higher cost (and hence less preference value) to oscillating (flapping) paths; (2) with new additions, the algorithm enables the nodes to dynamically adapt to any state of the network. After the system stabilizes, we let the nodes attempt to restore some of the local preference values of their paths which they have modified so as to keep the overall path rank change minimal [1]; (3) a new mechanism is added to distinguish route flaps due to topology changes, so as not to confuse them with those due to policy conflicts; (4) BGP extensions of the proposed algorithm are specified; (5) a correctness and convergence analysis of the proposed algorithm based on the sub-stability property of chosen paths is presented; (6) implementation of the proposed algorithm in the SSFNet simulator [15] is performed, and simulation results for different performance metrics are presented. The metrics capture the dynamic performance (in terms of instantaneous throughput, delay, routing load, etc.) of our algorithm as well as other competing solutions, thus exposing the often neglected aspects of performance. Although our exposition is BGP-specific, the problem of inconsistent policies at independent distributed entities is more general.

The paper is organized as follows: Section II reviews background and related work. Section III describes our algorithm in detail, and Section IV presents convergence and correctness analysis. Simulation results and conclusion are presented in Section V and Section VI, respectively.

II. Background and Related Work

A. Border Gateway Protocol Abstraction

We use the abstraction of BGP proposed by Griffin $et\ al.$ [7], which is called *Safe Path Vector Protocol (SPVP)*. SPVP is a distributed algorithm for solving the so-called *Stable Paths Problem* (SPP). This model abstracts away low_level details of BGP and makes it easier to reason about convergence related issues.

Informally, SPP consists of an undirected graph with a single destination. Each node in the graph has a set of *permitted paths* to the destination, which are the routes learned from peers, and allowed by the local policy of the node. Each node also has a *ranking function* to set an order of preference on the paths, such that more preferable paths have higher values assigned to them. A solution of an SPP is an assignment of permitted paths to the nodes that is consistent with the path chosen by its next-hop neighbor: Node u may choose the path $P =< u, v, w, \ldots, destination >$ only if the current path at node v is $< v, w, \ldots, destination >$ and path P is the current best path of node u.

The formal definition of SPP is 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. Nodes represent BGP routers and edges represent BGP sessions. For a node u, its set of $peers$ is $peers(u) = \{w|\{u, w\} \in E\}$. Node 0 is the destination 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 $i, 1 \leq i \leq k$.

An empty path, ϵ, indicates that a router cannot reach the destination. Nonempty paths $P =< v_1, v_2, \cdots, v_k >$ and $Q =< w_1, w_2, \cdots, w_m >$ can be *concatenated* as follows $PQ =< v_1, v_2, \cdots, v_k, w_2, \cdots, w_m >$ if $v_k = w_1$. For every path P, concatenation with the empty path returns the path itself: $P\epsilon = \epsilon P = P$.

For every $v \in V - \{0\}$, 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 every $v \in V - \{0\}$, there is a ranking function $\lambda^v : \mathcal{P}^v \rightarrow \mathbf{N}$. $\lambda^v(P)$ denotes the degree of preference that node v gives to the path $P \in \mathcal{P}^v$. More preferable paths have higher values of λ^v. Let $\Lambda = \{\lambda^v | v \in V - \{0\}\}$ be the set of all ranking functions.

An instance of a *Stable Paths Problem (SPP)* $S = (G, \mathcal{P}, \Lambda)$, is a graph with the permitted paths and ranking function at each node if the following conditions are satisfied for every $v \in V - \{0\}$:

(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 $P_1 = (v, u)P_1'$ and $P_2 = (v, u)P_2'$ for some node u.

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

Given a node u, and $W \subseteq \mathcal{P}^u$ with distinct next-hops, the *maximal path* in W, $max(u, W)$, is defined to be the highest ranked path in W. A *path assignment* is a function π that 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. Given a path assignment π and a node u, the set of permitted paths that are one-hop extension of paths through neighbors is defined as

$$choices(u, \pi) = \{(u, v)\pi(v) | \{u, v\} \in E\} \bigcap \mathcal{P}^u.$$

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.

SPVP is an abstraction of BGP. Every node runs a copy of the SPVP process. With this abstraction, messages are simply paths. Each node maintains two data structures: $rib(u)$ is the current path that node u is using to reach the $destination$, and $rib\_in(u \Leftarrow w)$ denotes the path that has been most recently advertised by peer w and processed at node u. The set of paths available at node u is updated as

$$choices(u) = \{(u, w)rib\_in(u \Leftarrow w) | w \in peers(u)\} \bigcap \mathcal{P}^u$$

and the best path at u is

$$best(u) = max(u, choices(u)) \text{ and } rib(u) = best(u)$$

As long as node u receives advertisements from its peers, *best(u)* is recomputed with the most recent *choices(u)*, and stored in $rib(u)$. Just as it is the case with BGP, when u changes its current path, it notifies its current peers about the

[1] Akin to distributed recovery mechanisms, e.g. congestion avoidance of TCP [14].

change. This may cause the peers to send advertisements to their peers. The network reaches a *stable state* when there is no node which would change its current path to the destination. If such a state is reached, then the resulting state is the *solution* of the Stable Paths Problem (SPP). If SPP has no solution, then SPVP diverges.

Griffin $et\ al.$ [16] present the structure called *dispute wheel* for the purpose of checking the existence of a solution, and show that the lack of a dispute wheel is a sufficient condition which guarantees that SPP has a unique solution. A dispute wheel of size k is a structure that consists of nodes, $u_1, u_2, \ldots, u_k$, and the set of paths $Q_1, Q_2, \ldots, Q_k$, and $R_1, R_2, \ldots, R_k$. For each $1 \leq i \leq k$, the following conditions are true: (1) R_i is a path from u_i to u_{i+1} ($u_1 = u_{k+1}$); (2) Q_i is a permitted path at u_i ; (3) R_iQ_{i+1} is a permitted path at u_i ($Q_1 = Q_{k+1}$); (4) Q_i is less preferred than R_iQ_{i+1} at node u_i. The dispute wheel of size k is shown in Figure 1. Q_is are called *spokes* of the dispute wheel and each spoke must be a simple (loop-free) path, *i.e.* no repeated nodes. None of the paths Q_i, R_i or Q_{i+1} can include node u_i. The paths R_is are called the *rims* of the wheel, and each rim is also a simple (loop-free) path. The nodes at the ends of the paths R_is are called *active nodes*. The active nodes are the nodes at which route preferences cause the dispute wheel.

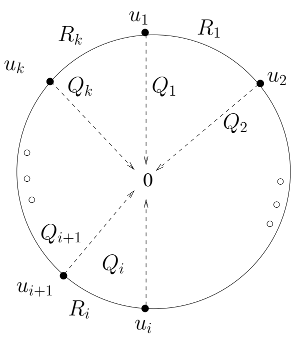

Fig. 1. Dispute wheel of size k

Note that the presence of a dispute wheel does not imply that the system will diverge. However, if the system diverges, there exists a dispute wheel, and the oscillation must be either because of multiple solutions or lack of a solution as we demonstrate later.

B. Related Work

The possibility of BGP divergence due to policy conflicts is first shown by Varadhan $et\ al.$ [1]. Since then, many studies proposed approaches to guarantee the safety, *i.e.* convergence, of BGP independent of the locally selected policies [5], [6], [7], [8], [9], [13], [10], [11], [12]. These approaches can be broadly classified into static and dynamic solutions. Static solutions require analyzing routing policies to verify that they are conflict free, and cannot lead to protocol divergence, whereas dynamic solutions require some mechanism to detect and resolve policy conflicts that are leading to divergence at run time.

Static solutions: Govindan $et\ al.$ [5] 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 $et\ al.$ [17] show that such kind of verification is computationally very expensive, and hard to achieve due to the private nature of the policies.

To avoid global coordination required in [5], Gao $et\ al.$ [6], [18] propose another static solution which restricts the routing policies to the hierarchical structure that arises from commercial relationships between ASes, which may either be *provider-customer* or *peer-peer* relationship. Gao $et\ al.$ give policy configuration guidelines in which each AS prefers routes heard from customers to the routes heard from providers and peers. Then the system is guaranteed to converge. This solution requires a database to keep relationships between ASes. Static periodic checks are required and performed by a global authority to verify conformance with these guidelines. Gao $et\ al.$ algorithm may lead to unnecessary disabling of many routes from the start to guarantee the stability of the system, which restricts the flexibility in the choice of routing policies. Feamster $et\ al.$ [19] argue that there maybe legitimate reasons to deviate from the guidelines proposed in [6].

Other static solutions [12], [11], [10] suggest different constraints that also prevent policy based oscillations in advance.

Dynamic Solutions: Although *route flap damping* [20] can suppress temporary instabilities very well, it cannot detect or eliminate policy conflicts leading to persistent oscillation. Therefore, when there is a policy conflict leading to persistent oscillation, using route flap damping only makes the oscillations run in slow motion.

Griffin and Wilfong [7] suggest an algorithm called SPVP, which extends BGP to carry additional information called *history* with each routing update message. $History$ allows each router to describe the sequence of events that led to the adoption of a path. Since a $history$ containing loops (cycles) is an indication of a potential protocol divergence, divergence can be detected dynamically, and prevented by discarding (eliminating) potentially problematic paths. A cycle in the *history* is a necessary but not sufficient condition for divergence. Therefore, there maybe false positives. Carrying *history* with each update message creates communication overhead, and may also reveal private information about the preferences of ASes over the routes. SPVP cannot differentiate between persistent and transient oscillations, and suggests observing the same loop for a number of times in the *history* before concluding that the loop is due to a potential policy conflict and persistent. However, this approach increases communication overhead even more due to even longer update messages. SPVP does not attempt to limit the number of eliminated paths, and does not enable the usage of an eliminated path later even if the dynamics of the system change, and the system becomes conflict-free.

Recently, Cobb $et\ al.$ [8], [9] propose two dynamic algorithms. The first work [8] proposes a mechanism to enforce monotonic path orderings. With this algorithm, convergence is guaranteed by preventing the selection of a path with higher order (preference) in one node, if doing so would cause a conflict with other nodes along the *routing tree*, *i.e.* a node would be forced to use another less preferred path. Preventing the violation of monotonic path ordering property is realized via diffusing computation along the routing tree whenever a node wants to update its path.

In their second work, Cobb and Musunuri [9] associate an integer cost with each node and exchange this cost value with each update message. The cost increases monotonically if the system diverges. Therefore, discarding advertisements from nodes whose cost is greater than a threshold is suggested. However, with this approach, cost of the nodes involved in the same conflict are very close (or the same). Therefore, all such nodes may end up giving up their most preferred paths at the same time as soon as their cost reaches the threshold. The other disadvantage of the Cobb and Musunuri algorithm is keeping per node cost, which causes aggregation of the paths through the same node. In this case, one flapping path may cause all the alternative paths (through the same node) to be eliminated. Cobb and Musunuri algorithm cannot reduce the cost of the nodes easily, which is essential when the topology and/or policies change. Resetting costs allows nodes to follow one more time their routing policies, where possibly this time no conflicts exist. Cobb and Musunuri suggest periodically resetting the cost of all nodes via a distributed reset protocol [21], which is based on a diffusing computation over a min-hop spanning tree. They also suggest resetting costs once a week or once a month. However, to limit the number of path eliminations, resetting costs should be performed as soon as the state of the system changes.

Other Work: Another line of work concentrated on solving policy conflicts by treating the routing problem as a game in which the ASes are strategic agents [22]. For the case where AS policies are restricted to the hierarchical relationships, Feigenbaum $et\ al.$ [22] present a *strategy-proof mechanism* that can be computed in polynomial time in a centralized computational model. However, it is shown that this mechanism is incompatible with BGP: If this mechanism is computed by a BGP-like distributed algorithm with similar data structures and communication patterns, it may cause BGP to converge very slowly, and/or can trigger extensive amount of update messages. [22] also show that if AS policies are not restricted, then it is NP-hard to compute a routing tree that maximizes the overall utility of the ASes.

III. Adaptive Policy Management Scheme (APMS)

A. Overview

We propose a new algorithm to dynamically detect and eliminate policy conflicts leading to BGP divergence. The idea is to locally detect the paths involved in a conflict, and eliminate the conflict by changing the relative preference of such paths.

Each node involved in a particular conflict observes *route flaps*: Constantly adopting a path and later abandoning it for another path. The flapping paths are the ones whose relative preference is conflicting with the preference of some other node. The node can control the flaps it is experiencing by sticking with the less preferred but more stable path, even when the better alternative is advertised. This can be achieved effectively by lowering the local preference value of the higher preferred path. When the node stops advertising paths alternately, the cyclic effect of the global conflict will be broken.

When the paths flap during persistent oscillation, it is mainly because of the node's preference over the current available paths through its neighbors (peers). Note that not every advertisement that the node receives during such instabilities is changing, *i.e.* not every route learned from its peers is constantly advertised and then withdrawn, and then re-advertised, and so on. We refer to the paths whose advertisement is not changing over time as *safe paths*, and suggest that the nodes experiencing route flaps choose the safe paths instead of better preferred paths to stop oscillation. If there are more than one stable path, the node chooses the most preferred stable path, and make rank changes accordingly.

To be able to locally detect the routes that are flapping repeatedly, each node needs to keep some form of *local history*. The information kept in the local history should be enough to distinguish paths that are constantly flapping. $Count$ values are associated with the paths in the local history, and increased by one for each related flap, and reset whenever there is a change in path preferences. Since the algorithm we are proposing is distributed and based on using only local information, there maybe many nodes synchronously detecting the same conflict and lowering the preferences of their higher preferred paths. To prevent this kind of simultaneous and unnecessary path preference changes, we suggest changing relative preferences probabilistically.

Because of the probabilistic adjustment of path preferences, even though the effect of a particular conflict is observed several times, it is possible that the conflict remains unresolved. $max\_threshold$ is introduced to handle such cases: When the $count$ associated with a particular path exceeds $max\_threshold$, then the path is removed from the set of permitted paths, and added to the set of *bad paths*. The

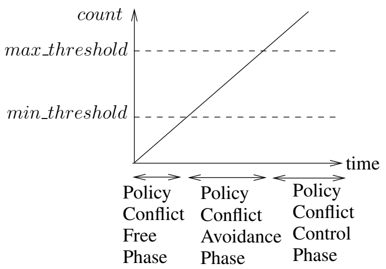

Fig. 2. Three phases of Adaptive Policy Management Scheme (APMS)

bad paths set is a data structure that keeps the list of paths which the node believes that their adoption leads to a conflict. Therefore, they are excluded from further consideration in the best path selection process (until they are restored as the algorithm adapts to a conflict-free state), even if they are advertised by peers and permitted by original local policies. Setting $max\_threshold$ to higher values helps reduce the number of paths placed in *bad paths*. However, smaller values may reduce convergence time. $Count$ values kept in the local history of a node are compared against $min\_threshold$, and $max\_threshold$ to detect and handle divergence as follows: (a) **Policy conflict-free phase**: When the *counts* are smaller than $min\_threshold$, then the node assumes that there is no persistent oscillation. Therefore, setting $min\_threshold$ to higher values helps prevent path preference changes when the oscillation is transient; (b) **Policy conflict-avoidance phase**: If any $count$ exceeds $min\_threshold$, but stays lower than $max\_threshold$, then the node assumes that there is a policy conflict leading to persistent oscillation, which can be avoided by changing the relative preference (rank) of the paths; (c) **Policy conflict-control phase**: If any $count$ exceeds $max\_threshold$, then the path associated with this $count$ is added to a set of *bad paths*, and excluded from further consideration in the best path selection process. Figure 2 shows these three different phases of our algorithm.

B. Details of APMS

There maybe different instantiations of the algorithm depending on the exact nature of the path information kept in the local history, and the way $count$ values are associated with the paths. In this section, we describe the instantiation that we have chosen. Throughout this section, we assume that there is a single destination. Node u has *adopted path p* means that node u has chosen path p as its best path, and currently using it to reach the destination. Node u has *abandoned path p* means that node u was using path p to reach destination, and since node u's best path has changed, node u is no longer using p to reach the destination.

1) Data Structures: In addition to data structures required for BGP, APMS requires the usage of the following data structures:

- Each node u keeps a *local history* in the form of ($path$, $count$) tuples, where $path$ indicates a path that has been recently adopted by node u, and $count$ indicates how many times the $path$ has been adopted and later abandoned. When there is divergence, some $count$ values keep increasing because of constant flapping.
- $peerStability$ is an integer associated with each peer of node u. When $w \in peers(u)$ sends an update for a particular destination that advertises a path p that is different from the one that has been advertised previously, *i.e.* rib_in($u \Leftarrow w$) $\neq$ p, the $peerStability$ value corresponding to peer w is increased. The purpose of this counter is to differentiate the peers, and hence the paths advertised by those peers, that are stable. Therefore, if node u is observing a route flap, the flap can be stopped by adopting a path advertised by *a stable peer*: The smaller value of $peerStability$ indicates a more stable peer. If $peerStability$ is equal to one for peer w, it means the path advertised by w never changed later. We refer to such paths as *safe paths* and the peers advertising these paths as *stable peers*. To make stability last, after adopting a stable path, node u also changes its preference of the paths to reflect this choice. Node u updates the local preference of the stable path so that it will be the most preferred path, *i.e.* $rank(safe\,path) = 1$, where $rank(p)$ is the index of path p among the current alternative paths in the order of decreasing local preference value. If there are more than one safe path, the one that is originally preferred more is chosen.
  Note that $count$ values associated with $path$s in *local history* cannot be used to measure stability of peers. A path p advertised by w may have a high $count$ value associated with it, even if w never changes this advertisement. If the advertisement sent by peer w does not change, $peerStability$ will always be equal to 1 for peer w even if the node receiving this update may constantly adopt p and later abandon it for a higher preferred path as a result of a policy conflict. In such case, $count$ associated with p keeps increasing as the node constantly adopt p and later abandon it for a higher preferred path.
- B indicates the *bad path* set, which keeps the paths whose $count$ exceeds the $max\_threshold$ value. Such paths are eliminated from the permissible set of paths at node u and not considered in the best path selection process even though they maybe advertised by a peer.
- $keepaliveCount$ is used to count the number of times a KEEPALIVE message is received from a peer w. If the value of $keepaliveCount$ exceeds a threshold, $ka\_threshold$, for all peers of node u, then node u concludes that the system has stabilized and there are no more policy conflicts. After this point, node u probabilistically (and more conservatively) resets some of the local preferences back to their original values [2]. Although there is a possibility of introducing instability back into the system, our algorithm adapts to the changes dynamically and stabilizes at some state of path preferences eventually.

The state of the system is defined by the values kept in these data structures, as well as the path orderings at each node, which correspond to different policies.

2) Update Handling: Figure 3 shows the pseudo-code of the Adaptive Policy Management Scheme (APMS) for handling routing updates. The process runs at each node u in response to a received update. When node u adopts a path $p \in \mathcal{P}^u$, it informs each of its peers by sending an update message. rib(u) indicates the current best path to the destination selected at node u. rib_in($u \Leftarrow w$) indicates the most recent path sent from $w \in peers(u)$, and processed at node u. The set of path choices available at node u that are considered for best path selection, excluding the bad paths in $B(u)$, is defined as

$$choices_B(u) = \{(u,w)rib\_in(u \Leftarrow w) - B(u) | w \in peers(u)\} \bigcap \mathcal{P}^u$$

and the best path as

$$best_B(u) = max(u, choices_B(u)).$$

As long as node u receives advertisements from its peers, $best_B(u)$ is recomputed with the most recent $choices_B(u)$. When $rib(u)$ changes, node u notifies its peers by sending an

[2]This is akin to increase/decrease adaptation rules employed in many adaptive feedback-control systems.

update message.

When the process goes into the policy conflict-avoidance phase, after successfully choosing a safe path and changing its rank to be most preferred, the state of the system changes. The state of the system also changes when the process places a path in the *bad path* set, *i.e.* in the policy conflict-control phase. In either case, this new state corresponds to a different Stable Paths Problem (SPP), possibly a stable one. Therefore, counters are reset to give opportunity for a fresh start and to see if the change is enough to reach stability. When the process goes into the policy conflict-avoidance phase, if there is no *safe path* for node u, *i.e.* $peerStability(w) \neq 1$ for any $w \in peers(u)$, then node u does not do anything to stop the oscillation. However, if there is a safe path P_{safe}, with probability 1/2, the path ordering at node u is changed such that $rank(P_{safe}) = 1$. If there are more than one safe path, then the most preferred one is chosen as the highest ranked path.

3) Increasing Local Preferences: When the system stabilizes, only KEEPALIVE messages are exchanged between peers. Figure 4 shows how node u probabilistically restores some rank changes for its paths. If restoring the local preference of a path does not affect the stability of the system, then the corresponding rank change was *unnecessary* for the stability of the system in the first place, and it is safe to restore such changes. Since policies are placed for a purpose by each node, such as traffic engineering or security or cost, it is important for ASes not to change them unless it is conflicting with the policies of other nodes and absolutely necessary to eliminate route oscillations. Although it is safe to restore rank changes that do not compromise the current stability, there is no way of knowing for node u which changes are safe to restore. Therefore, node u uses a probabilistic (albeit more conservative) approach, and risks introducing instability back into the system. Contrary to update handling, node u increases the local preference of a path with a much smaller probability, 1/4. We allow for bringing paths out of $B(u)$ with probability 1/4 as well, since the above argument is also true for the paths currently in the *bad path* set. If node u performs a rank change and/or remove (restore) a path from $B(u)$, counters kept in the local history are reset because this new state corresponds to a different network state. For this new state, the best path is computed, and if it is different from the current best path, an update is sent to the peers.

Note that re-using suppressed paths and/or restoring original ranks once the system reaches a stable state may introduce instability back into the system. However, this is the nature of all adaptive feedback control systems. By using much smaller probability for reset, *i.e.* 1/4, we provide a conservative way of probing the network state. This is akin to the congestion avoidance mechanism of TCP, during which the current state of the network is probed at a slower rate. Cobb and Musunuri algorithm [9] suggests periodically resetting the cost of all nodes via a distributed reset protocol [21] for this purpose. However, the algorithm lacks any mechanism to prevent introducing the same conflict back into the system when the costs are reset. Therefore, this approach causes the system to oscillate between the stable and the unstable state unless policy conflicts disappear in the meantime due to a topology/policy change. SPVP [7] resets $history$ when a cycle is detected and a path is eliminated. However, to be able to truly adapt to the system dynamics, SPVP should also reset the state by re-introducing all suppressed paths. Otherwise, the algorithm prevents usage of the suppressed paths even if the policy conflict disappears due to a topology/policy change. Similar to Cobb and Musunuri algorithm, such reset may cause oscillation between the original unstable state and the stable state. Resetting the states conservatively using the mechanism shown in Figure 4 makes APMS the only algorithm that allows adapting to every state of the network automatically while oscillating in the vicinity of a stable state at a very

```
process APMS_Update_Handling[u]          //Update Handling
  receive Update m from peer w   ⟶
    keepaliveCount(w)=0
    if rib_in(u ⇐ w) ≠ m            then
      peerStability(w)++
    rib_in(u ⇐ w) = m
    if rib(u)≠ best_B(u)  then
      P_old=rib(u)
      P_new = best_B(u)
      if(P_new ≠ ε)     then
        count(P_new)++
      if count(P_new) > max_threshold     then
        B(u) = B(u) ⋃ {P_new}                 //Policy Conflict−Control Phase
        P_new = best_B(u)
        count(Q)=0 for each path Q ∈ localHistory
        peerStability(v)=0 for each v ∈ peers(u)
      else if count(P_new) > min_threshold     then
        do with probability= 1/2
          find the most preferred safe path, P_safe    //Policy Conflict−Avoidance Phase
            rank(P_safe)=1
            P_new = P_safe
            count(Q)=0 for each path Q ∈ localHistory
            peerStability(v)=0 for each v ∈ peers(u)
    if P_new ≠ P_old   then
      rib(u) =P_new
      for each v ∈ peers(u)   do
        send rib(u)   to v
Note: The code to the right of ⟶ is assumed to be executed in one atomic step
```

Fig. 3. Adaptive Policy Management Scheme (APMS): Update Handling

```
process APMS_Keepalive_Handling[u]       //Keepalive Handling
  receive keepalive from w   ⟶
    keepaliveCount(w)++
    if keepaliveCount(v) ≥ ka_threshold for every v ∈ peers(u)
     for each v ∈ peers(u)
      r=rib_in(u ⇐ v)
      if (localpref(r) ≠ originallocalpref(r)) || (r ∈ B(u))   then
        do with probability= 1/4
          if r ∈ B(u)   then
            remove r from B(u)
          localpref(r)= originallocalpref(r)
          count(Q)=0 for each path Q ∈ localHistory
          peerStability(v)=0 for each v ∈ peers(u)
          keepaliveCount(v)=0 for each v ∈ peers(u)
          P_new=best_B(u)
          if P_new ≠ rib(u)      then
            rib(u) =P_new
     for each v ∈ peers(u)   do
       send rib(u)   to v
Note: The code to the right ~~of~~⟶ is assumed to be executed one atomic step
```

Fig. 4. Adaptive Policy Management Scheme (APMS): Increasing Local Preferences once Stability is Reached

slow rate. With APMS, the backoff and recovery of path preferences could also be guided by explicit input from local AS administrators. We will investigate this approach in a future report.

4) Handling Transient Oscillations due to Topology Changes: If there is a topology change, path updates experienced as a result of a change in topology may interfere with diagnosing policy conflicts. For example, link or node failure or recovery may create a route flap, and increase the chance of false positives. More importantly, topology changes affect policy dynamics. Even if the original topology had policy conflicts, the new topology maybe conflict-free, or vice versa. Therefore, during the process of resolving policy conflicts, it is important for the nodes to be aware of link/node failure and recovery events and distinguish them from route flaps due to policy conflicts.

Assume that node u's next-hop along the path to the

process APMS_TopologyChange_Handling[u] //Handling topology changes
//when node u learns that its next-hop link along
//the path to the destination has failed or restored
//or the next-hop node failed or restored
if link (u, v) has failed or node v has failed
r= rib_in$(u \Leftarrow v)$
if r is permissible
feasible(r)=false
if link (u, v) is restored or node v is restored
receive Update m **from** v
rib_in$(u \Leftarrow v) = path(m)$
//reset local states
B(u)={}
peerStability(w)= 0 for every $w \in peers(u)$
keepaliveCount(w)=0 for every $w \in peers(u)$
count(Q)=0 for each path $Q \in localHistory$
for every $w \in peers(u)$
$p \in$ rib_in $(u \Leftarrow w)$
if p is permissible
if localpref(p) $\neq$ originallocalpref(p)
localpref(p)=originallocalpref(p)
sequenceNumber++ //increase the sequence number of u
rib(u)=best$_B(u)$
for every $w \in peers(u)$
send update message m
where path(m)=rib(u), originator(m)=u, sequenceNumber(m)=sequenceNumber

Fig. 5. Adaptive Policy Management Scheme (APMS): Handling Failures and Recovery

destination is node v, *i.e.* $P = < u,\ v,\ \ldots,\ destination >$. When the link between u and v goes down, as soon as node u detects the change, it discards the route learned from v, and recomputes its best path to the destination. At this point, node u knows that its best path has changed because of a failure. We suggest that while sending the resulting update message, node u includes some information about the failure so that the other nodes which are not in the neigborhood of the failed link can deduce that the update they are receiving is triggered by a failure. Since from node u's perspective, the effects of failure of the peer v is similar, it can be handled the same way. Furthermore, link/peer restorations can also be handled in a similar way by observing OPEN messages exchanged when a peering TCP session is (re-)established.

We suggest adding two new fields to the update messages of BGP for this purpose: $originator$ and $sequenceNumber$.

process APMS_TopologyChange_Handling[u] //Handling topology changes
//when node u gets an update message m from node v with $originator(m) \neq null$
if $u == originator(m)$
//process this update as shown in APMS_Update_Handling[u]
if there is a new update to be sent
set $originator = null$ before sending update to the peers
else if ($sequenceNumber(m) > sequenceNumber$ of $originator(m)$ at node u)
B(u)={}
peerStability(w)= 0 for each $w \in peers(u)$
keepaliveCount(w)=0 for every $w \in peers(u)$
count(Q)=0 for each path $Q \in localHistory$
for every $w \in peers(u)$
$p \in$ rib_in $(u \Leftarrow w)$
if p is permissible
if localpref(p) $\neq$ originallocalpref(p)
localpref(p)=originallocalpref(p)
//update sequenceNumber of $originator(m)$ at node u
$sequenceNumber$ of $originator(m)$ at u=$sequenceNumber(m)$
rib(u)=best$_B(u)$
for every $w \in peers(u)$
send update message n
where path(n)=rib(u), originator(n)=originator(m)
sequenceNumber(n)=sequenceNumber(m)

Fig. 6. Adaptive Policy Management Scheme (APMS): Handling Failures and Recovery

While $originator$ carries the ID of the node that has detected the topology change, $sequenceNumber$ indicates the number of times the node detected a topology change. Each node also keeps track of the $sequenceNumber$ for the other nodes to distinguish the most recent topology update when the $originator$ is the same. For example, if the same link constantly going up and down quickly, the resulting updates will have the same $originator$. When these updates propagate at different speed, it is important to be able to distinguish which update reflects the latest topology change. With this addition, the messages exchanged between peers are triples $(P, originator, sequenceNumber)$. In the absence of topology changes, the $originator$ field is set to $null$. The pseudocode of the topology handling algorithm is shown in Figures 5 and 6.

When node u detects a topology change, it first resets the local state, and increases its $sequenceNumber$ as shown in Figure 5. The resulting update is sent with $originator$=u, and the current $sequenceNumber$ of node u. As the pseudocode in Figure 6 shows, when node u receives an update message m with $originator$=v and the $sequenceNumber$ indicating that this is a new topology change, node u temporarily turns off the policy conflict detection process and resets the local state. After updating the $sequenceNumber$ of v for its records, node u sends the resulting update message with the same $originator$ and $sequenceNumber$ values to further propagate the change. When node u receives an update message with itself as the $originator$, it restarts the policy conflict detection process by assigning $null$ to $originator$. Policy changes can be handled the same way as topology changes. Since a policy change is performed by a local authority, the AS performing policy change can use the above mechanism to provide temporary suspension of processing the resulting updates for policy conflict detection.

While this mechanism handles transient oscillations due to topology/policy changes, $min\_threshold$ still helps not to react too soon to transient oscillations arising from other causes.

IV. Convergence Analysis of APMS

Different path orderings at each node define different states of the network and correspond to different policies. Among these states, there are some stable configurations. Our goal is to show that starting with an arbitrary state of the system, the Adaptive Policy Management Scheme (APMS) converges to a stable state within a finite number of steps. To that end, we list some formal definitions for terms we use henceforth:

Definition 4.1: **Conflict-free node** is a node which is not involved in any policy conflict.

Definition 4.2: **Non-flapping path, or stable path** $P =< v, \ldots, destination >$ is the best path of a conflict-free node v, which does not change over time.

Definition 4.3: **Safe path** $(u, v)P$ is a permitted path at node u, where v is a peer of u, and v is advertising the non-flapping path $P =< v, \ldots, destination >$.

Definition 4.4: **Conflicting safe-alternative node** is a node which is involved in a policy conflict, and has an alternative safe path which is not most preferred.

Definition 4.5: **Conflicting node** is a node which is involved in a policy conflict, and does not have a safe path.

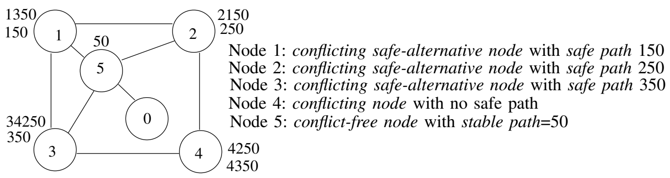

Fig. 7. An example showing different types of nodes.

Figure 7 shows an example of a group of nodes, *i.e.* {1,2,3,4}, involved in a cyclic conflict. None of the nodes in such group is *conflict-free*, but either a *conflicting safe-alternative* or a *conflicting* node. *Conflicting safe-alternative nodes* can break the conflict by holding onto their safe paths, *i.e.* changing path rankings such that the safe path becomes the most preferred path. If a node does not have any safe paths, *i.e. conflicting node*, then it cannot stop oscillation through a rank change of its paths. However, as soon as one of its next-hop neighbor stabilizes, it will start to observe stable path advertisements coming from this newly stabilized neighbor. At this point, the node becomes either a *conflicting safe-alternative node*, or a *conflict-free node*. For the example shown in Figure 7, as soon as node 2 changes its path preference to prefer (250) more than (2150): Node 2 becomes a *conflict-free* node; path (250) becomes a *stable path* of node 2; path (4250) becomes a *safe path* at node 4 and node 4 stabilizes on it since it is its most preferred path; node 3 then chooses its most preferred path (34250) through node 4; node 1 has no choice but to stabilize on path (150); and the conflict is resolved.

First, we assume that there is a single cyclic conflict in the system. Let N denote the set of nodes that are in this conflict, where $|N| \geq 2$. Since *conflicting safe-alternative nodes* are playing the key role at breaking conflicts, we would like to show that there must be some *conflicting safe-alternative nodes* in N for such a cyclic conflict to occur. Let M denote the set of such nodes. Obviously, the nodes in M have paths which they prefer more over their safe path, thus causing a cyclic conflict. Throughout the conflict, the better preferred paths are constantly advertised and withdrawn. If any node in M changes its preference to pick its safe path over its more preferred but oscillating path, then it can break this cyclic conflict. Independent of the size of the nodes in M, only one *conflicting safe-alternative node* suffices to perform path rank change for breaking the conflict. If we assume the nodes in M are $u_1, \ldots, u_k$, they will form a dispute wheel as shown in Figure 1. Node u_is, where $1 \leq i \leq k$, are the nodes at which route preferences cause the shown dispute wheel. Node u_i constantly changes its path from Q_i to its more preferred path $R_i Q_{i+1}$, which in return causes u_i to give up $R_i Q_{i+1}$ and use Q_i. Q_i is a safe path for u_i since we assumed that this is the only conflict in the system. If at least one node, u_i, changes the preference of its safe path so that Q_i is more preferred than $R_i Q_{i+1}$, the dispute wheel cannot form.

There maybe two types of persistent oscillations: *Type 1* is persistent oscillation due to two different solutions, and *type 2* is persistent oscillation due to the lack of any solution.

Lemma 4.1: Under BGP, policies of a group of nodes cannot lead to any type of persistent oscillation if only one of the nodes in this group is a *conflicting safe-alternative node*.

Proof: We prove this by contradiction: Assume that $u_1 \ldots u_n$ are the group of nodes who are involved in the persistent oscillation, and only one of the nodes in this group has a safe path. Let u_1 denote this only *conflicting safe-alternative node*, and P_1 denote its safe path. The other nodes in the group, $u_2 \ldots u_n$, are *conflicting nodes*, and their permitted paths must be traversing at least one node involved in the conflict because otherwise their paths would be safe. Let u_i be a *conflicting node*, where $2 \leq i \leq n$. Permitted paths of u_i must be in the form of $P_i =< \cdots, u_j, \cdots, destination >$, where u_j is a node in the conflict, *i.e.* $1 \leq j \leq n$ but $j \neq i$. (Since BGP requires the paths to be *simple*, *i.e.* no repeated nodes, the path P_i cannot traverse u_i.) Without losing generality assume the following: If there are multiple nodes involved in the conflict along path P_i, node u_j is the *closest* one to the destination. There are two cases: (1) $j \neq 1$; (2) $j = 1$. Case (1) cannot be true, because it implies that the subpath of P_i following u_j to the destination does not traverse any node in the conflict, and therefore this path is a safe path for node u_j. However, this makes node u_j a *conflicting safe-alternative node*, which contradicts our assumption that u_1 is the only *conflicting safe-alternative node* in the conflict. Therefore, case (2) holds for every node u_i where $2 \leq i \leq n$: The *conflicting nodes* reach the destination through u_1.

P_1 cannot include any other node in the dispute, otherwise it wouldn't be safe. For the presence of cyclic conflict, in addition to P_1, u_1 must have another permitted path P_2, which is not safe and more preferred than P_1. To satisfy this requirement under our assumption, P_2 must be advertised by a node in the dispute. However, this cannot be true since such a path cannot be *simple*: As explained in the previous paragraph, the paths advertised by the conflicting nodes, *i.e.* u_is where

$2 \leq i \leq n$, traverse u_1. Extension of such paths at node u_1 will have a loop and BGP does not allow such paths. ■

Lemma 4.2: If at least 3 nodes are *conflicting safe-alternative nodes* in a persistent oscillation, then the persistent oscillation is a type 2 oscillation.

Proof: Assume that $u_1 \dots u_n$ are the group of nodes which are involved in the persistent oscillation, and only three nodes in this group have safe paths. Let u_j, u_k, and u_h denote these *conflicting safe-alternative nodes*, and P_{j_1}, P_{k_1}, and P_{h_1} denote their safe paths, respectively. The other nodes in the group, $\{u_1 \dots u_n\} - \{u_j, u_k, u_h\}$, are *conflicting nodes*, and their permitted paths must be traversing at least one node involved in the conflict because otherwise their paths would be safe. Let u_i be a *conflicting node*, where $1 \leq i \leq n$, and $i \neq j$, $i \neq k$, $i \neq h$. Permitted paths of u_i must be in the form of $P_i =< \dots, u_l, \dots, destination >$, where u_l is a node in the conflict, *i.e.* $1 \leq l \leq n$ but $l \neq i$. (Since BGP requires the paths to be *simple*, the path P_i cannot traverse u_i.) Without losing generality assume the following: If there are multiple nodes involved in the conflict along path P_i, node u_l is the *closest* one to the destination. There are two cases: (1) $l \neq j$, $l \neq k$, and $l \neq h$; (2) $l = j$ or $l = k$ or $l = h$. Case (1) cannot be true, because it implies that the subpath of P_i following u_l to the destination does not traverse any nodes in the conflict, and this path is a safe path for node u_l. This makes u_l a *conflicting safe-alternative node*, which contradicts our assumption that u_j, u_k, and u_h are the only *conflicting safe-alternative nodes*. Therefore, case (2) holds for every node u_i where $1 \leq i \leq n$, and $i \neq j$, $i \neq k$, $i \neq h$, i.e. the permitted paths of the *conflicting nodes* must reach the destination through u_j or u_k or u_h. Therefore, the path assignments that *conflicting nodes* may have in the end as a solution is basically determined by the path assignments of the *conflicting safe-alternative nodes*, u_j, u_k, and u_h.

P_{j_1}, P_{k_1} and P_{h_1} cannot include any other node in the dispute, otherwise they wouldn't be safe. However, to be a *conflicting safe-alternative node*, u_j, u_k, and u_h must have other permitted paths, P_{j_2}, P_{k_2}, and P_{h_2}, respectively, which are not safe, and more preferred than their safe paths. Since u_j, u_k, and u_h are the only *conflicting safe-alternative nodes*, to be able to have a cyclic conflict, P_{j_2}, P_{k_2}, P_{h_2} must be in the following forms: $P_{j_2} =< \dots, u_k, \dots, destination >$, where the subpath of P_{j_2} from u_k to the destination is P_{k_1}. $P_{k_2} =< \dots, u_h, \dots, destination >$, where the subpath of P_{k_2} from u_h to the destination is P_{h_1}. $P_{h_2} =< \dots, u_j, \dots, destination >$, where the subpath of P_{h_2} from u_j to the destination is P_{j_1}, which completes the cyclic policy conflict. However, as can be seen from the dispute wheel shown in Figure 8, this case is very similar to BAD GADGET, where [17] proves that there is no solution for it. The case where there is more than 3 *conflicting safe-alternative nodes* in a persistent oscillation is similar. ■

Theorem 4.1: Under BGP, exactly 2 nodes are *conflicting safe-alternative nodes* in a persistent oscillation iff the persistent oscillation is a type 1 oscillation.

Proof: If exactly 2 nodes are conflicting safe-alternative nodes in a persistent oscillation, then the persistent oscillation is a type 1 oscillation: Assume that $u_1 \dots u_n$ are the group of nodes which are involved in the persistent oscillation, and only two nodes in this group have safe paths. Let u_j and u_k denote these *conflicting safe-alternative nodes*, and P_{j_1} and P_{k_1} denote their safe paths, respectively. The other nodes in the group, $\{u_1 \dots u_n\} - \{u_j, u_k\}$, are *conflicting nodes*, and their permitted paths must be traversing at least one node involved in the conflict because otherwise their paths would be safe. Let u_i be a *conflicting node*, where $1 \leq i \leq n$, and $i \neq j$, $i \neq k$. Permitted paths of u_i must be in the form of $P_i =< \dots, u_l, \dots, destination >$, where u_l is a node in the conflict, *i.e.* $1 \leq l \leq n$ but $l \neq i$. (Since BGP requires the paths to be *simple*, *i.e.* no repeated nodes, the path P_i cannot traverse u_i.) Without losing generality assume the following: If there are multiple nodes involved in the conflict along path P_i, node u_l is the *closest* one to the destination. There are two cases: (1) $l \neq j$ and $l \neq k$; (2) $l = j$ or $l = k$. Case (1) cannot be true, because it implies that the subpath of P_i following u_l to the destination does not traverse any nodes in the conflict, and this path is a safe path for node u_l. This makes u_l a *conflicting safe-alternative node*, which contradicts our assumption that u_j, and u_k are the only *conflicting safe-alternative nodes*. Therefore, case (2) holds for every node u_i where $1 \leq i \leq n$, and $i \neq j$, $i \neq k$, i.e. the permitted paths of the *conflicting nodes* must reach the destination through u_j or u_k. Therefore, the path assignments that *conflicting nodes* may have in the end as a solution is basically determined by the path assignments of the *conflicting safe-alternative nodes*, u_j and u_k.

P_{j_1} and P_{k_1} cannot include any other node in the dispute, otherwise they wouldn't be safe. However, to be a *conflicting safe-alternative node*, u_j and u_k must have other permitted paths, P_{j_2} and P_{k_2}, that are not safe, and more preferred than their safe paths. Since u_j and u_k are the only *conflicting safe-alternative nodes*, to be able to have a cyclic conflict, P_{j_2} and P_{k_2} must be in the following forms: $P_{j_2} =< \dots, u_k, \dots, destination >$, where the subpath of P_{j_2} from u_k to the destination is P_{k_1}, and $P_{k_2} =< \dots, u_j, \dots, destination >$, where the subpath of P_{k_2} from u_j to the destination is P_{j_1}. Then the two possible stable solutions are either u_j or u_k choosing its safe path.

If a persistent oscillation is a type 1 oscillation, then exactly 2 nodes must be conflicting safe-alternative nodes: Since the persistent oscillation is type 1, there are 2 different stable solutions. Having two different stable solutions when there is a persistent oscillation is not possible with only one *conflicting safe-alternative node*, because in such case there can only be one stable solution (as we have shown in Lemma 4.1). When there is a persistent oscillation, having two different stable solutions is not possible with three or more *conflicting safe-alternative nodes*, because for this case there is not even a single stable solution as shown in Lemma 4.2. ■

Theorem 4.2: Under BGP, at least 3 nodes are *conflicting safe-alternative nodes* in a persistent oscillation iff the persistent oscillation is a type 2 oscillation.

Proof: If the persistent oscillation is a type 2 oscillation, then at least 3 nodes must be conflicting safe-alternative nodes: For type 2 oscillation, there is no solution. If only 1 node has a safe path, there is no persistent oscillation as shown

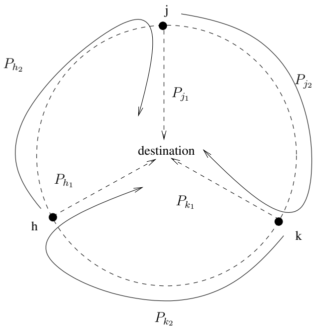

Fig. 8. Dispute wheel for the proof of Lemma 4.2

in Lemma 4.1. If only 2 nodes are *conflicting safe-alternative nodes*, there is type 1 oscillation as shown in Theorem 4.1. Therefore, there must be 3 or more *conflicting safe-alternative nodes* for type 2 oscillation to occur.

The converse is the same as Lemma 4.2. ■

For the above discussion, we have assumed that there is only one conflict involving a group of nodes. If there are multiple conflicts, nodes may get involved in many conflicts simultaneously. Then the safe paths of the nodes that are in a particular conflict may or may not be *observable*.

Definition 4.6: **Observable safe path** is a safe path $P =< u, v, \ldots, destination >$ at a node u if none of the other nodes along this path observes route flaps due to any conflicts.

Definition 4.7: **Innermost conflict** along the path $P =< u_k, u_{k-1}, \ldots, u_2, u_1, destination >$ is the conflict that involves node u_i, where u_i is the *closest* node to the destination and involved in a conflict. In this case the **innermost safe path** along the path P is $< u_i, u_{i-1}, \ldots, destination >$.

Definition 4.8: A **conflict-free node with observable safe path** is a *conflict-free node* whose most preferred path does not traverse any nodes in the conflict that has not been resolved yet, *i.e.* its most preferred safe path is *observable*.

Once the *innermost conflicts* along the safe paths are broken, the nodes of the outer conflicts start to observe their safe paths, and have a chance to break their conflict under APMS by sticking to their safe (albeit less preferred) path. For each conflict, by Theorems 4.1 and 4.2, we have at least 2 *conflicting safe-alternative nodes*. Starting from the innermost conflicts, by having at least one of these *conflicting safe-alternative nodes* perform path rank change at each step, we break the cyclic conflicts, and increase the number of *conflict-free* nodes that are stabilized on their paths. This is the idea behind the convergence proof of our Adaptive Policy Management Scheme (APMS).

Denote by S the set of nodes that are *conflict-free* and stabilized on their paths during the execution of the algorithm. During the k^{th} **step** of APMS, the nodes that are currently in S advertise their paths to their peers. The peers process these advertisements, and enter into S after stabilizing on their paths. However, the k^{th} step does not end until the resulting new peers that are "*conflict-free* with observable safe path" are also processed iteratively until no more "*conflict-free* node with observable safe path" can be reached.

A node changes its state over time as it runs APMS. APMS helps the node to stabilize by changing its path preferences by increasing the preference of the paths that are not changing over time. A *conflicting safe-alternative node* becomes *conflict free* when it performs rank change to make its safe path the most preferred path and stabilizes on it. A *conflicting node* u with flapping paths $(u, v)P_v$ and $(u, w)P_w$, becomes a *conflicting safe-alternative node* when the node advertising path P_v enters in S stabilizing on path P_v.

Lemma 4.3: During the execution of the APMS algorithm, the size of the set of nodes that are *conflict-free* and stabilized on their paths (set S) increases monotonically if we perform path rank changes whenever conflict is detected.

Proof: The nodes in S form a routing tree of the paths on which they are stabilized. This routing tree is rooted at the destination and grows as the nodes outside of S adopt and stabilize on the extension of the paths advertised by the nodes in S.

To show that S grows monotonically, we need to show that at each step of the algorithm, at least one conflict is resolved and at least one node is added to the set until the system converges. We use induction based on the number of nodes in S.

Basis: At the beginning, while S was empty, the destination is added. Hence it holds for the base case.

Hypothesis: At step k of the execution, assume the size of the set is n and up to this point the set S grew monotonically.

Induction Step: We need to show that at step $k + 1$, the size of S will be greater than n. When node u is about to be a member of S, one of the following is true:

Case 1) Node u has just received a path advertisement P_v from node v in S, where $(u, v)P_v$ is a permitted path at node u, but node u is not stabilizing on this path. Since $(u, v)P_v$ is a safe path of node u, node u must be in a policy conflict, *i.e.* node u is a *conflicting safe-alternative node*. At this point, node u performs rank change and sticks to the path $(u, v)P_v$, becomes a *conflict-free node*, and is added to S. By Theorems 4.1 and 4.2, we know that if there is a conflict, there will be at least 2 *conflicting safe-alternative nodes* and this is the step where they break the conflict. Consequent "*conflict-free* nodes with observable safe paths" are also handled iteratively, and entered into S until hitting the nodes that are not "*conflict-free* with observable safe paths".

Case 2) Node u was a *conflicting node* at previous step (step k), and observing path flap in the form of $P1_u$, $P2_u$, $P1_u$, $P2_u$, ... Assume that $P1_u$ is more preferred than $P2_u$ at node u, and $P1_u = (u, v)P_v$, where P_v is advertised by node v and $P2_u = (u, w)P_w$, where P_w is advertised by node w. At step k, neither v nor w was in S, since otherwise node u woudn't be a *conflicting node*. At the end of step k due to a resolved conflict, node v and/or node w might have entered S. The possible cases at step $(k+1)$ due to a resolved conflict at step k are: (i) Node v is in S and stabilized on path P_v, and node w is not in S; (ii) Node w is in S and stabilized on path P_w, and node v is not in S; (iii) Both node v and w are in S and stabilized on paths P_v and P_w, respectively. If (i) happens, node u stabilizes on $P1_u$ and becomes a conflict free node. If (ii) happens, node u becomes a *conflicting safe-alternative node* with safe path $P2_u$. Note that node u is still observing

$P1_u$ but this path is not safe since node v has not stabilized on P_v yet, *i.e* not in S. Then case 1 applies (*i.e.* node u makes path $P2_u$ its most preferred path, and stabilizes on it). If (iii) happens, node u stabilizes on its $P1_u$ and becomes a conflict free node.

Case 3) Node u was a *conflict-free node* at the previous step but its safe paths are not *observable* due to unresolved inner conflicts. If this is the case, node u's experience is the same as if it were a *conflicting safe-alternative node* or *conflicting node* depending on how many of its safe paths are not *observable*. Therefore, case 1 or case 2 applies.

Case 4) If none of the paths advertised by nodes in S are permitted at receiving node u, then nothing will happen and the size of S will stay the same. However, node u will become a *conflict-free node* converging to ϵ, *i.e.* unreachable destination. This also implies that not only immediate neighbors of nodes in S, but all the nodes outside of the set S at that point will converge to ϵ. Then the APMS algorithm returns with a stable routing tree. ■

Note that once a node is in S, during the execution of APMS, it does not get out of S. This is because as node u enters in S, it updates its path preferences so that the its safe is the most preferred path. Therefore, node u will not give up this path for any other path that node u may learn in the future. The only time when node u enters S and does not update its path preferences is when node u is already a *conflict-free* node. However, in such a case, node u would stabilize on its best path as it enters S, and remains there as a *conflict-free* node.

Theorem 4.3: Starting from an arbitrary state of the system, the Adaptive Policy Management Scheme (APMS) converges to a stable state within a finite number of steps with a reasonable probability. In the worst case, the number of steps is $(|V| - 1)$, where $|V|$ is the number of nodes in the topology, and the probability is $(1/2)^{k+l+\cdots+i+((|V|-1)/2)}$, where $k, l, \ldots, i$ are positive integers.

Proof: By Lemma 4.3, we can show that APMS runs in a finite number of steps. Since Lemma 4.3 shows that the size of the nodes in the conflict-free set, S, monotonically increases, at each step of the algorithm, the size of the set of nodes yet to be explored, *i.e.* $\{V - S\}$, must be decreasing monotonically too. Since there are only $|V|$ nodes, after a finite number of steps, the algorithm converges with all nodes moving to the set S. However, this is true only if APMS performs path rank changes (*i.e. conflicting safe-alternative nodes* stick to their lower ranked but safe path) whenever a conflict is detected.

If the rank change is done probabilistically, then reaching a stable state in a finite number of steps will take longer. The worst case happens when each conflict in the system has only two *conflicting safe-alternative nodes*, and the conflicts in the system are nested and independent from each other as shown in Figure 9. Since the nodes use probability 1/2 for rank change, the worst-case probability of breaking a cycle with only 2 *conflicting safe-alternative nodes* happens when only one of such nodes performs rank change and this probability is 1/2. Once the rank change is performed and conflict is resolved, both nodes will be *conflict-free*. Since the expected number of tries for successfully realizing this rank change is 2, in the worst case, the average number of steps is $(|V| - 1)$.

Since the probability of breaking the current innermost conflict after j attempts is $(1/2)^j(1/2)$, in the worst case, the probability of reaching a conflict-free state is $((1/2)^k(1/2))((1/2)^l(1/2))\ldots((1/2)^i(1/2)) = (1/2)^{k+l+\cdots+i+((|V|-1)/2)}$, where $k, l, \ldots, i$ are positive integers.

However, such worst-case scenario is not very realistic. In more practical cases, breaking a conflict may simultanously resolve more than one conflict, or several independent conflicts may break simultaneously, which shortens the convergence time. ■

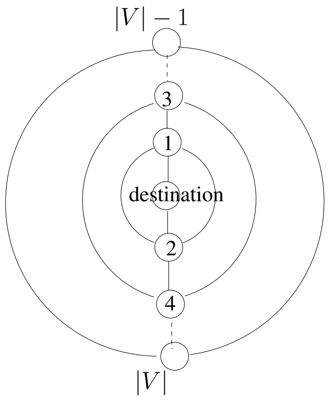

Fig. 9. Worst-case scenario for APMS when there is only 2 conflicting safe-alternative nodes in each conflict and the conflicts are nested and independent from each other. The depth is $(|V| - 1)/2$.

The analysis presented in this section does not consider adaptation and *recovery* of path rank changes. The next section (Section V) serves this purpose.

V. Simulation Results

We have simulated the algorithms in the SSFNet simulator [15]. We present two sets of results for two different topologies, which are presented in subsection V-A and subsection V-C, respectively. We first define our performance metrics:

- *The average of the percentage of paths that are eliminated per node at time t among permitted paths* to provide stability. The smaller value of this metric indicates better performance, since eliminating permitted paths (*i.e.* moving them to the *bad paths* set) may strain reachability, or force the router to choose a less preferred path to reach a destination.
- *The average of the percentage of the paths whose rank has been changed per node at time t*. Since changing the rank of the paths means changing locally configured policies that have been carefully placed for specific purposes, an algorithm causing a lot of rank changes would be undesirable.
- *The average of the percentage of the preference loss per node at time t among permitted paths*. Preference loss of a path is the difference between its original local preference and current preference value. If a path is placed in the bad path set, its preference loss is equal to its original local preference value. This metric helps us quantify the total effect of both path elimination and rank change.
- *The number of updates exchanged between routers* is an indication of stability. When the system is not stable,

the routers constantly exchange update messages. Therefore, smaller number of exchanged update messages reflects the efficiency of the protocol dealing with conflicts. To compute this metric, we have measured the number of updates carried in the last 2000 seconds, and averaged this value over 2000 seconds.

• *The number of bytes carried by update messages* is used to evaluate the overhead of the algorithms. Longer update messages takes longer to process and transmit. This overhead may negatively affect the overall performance of the system. We computed this metric by measuring the number of bytes carried in the last 2000 seconds, and averaged this value over 2000 seconds.

• *The average extra storage used at time t (in bytes)* is another metric for evaluating the overhead of the algorithms. For BGP4 [23], the value of this metric is always zero. For SPVP, $history$ is the newly added path attribute, and the main contributor of extra storage in routing tables, *i.e.* rib, rib_in and rib_out. The other source of extra storage is due to the bad path set since BGP4 does not have such set. The extra storage for Cobb and Musunuri algorithm is the per peer $cost$ kept at each node, which indicates the number of times a node has observed a route flap. For APMS, the size of *local history*, and *bad path set* are the main contributors of extra storage. We also added per peer $peerStability$ and $keepaliveCount$ as well as the list of $sequenceNumber$s that a node has learned during the execution of the algorithm, which are just integers. Depending on the occurrence of topology changes, the contribution of $sequenceNumber$s may be zero.

• *Power* is used to measure the ratio of throughput (total number of packets received in the last 50 seconds, and averaged over the last 50 seconds) and delay (the delay of the packets received, which is averaged over the total number of packets received in the last 50 seconds). The power metric captures the desire of achieving as high throughput as possible while keeping delay as small as possible.

• *Percentage of nodes that cannot reach the destination at time t* is also measured.

The performance plots presented next show 90% confidence intervals for these metrics.

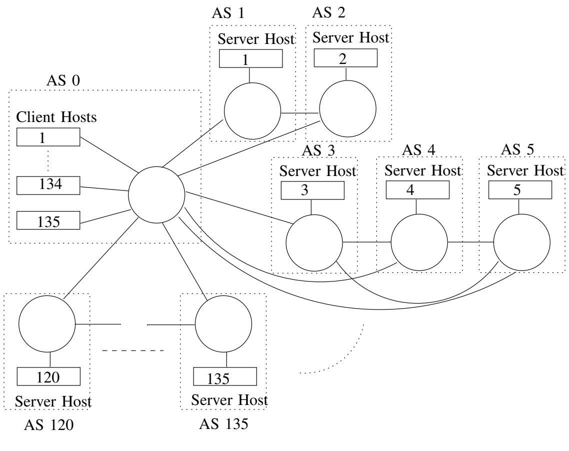

Fig. 10. Topology of Simulation Set I

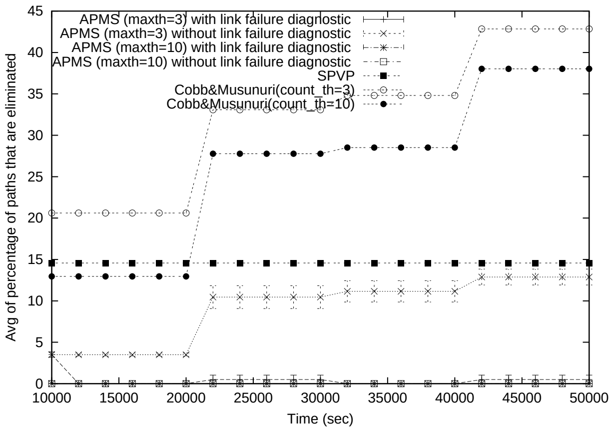

Fig. 11. Average Percentage of Paths Eliminated per Node

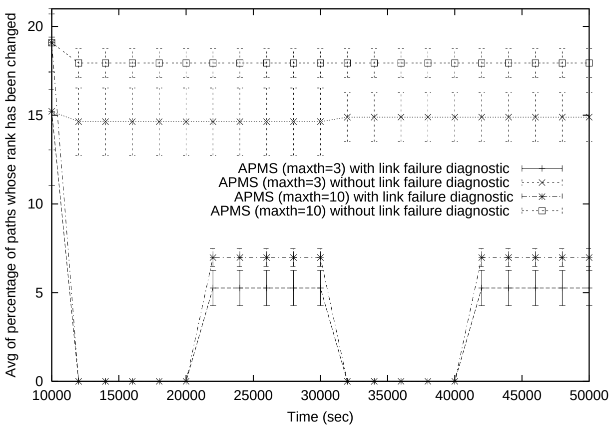

Fig. 12. Average Percentage of Paths Whose Rank Changed per Node

A. Simulation Set I

The topology is shown in Figure 10, and consists of 15 independent dispute wheels, where each AS in a dispute wheel has a direct connection to the destination AS 0. The destination AS has 135 client hosts, to whom there is constant data flow from the servers located in the other ASes. Each router within each AS has 3 permitted paths: The path through its clockwise neighbor, the direct path, and the path through its counter-clockwise neighbor. The policies are set to create policy conflicts, *i.e.* each AS prefers going through its clockwise neighbor rather than its direct path, which is preferred over going through the counter-clockwise neighbor—Local preference values assigned at each node are 100, 80, and 40, respectively.

Simulation is run for 50000 seconds, and data flow from servers to clients continues for the whole duration. We also introduced periodic link failures, during which all ASes lose their connection to the destination AS 0. After the system stabilizes at 10000 seconds, link failures are introduced. Recoveries and failures are then scheduled alternately every 10000 seconds.

The variations of APMS include using different values for $max\_threshold$ of 3 and 10, and whether route flaps due to topology change are distinguished. We set $min\_threshold$ to 2, and $ka\_threshold$ to 6. We have compared APMS

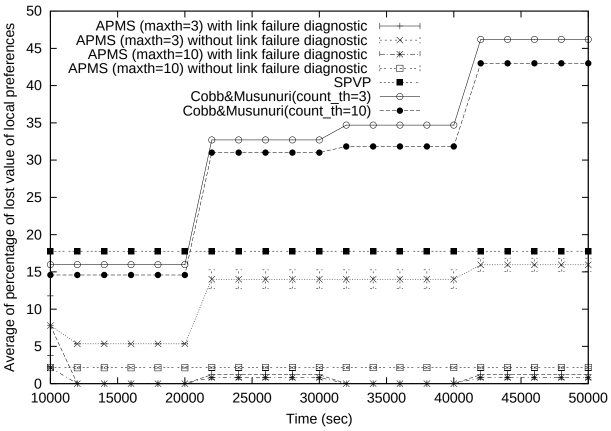

Fig. 13. Average Percentage of Lost Preference Value per Node

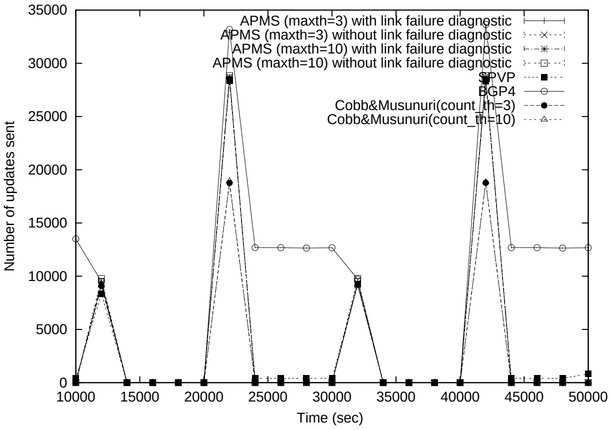

Fig. 14. Number of Updates Messages Sent

against the SPVP [7], Cobb and Musunuri algorithm [9], and BGP4 [23], where the details of these algorithms can be found in Section II-B. We have two versions of Cobb and Musunuri algorithm, where the threshold for cost is set to either 3 or 10 to be consistent with the values assigned to $max\_threshold$ for the variations of APMS. With SPVP, since there is no built-in mechanism to differentiate between transient oscillations and persistent oscillations due to policy conflicts, Griffin $et\ al.$ [7] suggest suppressing routes only after they are seen to contain some number of dispute cycles, or after the length of the history has exceeded some limit. In our simulations, we have used the first approach, and suppressed the routes only after seeing the same policy cycle twice. This is consistent with the value we have chosen for APMS, for which $min\_threshold$ is assigned a value of 2.

B. Results

The results for *the average of the percentage of paths that are eliminated per node at time t* is shown in Figure 11. BGP4 does not resolve conflicts, hence does not eliminate any paths. Therefore, the value of this metric is zero for BGP4 and not shown in the figure. In general, APMS resolves conflicts by means of path rank change instead of path elimination. APMS waits to see the same route flap $max\_threshold$ times before eliminating the path involved in the route flap. When $max\_threshold$ is larger, the system stays in the policy conflict-avoidance phase longer and tries harder to resolve conflicts through path rank change. When $max\_threshold$ is smaller, the system enters the policy conflict-control phase sooner, which causes elimination of more paths. With SPVP however, eliminating some of the flapping paths is its only way to deal with policy conflicts. Therefore, the performance of SPVP is worse than APMS for both small and larger values of the $max\_threshold$. With SPVP, on average each node eliminates 14.4% of the available paths. Cobb and Musunuri algorithm eliminates paths advertised by the peer whose cost has reached the threshold, *i.e.* 3 or 10 for this experiment. Both versions of Cobb and Musunuri algorithm eliminate most of the paths, upto 42.84% (for threshold=3) and 38.02% (for threshold=10), because of both aggregation of the paths through the high cost nodes and absence of any mechanism to distinguish route flaps due to topology change (false positives).

The version of APMS lacking any topology change diagnosis and using a smaller value of $max\_threshold$, 3, keeps eliminating the available paths since there is no mechanism to differentiate route flaps due to failure and/or recovery of the links from those triggered by policy conflicts. In this case, paths seem to be flapping more often, and $count$ values increase faster. For the same value of $max\_threshold$, adding topology change diagnosis to the algorithm provides big improvement: On average each node eliminates only 0.48% of the available paths to reach stability during non-fail periods. When $max\_threshold$ is larger, resolving conflicts by means of path rank change dominates that by path elimination. As we can see, this leads to minimal path elimination for both versions of APMS; with and without topology change diagnosis. However, this result remains valid for APMS without topology change diagnosis only when the $max\_threshold$ value is larger than the number of route flaps resulting from topology change. Otherwise, there will be false positives, and more path elimination.

For the previous metric, we have seen that for large enough value of $max\_threshold$, APMS can avoid eliminating paths performing path rank changes to resolve policy conflicts. However for APMS, we would like to see whether the mechanism is causing a lot of path rank changes, and hence significantly altering the policies that have been carefully placed for specific purposes. Figure 12 shows the results for *the average of the percentage of the paths whose rank has been changed per node*. Since APMS is the only algorithm that changes the policies to stabilize the system, for SPVP, Cobb and Musunuri algorithm and BGP4, the value of this metric is 0 and therefore not shown in the figure. For larger values of $max\_threshold$, we observe higher number of path rank changes due to longer policy conflict-avoidance phase. Using topology change diagnosis improves performance in the presence of failure and recovery of the links, and drops the metric value from 18% to 7.0% for non-fail periods. This corresponds to changes in the rank of only about 2 paths out of about 20 available paths per dispute wheel on average (plots not shown for lack of space), without eliminating a single path. Smaller value of $max\_threshold$ shows lower percentage of path rank change due to shorter policy conflict-avoidance phase. However, as we have seen in Figure 11, this version of the algorithm eliminates more paths to deal with conflicts.

The results for *the average of the percentage of the preference loss per node* is shown in Figure 13. Since BGP4 has no mechanism to deal with conflicts, there is no loss in terms of preference value, and hence BGP4 is not shown in the figure. SPVP causes loss of around 18%, which is contributed only by eliminated paths. Cobb and Musunuri algorithm causes loss of 46.2% for cost threshold 3, and 42.0% for cost threshold 10, which are contributed only by eliminated paths. With APMS, the performance is always better than SPVP and the Cobb and Musunuri algorithm. Using larger values of $max\_threshold$, 10, and topology change diagnosis significantly improves performance to less than 1% loss in path preference.

Figure 14 presents the results for *the number of update messages sent* for each 2000 interval of time. Topology changes in the form of link failure or restoration cause a burst of updates, and the burst is smaller in the case of link failures due to limited reachability. When each node loses its reach to the destination, Hold Timer expires, and all of the paths to the destination become infeasible and are withdrawn. When the links are restored, BGP sessions are re-established, and the whole routing tables are exchanged. The biggest routing table is of node 0, since it has a BGP session with every other node, and this table contributes a very big portion of the high peaks in the graph right after the link restorations (at times 20000, 40000, $\cdots$). For topology changes in the form of link failures, we observe a smaller burst of updates due to restricted reachability: After the first 2000 seconds of each fail period, the number of updates sent is zero for all algorithms. This is because the new topology does not have any policy conflicts, and therefore it stabilizes independent of the algorithm used. For non-fail periods, since with BGP4 the system does not stabilize, the number of updates sent under BGP4 does not get close to zero. SPVP and APMS show very close performance regarding this metric, and the number of updates sent is much smaller than BGP4. Since reachability is the most restricted with both versions of Cobb and Musunuri algorithm (see Figure 11), they have the lowest value for this metric.

To evaluate of the overhead of the algorithms, *the number of bytes carried by update messages* is shown in Figure 15. In Figure 14, we have seen that with SPVP the total number of updates exchanged was much smaller than BGP4, and very close to APMS. However, the divergence detection mechanism of SPVP requires carrying the sequence of path change events in each update, *i.e. history*. Thus, SPVP has the highest number of bytes carried by its update messages. All versions of APMS show very close performance, and all are better than both BGP4 and SPVP. When topology change diagnosis is deployed in APMS, the update messages carry some extra information in the $originator$ and $sequenceNumber$ fields (6 bytes in total). However, as we can observe, the mechanism used by APMS to distinguish temporary oscillations (due to topology/policy changes) is much more efficient than the SPVP mechanism of observing repeated cycles in the history. Although the only

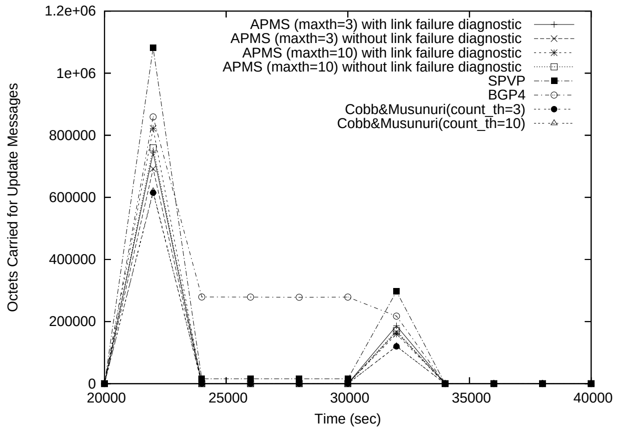

Fig. 15. Number of Bytes Carried by Update Messages

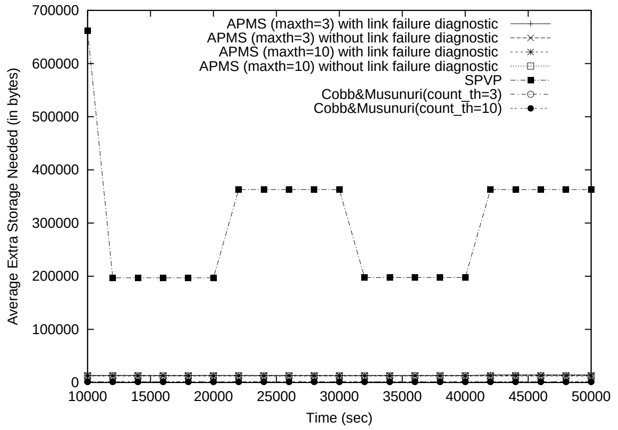

Fig. 16. Average Extra Storage Needed at Each Node by the Algorithms (in bytes)

extra information carried with update messages in the Cobb and Musunuri algorithm is $cost$ associated with each node, this is not the only reason for its best performance. Cobb and Musunuri algorithm has the lowest value because of the least number of exchanged update messages due to limited reachability (see Figure 14 and Figure 11), which is a result of both aggregation (and elimination) of all paths through high cost nodes and absence of any mechanism to distinguish route flaps due to topology change (false positives).

Figure 16 shows the results for *the average extra storage used at time t (in bytes)*. Due to the size of $history$, the storage required by SPVP is much larger than that required by APMS and the Cobb and Musunuri algorithm. For SPVP, the value of the metric is higher for non-fail periods than fail periods due to better reachability during the non-fail periods. For this experiment, while APMS requires around 10KB extra storage, SPVP requires 200KB-360KB extra storage. APMS requires a little bit more extra storage than the Cobb and Musunuri algorithm due to bigger local state kept at each node.

C. Simulation Set II

To be able to observe $throughput$ and $delay$ better, in the second setup we have used a smaller topology shown in Figure 17. The topology consists of 7 dispute wheels, where each

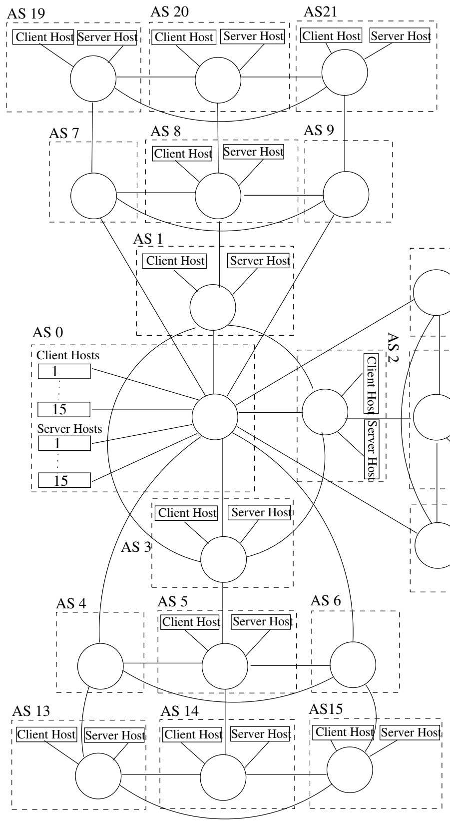

Fig. 17. Topology Used for Simulation Set II

Node	Permitted Paths	Local Preference
1	(1 2 0)	100
	(1 0)	80
	paths learned from 8	1
2	(2 3 0)	100
	(2 0)	80
	paths learned from 11	1
3	(3 1 0)	100
	(3 0)	80
	paths learned from 5	1
4	(4 6 0)	100
	(4 0)	80
	paths learned from 13	1
5	(5 4 0)	100
	(5 3 0)	80
	(5 3 1 0)	80
	paths learned from 14	1
6	(6 5 3 0)	100
	(6 5 3 1 0)	100
	(6 0)	80
	paths learned from 15	1
7	(7 8 1 2 0)	100
	(7 8 1 0)	100
	(7 0)	80
	paths learned from 19	1
8	(8 9 0)	100
	(8 1 0)	80
	(8 1 2 0)	80
	paths learned from 20	1
9	(9 7 0)	100
	(9 0)	80
	paths learned from 21	1
10	(10 11 2 0)	100
	(10 11 2 3 0)	100
	(10 0)	80
	paths learned from 16	1
11	(11 12 0)	100
	(11 2 0)	80
	(11 2 3 0)	80
	paths learned from 17	1
12	(12 10 0)	100
	(12 0)	80
	paths learned from 18	1

Node	Permitted Paths	Local Preference
13	(13 15 6 0)	100
	(13 15 6 5 3 0)	100
	(13 15 6 5 3 1 0)	100
	(13 4 0)	80
	(13 4 6 0)	80
14	(14 13 4 0)	100
	(14 13 4 6 0)	100
	(14 5 3 0)	80
	(14 5 3 1 0)	80
	(14 5 4 0)	80
15	(15 14 5 3 0)	100
	(15 14 5 3 1 0)	100
	(15 14 5 4 0)	100
	(15 6 0)	80
	(15 6 5 3 0)	80
	(15 6 5 3 1 0)	80
16	(16 17 11 12 0)	100
	(16 17 11 2 0)	100
	(16 17 11 2 3 0)	100
	(16 10 0)	80
	(16 10 11 2 0)	80
	(16 10 11 2 3 0)	80
17	(17 18 12 0)	100
	(17 18 12 10 0)	100
	(17 11 2 0)	80
	(17 11 2 3 0)	80
	(17 11 12 0)	80
18	(18 16 10 0)	100
	(18 16 10 11 2 0)	100
	(18 16 11 2 3 0)	100
	(18 12 10 0)	80
	(18 12 0)	80
19	(19 20 8 9 0)	100
	(19 20 8 1 0)	100
	(19 20 8 1 2 0)	100
	(19 7 0)	80
	(19 7 8 1 2 0)	80
	(19 7 8 1 0)	80
20	(20 21 9 0)	100
	(20 21 9 7 0)	100
	(20 8 9 0)	80
	(20 8 1 0)	80
	(20 8 1 2 0)	80
21	(21 19 7 0)	100
	(21 19 7 8 1 2 0)	100
	(21 19 7 8 1 0)	100
	(21 9 0)	80
	(21 9 7 0)	80

Fig. 18. Path Rankings for Topology Used in Simulation Set II

AS in a dispute wheel prefers the path through its clockwise neighbor. The wheels consist of the following group of ASes: {AS 1, AS 2, AS 3}, {AS 4, AS 5, AS 6}, {AS 7, AS 8, AS 9}, {AS 10, AS 11, AS 12}, {AS 13, AS 14, AS 15}, {AS 16, AS 17, AS 18}, {AS 19, AS 20, AS 21}. Permitted paths and path preferences are shown in Figure 18. Simulation is run for 350

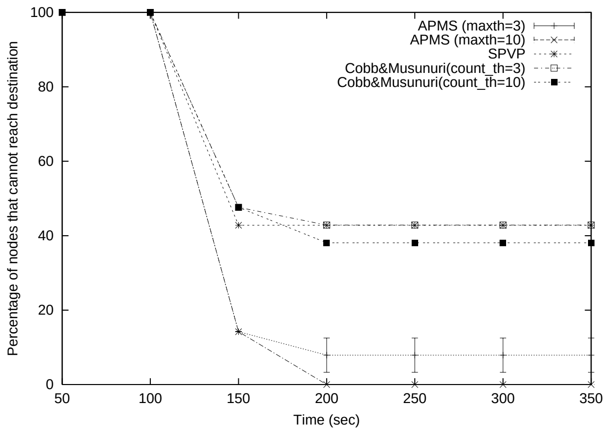

Fig. 19. Percentage of nodes that cannot reach the destination

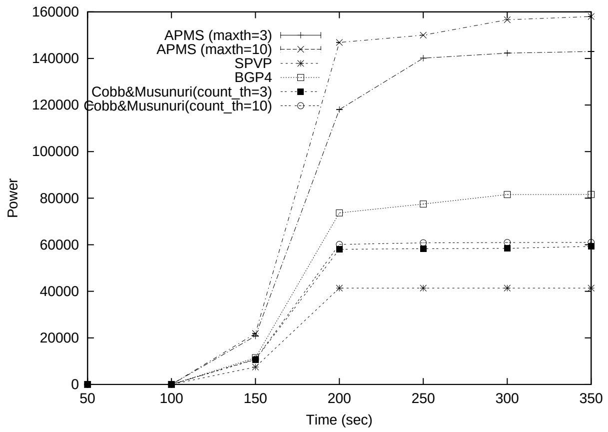

Fig. 20. Power

seconds, and data flow from servers to clients continues for the whole duration. For this experiment set, there is no topology change. Buffer size is 50000 bytes, and routing packets are given priority over data packets when there is congestion at the buffers. Our findings are presented in Section V-D.

D. Results

Figure 19 shows the percentage of the nodes that cannot reach the destination AS 0. SPVP and the Cobb and Musunuri algorithms eliminate a high number of paths while enforcing stability, and therefore leave a higher number of nodes with unreachable destination. Different versions of APMS perform much better than SPVP and the Cobb and Musunuri algorithms. With APMS, higher $max\_threashold$ value helps forcing conflict resolution through changing path preferences, and therefore minimizing the number of path eliminations. For $max\_threshold$=10, the system stabilizes to a state where each node has a way to reach the destination. Higher threshold value also helps the Cobb and Musunuri algorithm achieve better performance, but the improvement is not much because of the simultaneous elimination of the paths through the same high cost node.

Figure 20 shows the results for *power*. Different versions of APMS have higher value than SPVP and the Cobb and Musunuri algorithms for this metric because APMS maximizes the throughput. The different performance for throughput stems from both unreachable destinations, and/or competition for the limited buffer size. SPVP and the Cobb and Musunuri algorithms leave a higher number of nodes with unreachable destination (Figure 19). SPVP has the longest update messages, which take longer to process and require more memory to be stored in the buffers. Although BGP4 causes constant exchange of updates due to divergence, its performance is better than SPVP and the Cobb and Musunuri algorithms! This is because BGP4 does not cause permanent path elimination, even though some packets may not reach the destination temporarily due to instability.

VI. Summary and Future Work

Unlike static solutions (e.g. Gao *et al.* [6] algorithm) which may lead to unnecessary disallowance of the usage of many routes from the start to guarantee the stability of the system, APMS is a dynamic algorithm, and allows ASes to adapt to the current state of the network, either conflict-free or potentially conflicting. APMS makes minimal changes to local policies as the primary means of removing policy conflicts (and associated routing oscillations). Path elimination with APMS happens only if probabilistic rank change of paths does not resolve the conflict [3]. APMS attempts to keep as many paths as possible to have better connectivity, and more flexibility in path selection for the stabilized system. APMS is distributed, and does not require a global authority or global database.

Compared to other dynamic algorithms (e.g. SPVP [7] and Cobb and Musunuri [9]), APMS has several advantages: (1) APMS eliminates the need to carry possibly large amount of information like *history* in the update messages. Instead, APMS uses local state information. Therefore, with APMS there is no communication overhead, nor any concerns about revealing private information about the preferences of ASes over their routes; (2) APMS minimizes path elimination by path rank changes and by adapting to a conflict-free state by (conservatively) restoring some of the original preferences as well as eliminated paths; (3) APMS has a more effective mechanism for clearly distinguishing route flaps due to topology change, which helps minimize false positives and communication overhead; (4) APMS automatically adapts to the dynamics of the system by observing either path changes or keepalive messages without requiring an expensive protocol, such as diffusing computation [9]; and (5) APMS deals with the problem at the path level instead of node level, which prevents aggregation of paths through the same node, and hence elimination of many paths whose preference does not cause conflicts.

Route flap damping [20] is not an alternative solution to APMS, and only effective in suppressing temporary instabilities. APMS is still needed on top of route flap damping to detect and resolve policy conflicts leading to divergence and/or adapt to dynamically changing network state. Usage of route flap damping may delay diagnosing persistent oscillations via APMS due to suppressed updates. If there is a topology change, APMS needs to be aware of this change so it handles the resulting route flaps differently from the flaps resulting from policy conflicts until the system adapts to the new network state. However, there maybe no need to react and adapt to every topology change. For example, an unstable link may go down and up constantly for a period of time. In such cases, it is better that APMS does not react to such short-term changing link status. APMS may take route flap damping into consideration to hide such shorter term topology changes from other ASes.

If only some of the nodes in the network upgraded to deploy APMS, APMS still can catch and resolve policy conflicts since the algorithm is based on only local information kept at each node. However, in such heterogenous settings, the nodes deploying APMS will be the only ones which may give up their preferred paths for the sake of the network stability without knowing whether or not the other nodes are working for the same purpose. As future work, to be able to prevent selfish behavior, and improve cooperation among ASes to deploy APMS, incentives should be proposed.

To increase transparency of APMS, we plan to investigate allowing local AS administrators to explicitly guide the backoff and recovery probabilities for lowering and restoring the rank of paths. With such a human impact, it may be possible to resolve policy conflicts in a more efficient way albeit at a longer time scale.

APMS is explained using an abstract model of BGP (in Section III). Although the implementation of APMS in SSFNet relaxes some of these abstract model assumptions (Section V), there are still many details to be addressed such as IGP/BGP interactions, duplicate updates, session resets etc. These details are left for future work.

[3] Since the approach is probabilistic, there is a non-zero probability that none of the nodes change the rank of their paths.

References

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

[2] C. Labovitz, G. Malan, and F. Jahanian, "Internet Routing Instability," *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 515–528, 1997.

[3] R. Govindan and A. Reddy, "An Analysis of Interdomain Routing Topology and Route Stability," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, Kobe Japan, April 1997.

[4] V. Paxson, "End-to-end Routing Behavior in the Internet," IEEE/ACM Transactions on Networking, Vol.9, No.4, pp. 392-403, August 2001.

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

[6] L. Gao and J. Rexford, "Stable Internet Routing without Global Coordination," in *Proceedings of ACM SIGMETRICS*, Santa Clara CA, June 2000.

[7] T. Griffin and G. Wilfong, "A Safe Path Vector Protocol," in *Proceedings of IEEE INFOCOM*, Tel Aviv Israel, March 2000.

[8] J. A. Cobb, M. G. Gouda, and R. Musunuri, "A Stabilizing Solution to the Stable Paths Problem," *Symposium on Self-Stabilizing Systems, Springer Verlag Lecture Notes in Computer Science*, vol. 2704, pp. 169–183, 2003.
[9] J. A. Cobb and R. Musunuri, "Enforcing Convergence in Inter-Domain Routing," in *Proceedings of IEEE Global Communications (GLOBECOM) Conference*, Dallas TX, December 2004.
[10] T. Griffin, A. Jaggard, and V. Ramachandran, "Design Principles of Policy Languages for Path Vector Protocols," in *Proceedings of ACM SIGCOMM*, August 2003.
[11] A. Jaggard and V. Ramachandran, "Robustness of Class-based Path-Vector Systems," in *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, March 2004.
[12] J. Sobrihnho, "Network Routing with Path Vector Protocols: Theory and Applications," in *Proceedings of ACM SIGCOMM*, Karlsruhe Germany, August 2003.
[13] S. Yilmaz and I. Matta, "A Randomized Solution to BGP Divergence," in *Proceedings of the 2nd IASTED International Conference on Communication and Computer Networks (CCN'04)*, Cambridge MA, November 2004.
[14] V. Jacobson, "Congestion Avoidance and Control," in *ACM SIGCOMM '88*, Stanford CA, August 1988, pp. 314–329.
[15] SSFNet: Scalable Simulation Framework, "http://www.ssfnet.org."
[16] T. Griffin, F. Shepherd, and G. Wilfong, "Policy Disputes in Path-Vector Protocols," in *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, Toronto Canada, November 1999.
[17] T. Griffin and G. Wilfong, "An Analysis of BGP Convergence Properties," in *Proceedings of ACM SIGCOMM*, Cambridge MA, September 1999.
[18] L. Gao, T. Griffin, and J. Rexford, "Inherently Safe Backup Routing with BGP," in *Proceedings of IEEE INFOCOM*, Anchorage AK, April 2001.
[19] N. Feamster, H. Balakrishnan, and J. Rexford, "Some Foundational Problems in Interdomain Routing," in *ACM SIGCOMM Workshop on Hot Topics in Networking (HOTNets-III)*, San Diego CA, November 2004.
[20] C. Villamizar, R. Chandra, and R. Govindan, "BGP Route Flap Damping," RFC 2439, 1998.
[21] A. Arora and M. Gouda, "Distributed Reset," *IEEE Transactions on Computers*, vol. 43, pp. 1026–1038, 1994.
[22] J. Feigenbaum, R. Sami, and S. Shenker, "Mechanism Design for Policy Routing," in *Proceedings of the Symposium on Principles of Distributed Computing (PODC)*, Newfoundland Canada, July 2004.
[23] Y. Rekhter and T. Li, "A Border Gateway Protocol," RFC 1771, 1995.