

An Adaptive Policy Management Approach to BGP Convergence

SELMA YILMAZ

IBRAHIM MATTA

Computer Science Department
Boston University
Boston, MA 02215, USA

{selma,matta}@cs.bu.edu

Technical Report BUCS-TR-2005-028
July 6, 2005

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 adopt a set of local policies that specify which routes it accepts and advertises from/to other networks, as well as 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 (policy conflict-avoidance vs. -control mode), 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, routing load, etc.) of APMS and other competing solutions, thus exposing the often neglected aspects of performance.

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 where distance-based metrics are overridden by policy-based metrics. 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 p_1 and p_2 , where p_1 is preferred over p_2 . If extensions of p_1 and p_2 , i.e. $(u v)p_1$ and $(u v)p_2$, are permitted at AS u , and $(u v)p_2$ is preferred over $(u v)p_1$, then there is a policy conflict. When AS v improves its best

path from p_2 to p_1 , 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.

Although persistent oscillations due to policy conflicts have not been observed so far in the current Internet, it is strongly believed that as the commercial infrastructure of the Internet continues to grow, so does the potential for developing persistent route oscillation because of the expectation of parallel growth of policies both in size and complexity [20], [15].

Several studies [9], [15], [16] 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 [8], [7], [13], [2], [3], [10], [14], [18]. However, some of these solutions are static, and dynamic solutions have high overhead. To overcome the shortcomings of the current solutions, in our previous work [21], 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 [21] 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 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; (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 SSF network simulator [19] 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.

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* [13], which is called *Simple 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 $P = \langle u, v, w, \dots, \text{destination} \rangle$ only if the current path at node v is $\langle v, w, \dots, \text{destination} \rangle$ 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 $V = \{0, 1, \dots, n\}$ 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 $\text{peers}(u) = \{w | \{u, w\} \in E\}$. 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 $(v_k, v_{k-1}, \dots, v_1, v_0)$, 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 $P = (v_1, v_2, \dots, v_k)$ and $Q = (w_1, w_2, \dots, w_m)$ can be *concatenated* as follows $PQ = (v_1, v_2, \dots, v_k, w_2, \dots, w_m)$ 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 , $\text{max}(u, W)$, 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

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

The path assignment π is called *stable at node* u if

$$\pi(u) = \text{max}(u, \text{choices}(u, \pi)).$$

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, \dots, 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: $\text{rib}(u)$ is the current path that node u is using to reach the *destination*, and $\text{rib_in}(u \leftarrow w)$ 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

$$\text{choices}(u) = \{(u, w)\text{rib_in}(u \leftarrow w) | w \in \text{peers}(u)\} \cap \mathcal{P}^u$$

and the best path at u is

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

As long as node u receives advertisements from its peers, $best(u)$ is recomputed with the most recent $choices(u)$, which is 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 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* [11] 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, \dots, u_k , and the set of paths Q_1, Q_2, \dots, Q_k , and R_1, R_2, \dots, 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) $R_i Q_{i+1}$ is a permitted path at u_i ($Q_1 = Q_{k+1}$); (4) Q_i is less preferred than $R_i Q_{i+1}$ at node u_i .

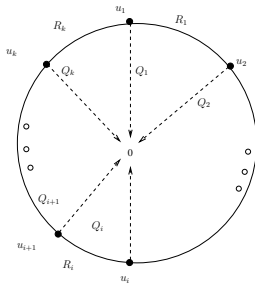


Fig. 1. Dispute Wheel of Size k .

The dispute wheel of size k is shown in Figure 1. Q_i s 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 path R_i s are called the *rim* of the wheel, and the nodes on the rim of the wheel are called *rim nodes*. Each rim is also a simple (loop-free) path. The rim nodes at the ends of the path R_i s are called *active nodes*. The active nodes are the nodes at which route preferences cause the dispute wheel.

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* [20]. Since then, many studies proposed approaches to guarantee the safety, *i.e.* convergence, of BGP independent of the locally selected policies [8], [7], [13], [2], [3], [21], [10], [14], [18]. These approaches can be broadly classified into static and dynamic solutions.

Static solutions: Govindan *et al* [8] 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* [12] 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 [8], Gao *et al* [7], [6] propose another static solution which restricts the routing policy 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* [4] argue that there maybe legitimate reasons to deviate from the guidelines proposed in [7].

Other static solutions [18], [14], [10] suggest different constraints that also prevent policy based oscillations in advance.

Dynamic Solutions: Griffin and Wilfong [13] suggest a dynamic mechanism to detect and suppress BGP oscillations that arise because of policy conflicts. The idea is to extend BGP to carry additional information called *history* 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 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 a very big communication overhead. The algorithm 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 hence it is persistent. However, this approach increases communication overhead even more due to even longer update messages. This way of differentiating persistent and transient oscillations also increases convergence time. On the other hand, resolving loops sooner may lead to false positives, and unnecessary path eliminations. Griffin and Wilfong algorithm 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. Furthermore, *histories* are carried across ASes, and may reveal private information about the preferences of ASes over the routes.

Recently, Cobb *et al* [2], [3] propose two dynamic algorithms. The first work [2] 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, thus the overhead may become very big.

In their second work, Cobb and Musunuri [3] 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. Elimination of so many paths for a conflict is obviously more than necessary. The other disadvantage of the approach 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. Furthermore, this approach [3] cannot reduce the cost of the nodes easily. Resetting the cost of the nodes is essential when (1) there is a topology change, and (2) there are nodes whose cost has reached the threshold. 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 [1], 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 [3]. 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 [5]. For the case where AS policies are restricted to the hierarchical relationships, Feigenbaum *et al* [5] 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. [5] 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 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 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 lower 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.

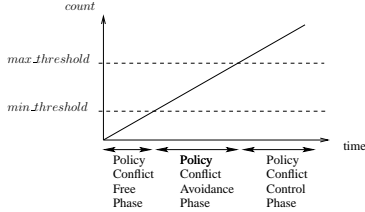


Fig. 2. Three Phases of Adaptive Policy Management Scheme.

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.

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 $peerStability$ equals 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.

- 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 resets some of the local preferences back to their original values¹. 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

$$choices_B(u) = \{(u, w)rib_in(u \leftarrow w) - B(u) | w \in peers(u)\} \cap \mathcal{P}^u$$

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 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

¹This is akin to increase/decrease adaptation rules employed in many adaptive feedback-control systems.

that $\text{rank}(P_{safe}) = 1$. If there are more than one safe path, then the most preferred one is chosen as the highest ranked path.

```

process APMS.Update_Handling[u]      //Update Handling
receive Update m from peer w  →
  if (rib.in(u ← w) ≠ m
    peerStability(w)++
    rib.in(u ← w) = m
  if rib(u) ≠ bestB(u) then
    Pold = rib(u)
    Pnew = bestB(u)
    if (Pnew ≠ Pold) and (Pnew ≠ ε) then
      count(Pnew)++
    if count(Pnew) ≥ max.threshold then
      B(u) = B(u) ∪ {Pnew}           //Policy Conflict Control
      Pnew = bestB(u)
      count(Q)=0 for each path Q ∈ localHistory
      peerStability(v)=0 for each v ∈ peers(u)
    else if count(Pnew) ≥ min.threshold then
      do with probability= 1/2
        find the most preferred safe path, Psafe //Policy Conflict Avoidance
        rank(Psafe)=1
        Pnew = Psafe
        count(Q)=0 for each path Q ∈ localHistory
        peerStability(v)=0 for each v ∈ peers(u)
  if Pnew ≠ Pold then
    rib(u) = Pnew
    for each v ∈ peers(u) do
      send rib(u) to v

```

Note: The code to the right of the →
is assumed to be executed in one atomic step .

Fig. 3. Adaptive Policy Management Scheme: Update Handling.

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 approach, and risks introducing instability back into the system. However, contrary to update handling, node u increases the local preference of a path with a smaller probability, 1/4.² We also allow for bringing paths out of $B(u)$ 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.

²This akin to a slower (conservative) probing of the current state in adaptive feedback control systems, e.g. the congestion control mechanism of TCP.

```

process APMS.Keepalive_Handling[u] //Keepalive Handling
receive keepalive from 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
          Pnew = bestB(u)
          if Pnew ≠ rib(u) then
            rib(u) = Pnew
        for each v ∈ peers(u) do
          send rib(u) to v

```

Note: The code to the right of the →
is assumed to be executed one atomic step .

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

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 destination is node v , i.e. $P = \langle u, v, \dots, destination \rangle$. If the link between u and v goes down, node u won't be able to get KEEPALIVE messages from v for a period of time, which is specified by the Hold Time value. At the end of this period, node u gets a NOTIFICATION message with Hold Timer Expired Error Code, and the session with node v ends. Node u 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 when node u sends an update message about this change, it includes some kind of information about the failure too. Then the other nodes which are not in the neighborhood of the failed link can use this information to recognize that the update was triggered because of a failure. Note that from node u 's perspective, the effects of the failure of node v is the same as the failure of the link between u and v . Therefore, failure of a peer is handled the same way as failure of the link between them. Link/node restorations can be handled in a similar way by observing OPEN messages exchanged when a peering TCP session is (re-)established.

To be able to realize the idea presented above, we suggest adding two new fields to the UPDATE messages of BGP:

- *topologyChange* is a bit that indicates if the node got an update originating from a topology change. Topology change can be a link failure, failure of a node, or recovery of a failed

link or a node. When a node receives an UPDATE message with the *topologyChange* field set, it temporarily turns off the policy conflict detection process, and does not process the path change resulting from this update for policy conflict detection. It also resets the local state, which reflects the dynamics of the previous topology.

- *originator* field is a list of nodes that have already learned of a particular topology change, and processed it. If node u gets an UPDATE message with the *topologyChange* field set, and node u is not listed in the *originator* field, the policy conflict detection process is turned off temporarily, and the resulting path change is not processed for policy conflict detection. Node u resets the local state, which reflects the dynamics of the previous topology, and updates the *originator* field by adding its AS number to the field before sending the UPDATE message triggered by this advertisement. This UPDATE message is sent with the *topologyChange* field set as well. If node u gets an UPDATE message with the *topologyChange* field set, and node u is already listed in the *originator* field, then node u deduces that the UPDATE message is due to a topology change, to which node u has already reacted and adapted. Therefore, node u processes the path change resulting from this update for policy conflict detection and sends the resulting UPDATE message with *topologyChange*=0, and *originator*={}.

With this addition of handling topology changes, the messages exchanged between peers are no longer paths only. A message m is a triple $(P, \textit{topologyChange}, \textit{originator})$. The pseudo-code of this topology handling algorithm is shown in Figure 5. While this mechanism handles transient oscillations due to topology changes, *min.threshold* still helps not to react too soon to transient oscillations arising from other causes such as a change in policy, change in the next-hop, etc.

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 $P = (v, \dots, \textit{destination})$ 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 neighbor of u , and v is advertising the non-flapping path $P = (v, \dots, \textit{destination})$.

Definition 4.4: Conflicting safe-alternative node is a node which is involved in a policy conflict, and has an alternative safe path.

Definition 4.5: Conflicting node is a node which is involved in a policy conflict, and does not have a safe path. If a group of nodes are involved in a cyclic conflict, then none of the nodes in the group can be a *conflict-free* node. Such

nodes are either *conflicting safe-alternative node* or *conflicting node*. If node u 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 neighbors stabilizes, it will start to observe stable path advertisements coming from this newly stabilized neighbor. At this point, node u is no longer a *conflicting node*, but either a *conflicting safe-alternative node*, or a *conflict-free node* depending on the number of permitted paths at node u , the neighbors advertising these paths and the conflicts they are involved in.

For the following discussion, we assume that there is a

```

process APMS.TopologyChange.Handling[u]           //Handling topology changes
//when a node 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 ← v)
    if r is permissible
      feasible(r)=false
  if link (u, v) is restored or node v is restored
    receiveUpdate m from v
    rib.in(u ← v) = path(m)
  count(Q)=0 for each path Q ∈ localHistory
  B(u)={}
  peerStability(w)= 0 for every w ∈ peers(u)
  for every w ∈ peers(u)
    p ∈ rib.in (u ← w)
    if p is permissible
      if localpref(p) ≠ originallocalpref(w)
        localpref(p)=originallocalpref(w)
  rib(u)=bestB(u)
  for every w ∈ peers(u)
    send update message m
    where path(m)=rib(u), topologyChange(m)=1, originator(m)={u}

//when node u gets an update message m from node v with topologyChange=1
if u ∈ originator(m)
  //process this update as shown in " APMS_Update.Handling[u]"
  if there is a new update to be sent
    set topologyChange=0, originator={} before sending it to the peers
else
  count(Q)=0 for each path Q ∈ localHistory
  B(u)={}
  peerStability(w)= 0 for every w ∈ peers(u)
  for every w ∈ peers(u)
    p ∈ rib.in (u ← w)
    if p is permissible
      if localpref(p) ≠ originallocalpref(w)
        localpref(p)=originallocalpref(w)
  rib(u)=bestB(u)
  for every w ∈ peers(u)
    send update message n
    where path(n)=rib(u), topologyChange(n)=1, originator(n)=m.originator ∪ {u}

```

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

single cyclic conflict in the system. Let N denote the set of nodes that are in this conflict, where $|N| \geq 2$. We would like to show that there must be some *conflicting safe-alternative nodes* in N for such conflict to be realized. 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. Therefore, 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. Only after this happens, the *conflicting nodes*

will be able to stabilize on their paths.

There maybe two types of persistent oscillations: *Type 1* is persistent oscillation due to two different solutions, *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 has a safe path.

Proof: We prove this by contradiction: Assume policies of a group of nodes lead to a persistent oscillation, and only one of the nodes in this group has a safe path. Let u_1 denote the only *conflicting safe-alternative node*, and p_1 denote its safe path. Since we have assumed that this is the only conflict in the system, and none of the other nodes has a safe path, their next-hops can only be the nodes involved in the conflict. Therefore, they must reach the destination through u_1 . p_1 cannot include any other nodes in the dispute, otherwise it wouldn't be safe. For the presence of cyclic dependency to create a 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*, i.e. no repeated nodes, and non-simple paths are not permitted by BGP. ■

Theorem 4.1: Under BGP, exactly 2 nodes have safe paths in a persistent oscillation iff the persistent oscillation is a type 1 oscillation.

Proof: *If exactly 2 nodes have safe paths in a persistent oscillation, then the persistent oscillation is a type 1 oscillation:* For a persistent oscillation to occur there must be nodes whose policies conflict in a cyclic manner. Let u_1 and u_k be the only *conflicting safe-alternative nodes* in this dispute. Then these 2 nodes must prefer going through each other, and the other *conflicting* nodes should carry this desire in opposite directions. The two possible stable solutions involve either u_1 or u_k choosing its safe path. (An example is shown in Figure 6. The permitted paths at node u_1 are Q_1 , which is the safe path, and $u_1u_2 \dots u_{k-1}u_kQ_k$, which is the more preferred path. The permitted paths at node u_k are Q_k , which is the safe path, and $u_ku_{k+1} \dots u_nu_1Q_1$, which is the more preferred path. In this case the two possible stable solutions are: u_1 chooses Q_1 and u_k chooses $u_ku_{k+1} \dots u_nu_1Q_1$, or u_1 chooses $u_1u_2 \dots u_{k-1}u_kQ_k$ and u_k chooses Q_k .)

If a persistent oscillation is a type 1 oscillation, then exactly 2 nodes must be conflicting safe-alternative nodes: Since the oscillation is a type 1 oscillation, there are 2 different stable solutions. Having two different stable solutions is not possible with only one *conflicting safe-alternative node*, because in such case there can only be one stable solution. Having two different stable solutions is not possible with three *conflicting safe-alternative nodes*, because for this case there is not even a single stable solution (An example is shown in Figure 7). The case for more than 3 *conflicting safe-alternative node* is similar and has no stable solution. ■

Theorem 4.2: Under BGP, at least 3 nodes have safe paths 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 have safe paths:* For type 2

oscillation, there is no solution. If only 1 node has a safe path, there is no persistent oscillation as shown in Lemma 4.1. If only 2 nodes have safe paths, there is type 1 oscillation as shown in Theorem 4.1. Therefore, there must be 3 or more nodes with safe paths for type 2 oscillation to occur.

If 3 nodes have safe paths in a persistent oscillation, then the persistent oscillation is a type 2 oscillation: There cannot be a stable path assignment with 3 nodes whose policies are conflicting. Therefore, it is type 2 oscillation. ■

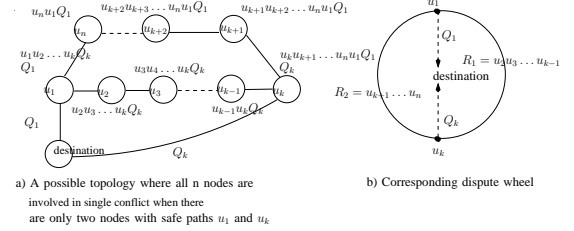


Fig. 6. An Example Topology and Corresponding Dispute Wheel.

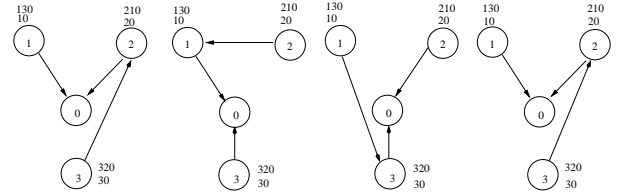


Fig. 7. A cyclic policy conflict with 3 conflicting safe-alternative nodes: There is no stable solution. The routing tree cycles back to the same configuration. The safe paths are the direct (one-hop) paths to the destination and are less preferred (listed at the bottom).

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 other conflicts simultaneously. Then the safe paths for a particular conflict may or may not be *observable* at the *conflicting safe-alternative nodes*.

Definition 4.6: Observable safe path is a safe path $P = (u, v, \dots, \text{destination})$ at a conflicting safe-alternative node u if none of the other nodes along this path observes route flaps due to other conflicts.

Definition 4.7: Innermost conflict along the path $P = (u_k, u_{k-1}, \dots, u_2, u_1, \text{destination})$ 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 $(u_i, u_{i+1}, \dots, \text{destination})$.

Once the *innermost conflicts* along the safe paths are broken, the *conflicting safe-alternative 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 lower preferred (ranked) but safe path. For each conflict, by Theorems 4.1 and 4.2, we have at least 2 nodes with safe paths. By having a node stabilized to a path at each step, we break the cyclic conflicts. This is the idea behind the convergence proof of our adaptive policy management scheme (APMS), and the following theorem states this idea formally.

Lemma 4.2: During the execution of the APMS algorithm, the size of the set of the nodes that are *conflict-free* increases

monotonically if we perform path rank changes whenever conflict is detected.

Proof: Denote the set of nodes that are *conflict-free* at any step of the algorithm by S . The nodes in S form a routing tree of the paths to which they stabilized. This routing tree is rooted at the destination, and grows as the nodes in S advertise their chosen paths. To show that S grows monotonically, we need to show that at each step of the algorithm, 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 . Assume that node v is in S and already has stabilized to its path $P_v = (v, \dots, destination)$. When node v advertises path P_v to its neighbors that are not in S , one of the following may happen:

Case 1) Path P_v is not permitted at receiving node u , then nothing will happen.

Case 2) Path P_v is permitted at receiving node u and node u stabilizes to path $P_u = (u, v)P_v = (u, v, \dots, destination)$. Then node u is now a *conflict-free node*, and added to S . This happens when node u was already a *conflict-free node*, but its path to the destination contained a node that had been involved in a conflict. Another possibility is node u was a *conflicting node*, and the path P_u is its most preferred path.

Case 3) Path $(u, v)P_v$ is permitted at node u , but node u is not stabilizing on this path. Note that $(u, v)P_v$ is a safe path of node u since node v has stabilized and will keep advertising the same path, P_v , and node u must be in a policy conflict. This also means that node u is a *conflicting safe-alternative node*. At this point, node u performs rank change and sticks to the path $P_u = (u, v, \dots, destination)$, 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.

For cases 2 and 3, the size of S increases monotonically. However, for case 1 the size of S does not change. Looking closer at case 1: The nodes that are not in S , but are immediate neighbors of the nodes in S do not permit the path advertisements coming from nodes in S , then such nodes cannot reach the destination. Then they all become *conflict-free nodes* converging to ϵ . This also implies that not only immediate neighbors of nodes in S , but all the nodes outside of the set S , converge to ϵ . Then the APMS algorithm returns with a stable routing tree. ■

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.

Proof: By using Lemma 4.2, we can show that the algorithm runs in a finite number of steps. Assume that there are $|V|$ nodes in the topology. Since Lemma 4.2 shows that the size of the conflict-free set, S , is monotonically increasing, at each step of the algorithm, the size of the 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 such rank change is done probabilistically, then reaching a stable state in a finite number of steps will take twice as long on average if a conflicting safe-alternative node performs a path rank change with probability 1/2. ■

V. SIMULATION RESULTS

We have simulated the algorithms in the SSF network simulator [19]. We present two sets of results for two different topologies, which are presented in subsection V-A and subsection V-B, 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 updates carried in the last 2000 seconds, and averaged this value over 2000 seconds.

- *The number of octets 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 octets 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 [17], the value of this metric is always zero. For SPVP (Griffin and Wilfong) algorithm, *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 size of the whole set is added to the extra storage. 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 per peer *keepaliveCount*, which are just integers. We have also added the size of the vector that keeps the list of *originators* temporarily from the time of receipt of a topology update until the update is processed. Depending on the time of measurement and the failure, the size of this list maybe zero.

- *Instantaneous throughput* is computed by measuring the number of packets received in the last T seconds, and averaged this value over T seconds. We take $T=2000$ and 100 in our simulation sets, I and II, respectively.

- *Average delay for packets received* is also measured by the delay of the packets received and averaged by the total number of packets received. We computed the instantaneous value of this metric over the last T seconds. We take $T=2000$ and 100 in our simulation sets, I and II, respectively.

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

A. Simulation Set I

The topology is shown in Figure 8(a), and consists of 15 independent dispute wheels, where each AS in a dispute wheel has a direct connection to the destination AS. 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 time. 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 against the SPVP (Griffin and Wilfong) algorithm [13] (see Section II-B for details of this algorithm) and BGP4 [17]. With SPVP, since there is no built-in mechanism to differentiate between transient oscillations and persistent oscillations due to policy conflicts, Griffin *et al* [13] 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.

The first metric measures *the average of the percentage of paths that are eliminated per node at time t* to provide stability. The results are shown in Figure 8(b). BGP4 does not resolve

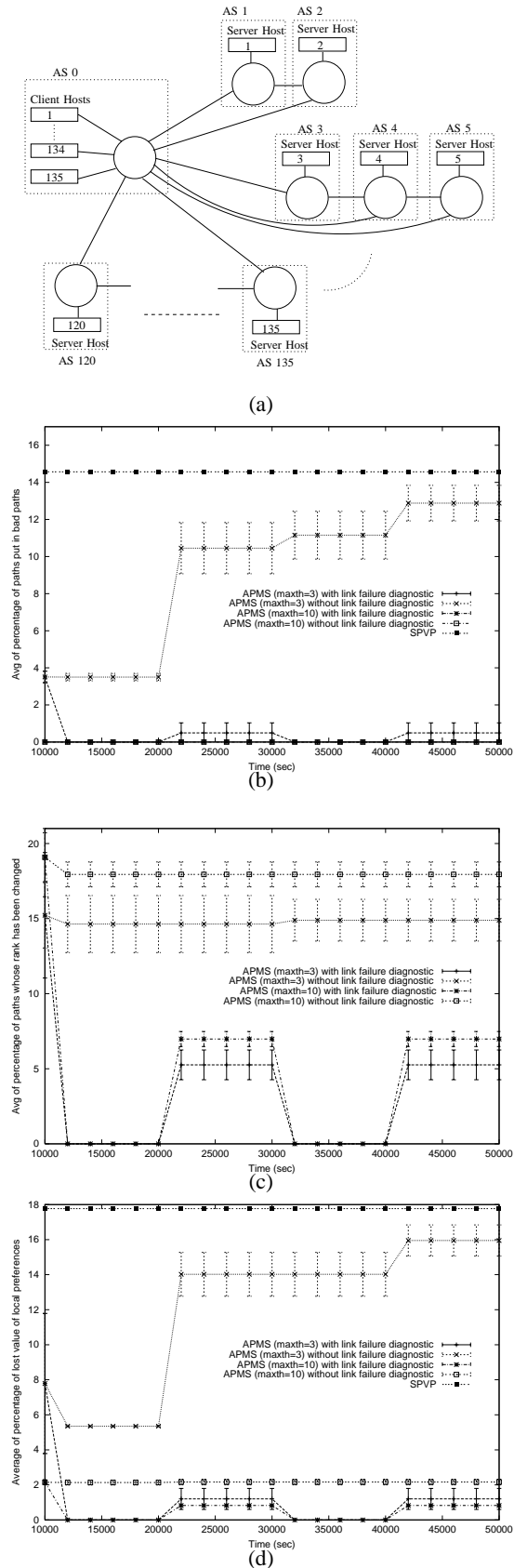


Fig. 8. (a) Topology of Simulation Set I; (b) Average Percentage of Paths Eliminated per Node; (c) Average Percentage of Paths Whose Rank Changed per Node; (d) Average Percentage of Lost Preference Value per Node

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.

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 1.48% 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 8(c) 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 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 8(b), 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 8(d). 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. With APMS, the performance is always better than SPVP. Using larger values of $max_threshold$, 10, and topology change diagnosis significantly improves performance to less than 1% loss in path preference.

Figure 9(a) 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, ...). For topology changes in the form of link failures, we observe a smaller burst of updates due to the 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.

To evaluate of the overhead of the algorithms, *the number of octets carried by update messages* is shown in Figure 9(b). In Figure 9(a), 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 octets carried by its update messages. All versions of APMS show very close and best overall performance. When topology change diagnosis is deployed in APMS, the update messages carry some extra information in the *originator* field, which also contribute to the number of octets carried. However, as we can see the mechanism used by APMS to distinguish temporary oscillations (due to topology change) is much more efficient than the SPVP mechanism of observing repeated cycles in the history.

The results for *throughput* are shown in Figure 9(c). The only factor that affects the throughput in this setting is that of unreachable destination due to the elimination of some of the permitted paths. All versions of APMS perform better than SPVP, because SPVP eliminates the highest number of paths while enforcing stability, and leaves the highest number of nodes with unreachable destination. Among the different versions of APMS, the performance of the version lacking topology change diagnosis, and $max_threshold$ set to

3 slightly deteriorates over time as the number of eliminated paths increases. BGP4 performs well since there is no path elimination. For this experiment, we have unbounded buffers. Therefore, the data packets and update messages are not competing for the buffers. However, if the data and control packets compete for buffers, then the throughput of BGP4 is expected to degrade when the number of update messages are high due to divergence as the experimental setup II demonstrates.

Figure 9(d) 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. For non-fail periods, the value of the metric under SPVP gets higher due to better reachability. For this experiment, while APMS requires around 10KB extra storage, SPVP requires 200KB-360KB extra storage.

B. Simulation Set II

To be able to observe *throughput* and *delay* better, in this setup we use a smaller topology shown in Figure 10(a). The topology consists of 10 independent dispute wheels, where each AS in a dispute wheel has a direct connection to the destination AS. The destination AS has 8 client hosts, to whom there is constant data flow from the servers located in some other ASes. The destination AS has also 8 server hosts, from whom there is constant data flow to the clients located in some other ASes. The routing policies are set to create policy conflicts as in simulation set I.

Simulation is run for 700 seconds, and data flow from servers to clients continues for the whole time. For this experiment set, we have not introduced any topology change. We also limited the buffer sizes, and routing packets are given priority over data packets when there is congestion at the buffers.

Figure 10(b) shows the results for *throughput*. The difference in throughput stems from both unreachable destinations, and/or competition for the limited buffer size. Different versions of APMS perform better than SPVP, because SPVP eliminates the highest number of paths while enforcing stability, and therefore leaves the highest number of nodes with unreachable destination. Also, SPVP has the longest update messages, which take longer to be processed and require more memory to be stored in the buffers. Although BGP4 causes constant exchange of updates due to divergence, its performance is actually better than SPVP! This is because BGP4 does not cause permanent path elimination, even though some packets may not reach the destination temporarily due to instability. Furthermore, the update messages exchanged with BGP4 are not as long as those of SPVP, which results in a smaller number of data packet drops at the buffers. APMS provides the highest throughput as it causes a smaller number of path eliminations than SPVP, and generates a smaller number of update message exchanges than BGP4 (as APMS resolves policy conflicts and resulting oscillations). With APMS, the size of the update messages is much smaller than SPVP: If there is no topology change, the size of the update messages is basically the same as BGP4. Therefore, APMS results in less data packet drops than SPVP both due

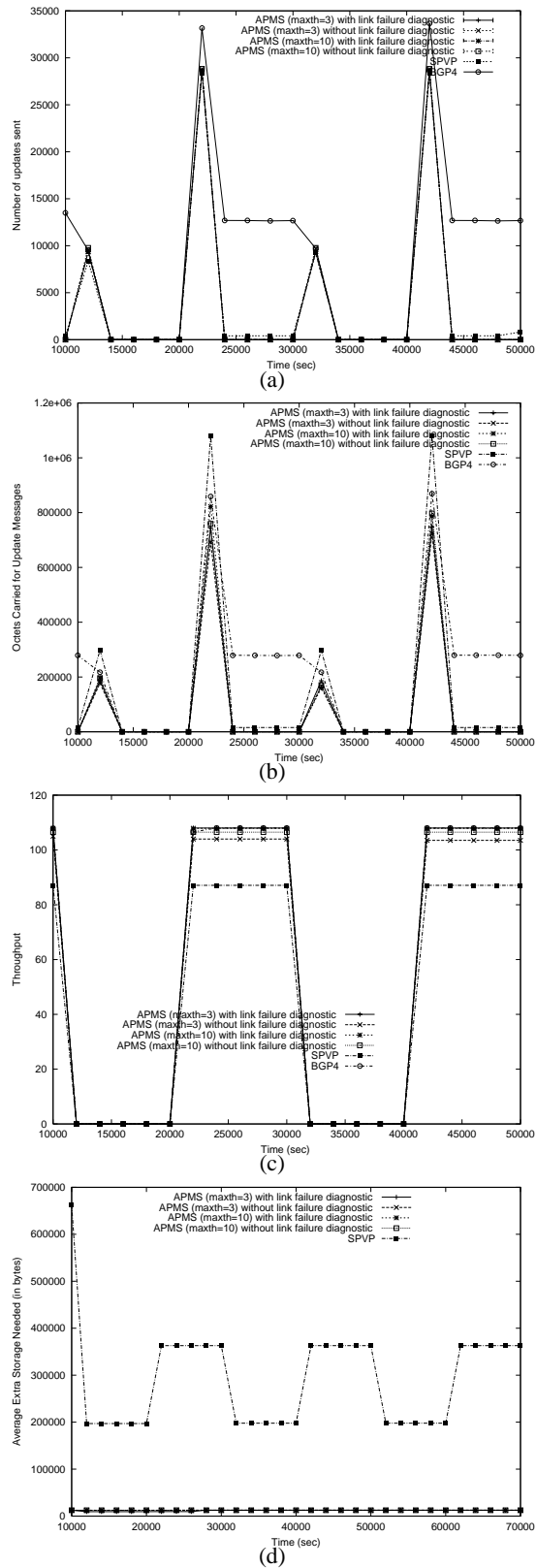


Fig. 9. (a) Number of Updates Sent; (b) Number of Octets Carried by Update Messages; (c) Throughput; (d) Average Extra Storage at Each Router (in bytes)

to less unreachable destinations, and due to shorter update messages. APMS also results in less data packet drops than BGP4 by reaching stability quickly, which leads to smaller number of update messages exchanged, stored and processed.

Figure 10(c) presents the results for *the average delay for packets received*. SPVP causes the highest packet delay due to its longest update messages. The performance of BGP4 is better than SPVP due to shorter update messages, however its performance is worse than APMS due to higher number of update messages exchanged during divergence.

VI. SUMMARY

- 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 conflict³. 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. Griffin and Wilfong [13] and Cobb and Musunuri [3]), 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 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; and (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 [3].

REFERENCES

- [1] A. Arora and M. Gouda. "Distributed Reset", IEEE Trans. Comput., vol. 43, no.9, pp. 1026-1038, 1994.
- [2] J. A. Cobb, M. G. Gouda, and R. Musunuri. "A Stabilizing Solution to the Stable Paths Problem". *Symp. on Self-Stabilizing Systems*, Springer Verlag Lecture Notes in Computer Science, 2704:169-183, 2003.
- [3] J. A. Cobb and R. Musunuri. "Enforcing Convergence in Inter-Domain Routing". In *IEEE Global Communications (GLOBECOM) Conference*, Dallas, December 2004.
- [4] 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.
- [5] J. Feigenbaum, R. Sami, and S. Shenker. "Mechanism Design for Policy Routing". PODC, July 25-28, 2004.

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

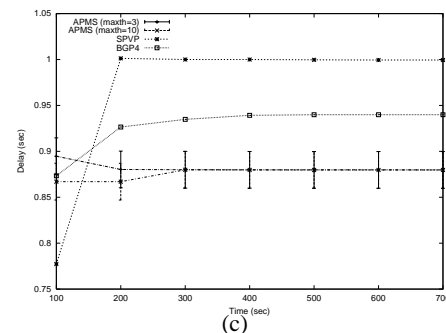
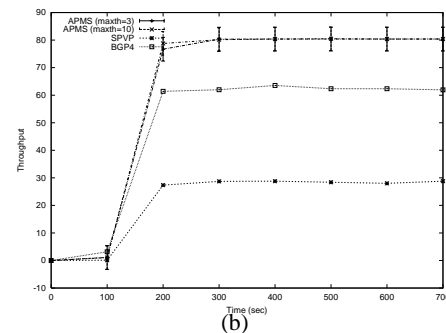
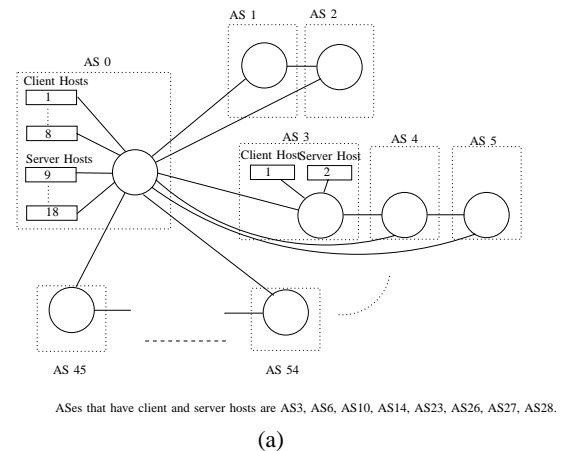


Fig. 10. (a) Topology of Simulation Set II; (b) Throughput; (c) Delay

- [6] L. Gao, T. Griffin, and J. Rexford. "Inherently Safe Backup Routing with BGP". In *Proc. IEEE INFOCOM*, April 2001.
- [7] L. Gao and J. Rexford. "Stable Internet Routing without Global Coordination". In *Proc. ACM SIGMETRICS*, June 2000.
- [8] R. Govindan, C. Alaettinoglu, G. Eddy, D. Kessens, S. Kumar, and W. Lee. "An Architecture for Stable, Analyzable Internet Routing". *IEEE Network*, 13(1):29-35, 1999.
- [9] R. Govindan and A. Reddy. "An Analysis of Interdomain Routing Topology and Route Stability". *INFOCOM*, 1997.
- [10] T. Griffin, A. Jaggard, and V. Ramachandran. "Design Principles of Policy Languages for Path Vector Protocols". In *Proc. ACM SIGCOMM*, August 2003.
- [11] T. Griffin, F. Shepherd, and G. Wilfong. "Policy Disputes in Path-Vector Protocols". In *Proc. IEEE ICNP*, 1999.
- [12] T. Griffin and G. Wilfong. "An Analysis of BGP Convergence Properties". In *Proc. ACM SIGCOMM*, September 1999.
- [13] T. Griffin and G. Wilfong. "A Safe Path Vector Protocol". In *Proc. IEEE INFOCOM*, March 2000.
- [14] A. Jaggard and V. Ramachandran. "Robustness of Class-based Path-Vector Systems". In *Proc. ICNP*, March 2004.
- [15] C. Labovitz, G. Malan, and F. Jahanian. "Internet routing instability". *IEEE/ACM Transactions on Networking*, 6(5):515-528, 1997.
- [16] V. Paxson. "End-to-end Routing Behavior in the Internet". *Transactions on Networking*, 1997.
- [17] Y. Rekhter and T. Li. "A Border Gateway Protocol". RFC 1771, 1995.
- [18] J. Sobrihho. "Network Routing with Path Vector Protocols: Theory and Applications". In *Proc. ACM SIGCOMM*, August 2003.

- [19] SSF. <http://www.ssfnet.org>.
- [20] K. Varadhan, R. Govindan, and D. Estrin. "Persistent Route Oscillations in Inter-Domain Routing". *Computer Networks*, 32:1-16, 2000.
- [21] 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, Massachusetts, November 2004.