An Adaptive Management Approach to Resolving Policy Conflicts*

Selma Yilmaz**[1] and Ibrahim Matta[2]

[1] CISCO Systems Inc., 170 West Tasman Drive, San Jose, CA 95134
seyilmaz@cisco.com
[2] Computer Science Department, Boston University, Boston, MA 02215
matta@cs.bu.edu

Abstract. The Border Gateway Protocol (BGP) is the current inter-domain routing protocol used to exchange reachability information among Autonomous Systems (ASes) in the Internet. BGP supports policy-based routing which allows each AS to independently define a set of local policies regarding which routes to accept and advertise from/to other networks, as well as which route the AS prefers when more than one route becomes available. However, independently chosen local policies may cause global conflicts, which result in protocol divergence. We propose a new algorithm, called Adaptive Policy Management (APM), 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. The convergence analysis of APM derives from the sub-stability property of chosen paths. APM and other competing solutions are simulated in SSFNet for different performance metrics.

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

1 Introduction

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 autonomous system (AS) independently defines a set of local policies regarding which routes to accept and advertise from/to other networks, as well as which route the AS 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. Any policy conflict can be resolved by changing the preference of the ASes over their paths, *i.e.* local policies.

*This work was supported in part by NSF CNS Cybertrust Award 0524477, CNS ITR Award 0205294, and EIA RI Award 0202067.

**This work was done while Selma Yilmaz was at the Computer Science Department of Boston University.

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 require some kind of intervention to stop.

Route instabilities taking place across ASes may negatively impact end-to-end network performance and efficiency of the Internet [1]—packets may be dropped or delivered out of order due to repeated advertising and withdrawal of routes. BGP is crucial for a healthy and efficient global routing, and thus it is a worthy goal to guarantee convergence of BGP independent of the locally selected policies.

Contribution of This Paper: There have been a number of studies on guaranteeing safety, *i.e.* convergence, of BGP independent of the locally selected policies [2], [3], [4]. In our previous work [5], 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 [5] 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 (conservatively) restore some of the original local preference values of their paths, which they have modified, so as to keep the overall path rank change minimal.[3]; (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 is developed based on the sub-stability property of chosen paths; (6) the proposed algorithm is implemented in the SSFNet simulator [7], and simulation results for different performance metrics are presented. The metrics also capture the dynamic performance of our algorithm as well as other competing solutions, thus exposing 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 2 reviews background and related work. Section 3 describes our algorithm. Results and future work are presented in Sections 4 and 5, respectively.

[3]Akin to distributed recovery mechanisms, e.g. congestion avoidance of TCP [6]. As we indicate later, the adaptation of local preference values of paths may also be influenced by input from local AS administrators.

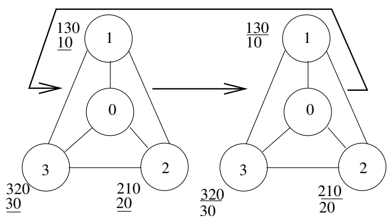

Fig. 1. An example of a divergence. Permitted paths are shown next to each node. Longer paths are more preferred than shorter paths. Current best paths are underlined. Due to the cyclic conflict, this group of nodes cannot reach a stable state and keep oscillating between the shown states.

2 Background

2.1 Border Gateway Protocol Abstraction

We use the abstraction of BGP proposed by Griffin $et\ al.$ [3], 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. In SPP, a network is represented as an undirected graph, $G = (V, E)$, where V represents the autonomous systems and E represents BGP sessions. Node 0 is the destination to which all other nodes are trying to find paths. A $path\ P$ is a sequence of nodes $(v_k, v_{k-1}, \cdots, v_1, 0)$, such that $(v_i, v_{i-1}) \in E$, for all $i, 1 \leq i \leq k$. Paths must be *simple*, *i.e.* no repeated nodes. An empty path, ϵ, indicates that a router cannot reach the destination. Each node v in the graph has a set of *permitted paths*, $\mathcal{P}^v$, to the destination, which are the routes learned from peers, and allowed by the local policy of the node. Each node v also has a *ranking function*, λ^v, to impose an order of preference on the paths, such that more preferable paths have higher values assigned to them.

Given a node u, and $W \subseteq \mathcal{P}^u$ with distinct next-hops, $max(u, W)$ is defined to be the highest ranked path in W. A *path assignment* is a function π that maps each node u to a permitted path. 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 $u \in V$. If a stable path assignment π exists, an SPP is solvable and every such assignment is called a *solution*. An instance of SPP may have no solution.

SPVP is an abstraction of BGP. With this abstraction, 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 $best(u) = max(u, choices(u))$ 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 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. Figure 1 shows an example of a policy conflict leading to divergence.

2.2 Prior Solutions to BGP Divergence

In this paper, due to space restrictions, we will focus only on dynamic solutions, which attempt to detect and resolve policy conflicts at run time. A complete review of previous solutions can be found in [8].

Safe Path Vector Protocol (SPVP): Griffin $et\ al.$ [3] suggest extending BGP to carry additional information called *history* with each routing update message. A possible trace of SPVP for the system shown in Figure 1 is shown in Figure 2 (a). $History$ allows each router to describe the exact sequence of events that led to the selection of a path as the best path. An event $(+P)$ indicates that the node has chosen path P as its best path, and P is more preferred than its previous best path. Similarly, an event $(-P)$ indicates that the node has updated its best path, and this current best path is less preferred than its previous best path P. A $history$ containing loops is an indication of a potential protocol divergence. At step 4 of Figure 2 (a), all 3 nodes have a cycle in the histories of their current best paths. SPVP assumes that such paths are problematic, and therefore eliminates them. For the assumed timing of events, with SPVP the system converges to unreachable destination for all nodes.

Since a cycle in the *history* is a necessary but not sufficient condition for divergence, there may be false positives. Carrying *history* with each update creates communication overhead, and may also reveal private information about the preferences of ASes over the routes. APM uses the idea of keeping track of history of path changes, but does it only locally. By keeping histories as local information and avoiding exchanging such information helps overcome related privacy concerns and communication overhead.

Cobb and Musunuri Algorithm: Cobb $et\ al.$ [4] propose an algorithm which associates an integer cost with each node and exchanges the cost with update messages. The cost increases monotonically if the system diverges. Therefore, discarding the advertisements from the nodes whose cost is greater than a threshold is suggested. Assuming threshold value of 2, Figure 2 (b) shows a possible trace of Cobb and Musunuri algorithm for the system. Since the cost of the nodes involved in the same conflict grows in tandem, all of the nodes simultaneously give up their most preferred paths and stabilize on their lowest preferred paths.

A weakness of this algorithm is keeping per node cost, which causes aggregation of the paths through the same node. One flapping path may cause all the alternative paths (through the same node) to be eliminated. With APM, we extend the idea of using $count$ to keep per-path state at each node instead of per-node state, which prevents aggregation of the paths through the same node. Empowered with this extra information

step	node	best path	history
0	1	(10)	◇
	2	(20)	◇
	3	(30)	◇
1	1	(130)	(+130)
	2	(210)	(+210)
	3	(320)	(+320)
2	1	(10)	(-130)(+320)
	2	(20)	(-210)(+130)
	3	(30)	(-320)(+210)
3	1	(130)	(+130)(-320)(+210)
	2	(210)	(+210)(-130)(+320)
	3	(320)	(+320)(-210)(+130)
4	1	~~(10)~~	$\boxed{(-130)}$(+320)(-210) $\boxed{(+130)}$
	2	~~(20)~~	$\boxed{(-210)}$(+130)(-320) $\boxed{(+210)}$
	3	~~(30)~~	$\boxed{(-320)}$(+210)(-130) $\boxed{(+320)}$
5	1	ϵ	(-10)
	2	ϵ	(-20)
	3	ϵ	(-30)

(a)

step	node	count of node	best path
0	1	0	(10)
	2	0	(20)
	3	0	(30)
1	1	0	(130)
	2	0	(210)
	3	0	(320)
2	1	1	(10)
	2	1	(20)
	3	1	(30)
3	1	1	(130)
	2	1	(210)
	3	1	(320)
4	1	2	(10)
	2	2	(20)
	3	2	(30)
5	1	count(3)≥2, won't use (130) will stabilize on (10)	
	2	count(1)≥2, won't use (210) will stabilize on (20)	
	3	count(2)≥2, won't use (320) will stabilize on (30)	

(b)

step	node	best path	local history	path preferences
0	1	(10)	((10),1)	(130)>(10)
	2	(20)	((20), 1)	(210)>(20)
	3	(30)	((30),1)	(320)>(30)
1	1	(130)	((130),1),((10),1)	(130)>(10)
	2	(210)	((210),1),((20),1)	(210)>(20)
	3	(320)	((320),1),((30),1)	(320)>(30)
2	1	(10)	((130),1),((10),2)	(130)>(10)
	2	(20)	((210),1),((20),2)	(210)>(20)
	3	(30)	((320),1),((30),2)	(320)>(30)
3	1	(130)	((130),2),((10),2)	(130)>(10)
	2	(210)	((210),2),((20),2)	(210)>(20)
	3	(320)	((320),2),((30),2)	(320)>(30)
4	1	(10)	((130),2),((10),3)	$count$(10)> $min\_threshold$ change rank with probability 1/2 assume this happens: (10)>(130)
	2	(20)	((210),2),((20),3)	$count$(20)>$min\_threshold$ change rank with probability 1/2 assume this does not happen: (210)>(20)
	3	(30)	((320),2),((30),3)	$count$(30)>$min\_threshold$ change rank with probability 1/2 assume this does not happen: (320)>(30)
5	1	(10)	stabilizes on lower preferred and available path	(10)>(130)
	2	(210)	stabilizes on most preferred and available path	(210)>(20)
	3	(30)	stabilizes on most preferred and available path	(320)>(30)

(c)

Fig. 2. Possible traces of the algorithms for the system shown in Figure 1: (a) SPVP; (b) Cobb and Musunuri algorithm assuming threshold value for $count$ is 2; (c) APM with $min\_threshold = 2$.

together with probabilistic update of path ranks, APM can pinpoint the paths causing problems, and lead to fewer path elimination.

3 Adaptive Policy Management (APM)

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. Note that such adaptation is limited to the node's set of permitted paths, any of which the AS is willing to use albeit at different preference level.

Each node involved in a particular conflict observes *route flaps*: Constantly chooses a path as its best path and later gives it up for another path. For example, in Figure 1, node 1 constantly upgrades its current best path to (130), but later it is forced to give up (130) for its less preferred path (10) as a result of its neighbors' response to this upgrade. The nodes observing constant route flaps can stop such behavior by sticking to their less preferred but more stable path, even when a better alternative is advertised. This can be achieved by changing the local preference of the paths. When the node stops advertising the paths alternately, the cyclic effect of the global conflict will be broken. In Figure 1, for example, if node 1 changes its local preferences to prefer (10) over (130), the system stabilizes on the following path assignment: (10)(210)(30).

To be able to locally detect route flaps and the paths whose preference is causing divergence, each node needs to keep some form of *local history*. We suggest keeping track of the paths that have been recently selected as best path, and their $count$s indicating how many times the path has been chosen as best path and later given up. Figure 2 (c) shows how $count$s keep increasing during divergence of the system shown in Figure 1. Nodes involved in the conflict can detect divergence by comparing $count$s against a threshold called $min\_threshold$. Since the algorithm we are proposing is distributed and based on using only local information, there may be many nodes synchronously detecting the same conflict and lowering the preferences of their higher preferred paths. If we assume $min\_threshold$=2 for each node in Figure 2 (c), at step 4, all 3 nodes simultaneously change their local preferences to prefer their shorter paths, which are more stable in the sense that they are always available. Note that the conflict can be broken even if only one of the nodes performs the path rank change. To prevent this kind of simultaneous and unnecessary path preference changes, we suggest changing relative preferences with probability 1/2.

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 $max\_threshold$, the path is removed from the set of permitted paths, and added to the *bad paths* set, B. The paths in this set 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.

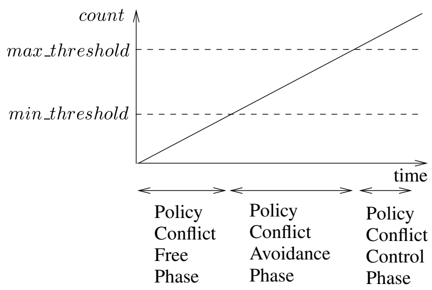

Fig. 3. Phases of APM.

Comparing $count$s against $min\_threshold$, and $max\_threshold$ helps each node independently classify the state of the network: (a) **Policy conflict-free phase**: When *counts* are smaller than $min\_threshold$, the node assumes that there is no persistent oscillation; (b) **Policy conflict-avoidance phase**: If any $count$ value exceeds $min\_threshold$, but stays lower than $max\_threshold$, 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 $max\_threshold$, 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 3 shows these three different phases of our algorithm.

Subsections 3.1 and 3.2 more formally describe APM.

3.1 Update Handling

Figure 4 shows the pseudo-code of our Adaptive Policy Management (APM) scheme for handling routing updates. The process runs at each node u in response to a received update. When node u chooses a path $p \in \mathcal{P}^u$ as its best path, it informs 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 update message.

To find the stable paths, $peerStability$ is associated with each peer. When a peer

```
//Update Handling
process APMS_Update_Handling[u]
  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
        //Policy Conflict−Control Phase
        B(u) = B(u) ⋃ {P_new}
        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
           //Policy Conflict−Avoidance Phase
           find the most preferred safe path, P_safe
             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 the  -->
  is assumed to be executed in one atomic step .
```

Fig. 4. APM: Update Handling.

advertises a path different from its previously advertised path, the $peerStability$ of the peer increases. $peerStability$ of 1 indicates that the path advertised by the peer has not changed over time. The paths advertised by such peers are referred to as *safe paths*. A node observing a route flap can stop the flap by making the safe path its most preferred

```
//Keepalive Handling
process  APMS_Keepalive_Handling[u]
   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 the ⟶
   assumed to be executed one atomic step
```

Fig. 5. APM: Restoring Local Preferences once Stability is Reached.

path, *i.e.* $rank(safe\,path) = 1$. 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.

The state of the system is defined by the local state as well as the local preferences at each node. The state changes whenever there is a path rank change, or a path is placed in B. In either case, this new state corresponds to a different SPP, possibly a stable one. Therefore, counters are reset to give opportunity for a fresh start.

3.2 Restoring Local Preferences

When the system stabilizes, only KEEPALIVE messages are exchanged between peers. Each node keeps track of the number of KEEPALIVE messages received from its peers, and compares this value against a threshold, denoted by $ka\_threshold$, to test the stability of the system. Figure 5 shows how node u probabilistically restores some rank changes for its paths after the system has stabilized. Since policies are placed for a purpose by each node, such as traffic engineering or security, it is important for ASes not to change them unless they are 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 for node u to know 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 with probability 1/4 as well. If node u performs a rank change and/or remove (restore) a path from B, counters kept in the local history are reset because this new state corresponds to a different SPP.

Note that the much smaller probability for reset, *i.e.* 1/4, provides a conservative way of probing the network state, thus oscillating in the vicinity of a stable state at a very slow rate. This is akin to the congestion avoidance mechanism of TCP, during which the current state of the network is probed at a slower rate.

We refer the reader to [8] for the convergence analysis of APM.

4 Simulation Results

We have simulated the algorithms in the SSFNet simulator [7]. We only present one set of results and refer the reader to [8] for additional results.

We have compared APM against the SPVP algorithm [3], Cobb and Musunuri algorithm [4], and BGP4 [9], where the details of these algorithms can be found in Section 2.2. The variations of APM include using different values for $max\_threshold$ of 3 and 10. $min\_threshold$ is set to 2, and $ka\_threshold$ is set to 6. We have two versions of the Cobb and Musunuri algorithm, where the threshold for node cost is set to either 3 or 10 to be consistent with the $max\_threshold$ value of APM. Griffin $et\ al.$ [3] suggest suppressing routes only after seeing the same policy cycle multiple times to handle transient oscillations. In our simulations, we suppressed the routes only after seeing the

Node	Permitted Paths	Local Preference
1	(1 2 0) (1 0) paths learned from 8	100 80 1
7	(7 8 1 2 0) (7 8 1 0) (7 0) paths learned from 19	100 100 80 1
8	(8 9 0) (8 1 0) (8 1 2 0) paths learned from 20	100 80 80 1
9	(9 7 0) (9 0) paths learned from 21	100 80 1

Node	Permitted Paths	Local Preference
19	(19 20 8 9 0) (19 20 8 1 0) (19 20 8 1 2 0) (19 7 0) (19 7 8 1 2 0) (19 7 8 1 0)	100 100 100 80 80 80
20	(20 21 9 0) (20 21 9 7 0) (20 8 9 0) (20 8 1 0) (20 8 1 2 0)	100 100 80 80 80
21	(21 19 7 0) (21 19 7 8 1 2 0) (21 19 7 8 1 0) (21 9 0) (21 9 7 0)	100 100 100 80 80

Fig. 6. Path Rankings for Topology Used in Simulation (Only for the top portion is shown; rest is symmetric).

same policy cycle twice. This is consistent with the $min\_threshold$ value we have chosen for APM. To be able to observe $throughput$ and $delay$, we have used the topology shown in Figure 7. There are 7 groups of nodes: $\{$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$\}$. Each node in a group prefers the path through its clockwise neighbor, which creates a policy conflict for each group. Permitted paths and path preferences are shown in Figure 6.

Simulation is run for 350 seconds, and data flow from servers to clients continues

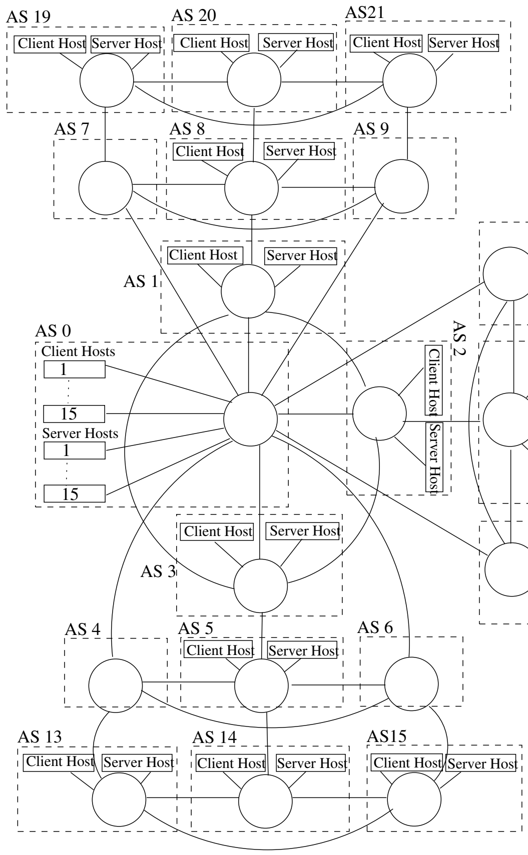

Fig. 7. Topology.

for the duration of the simulation. Buffer size is 50000 bytes, and routing packets are given priority over data packets when there is congestion at the buffers. The performance plots presented next show 90% confidence intervals for the metrics.

Figure 8 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 des-

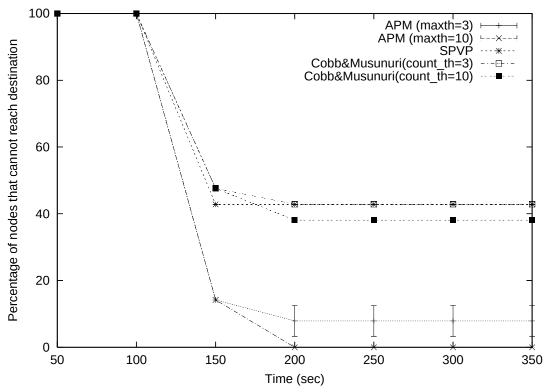

Fig. 8. Percentage of nodes that cannot reach the destination.

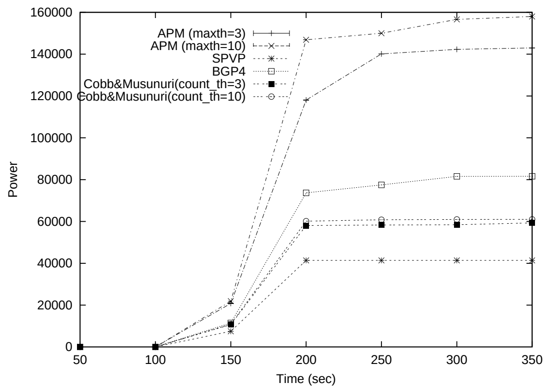

Fig. 9. Power.

tination. Different versions of APM perform much better than SPVP and the Cobb and Musunuri algorithms. With APM, a higher $max\_threshold$ value helps resolve policy conflicts through changing path preferences, and therefore minimize 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. A 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 9 shows the results for *power*. This metric is used to measure the ratio of throughput (average total number of packets delivered over the last 50 seconds) and delay (average delay of delivered packets over the last 50 seconds). The power metric captures the desire of achieving as high throughput as possible while keeping delay as small as possible. Different versions of APM have higher power value than SPVP and the Cobb and Musunuri algorithms because APM maximizes 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 8). 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.

5 Summary and Future Work

Unlike static centralized solutions (e.g. Gao $et\ al.$ [2] algorithm) which may lead to unnecessary elimination of many routes from the start to guarantee stability, APM is a dynamic distributed algorithm allowing ASes to adapt to the current state of the network, either conflict free or potentially conflicting. In this paper, we demonstrated the superiority of APM over other dynamic algorithms [3], [4].

If only some of the nodes in the network were upgraded to deploy APM, APM 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 APM will be the only ones which may give up their preferred paths for the sake of network stability without knowing whether or not the other nodes are working for the same purpose. As future work, to be able to improve cooperation among ASes to deploy APM, incentives should be proposed.

To increase the transparency of APM, we plan to investigate allowing local AS administrators to explicity guide the backoff and recovery probabilities for lowering and restoring the ranks of paths. With such human input, it may be possible to resolve policy conflicts in a more efficient way albeit at a longer timescale. We are currently working on developing a prototype implementation of APM.

References

1. Govindan, R., Reddy, A.: An Analysis of Interdomain Routing Topology and Route Stability. In: Proceedings of the Conference on Computer Communications (IEEE Infocom), Kobe Japan (April 1997)
2. Gao, L., Rexford, J.: Stable Internet Routing without Global Coordination. In: Proceedings of ACM SIGMETRICS, Santa Clara CA (June 2000)
3. Griffin, T., Wilfong, G.: A Safe Path Vector Protocol. In: Proceedings of IEEE INFOCOM, Tel Aviv Israel (March 2000)
4. Cobb, J.A., Musunuri, R.: Enforcing Convergence in Inter-Domain Routing. In: Proceedings of IEEE Global Communications (GLOBECOM) Conference, Dallas TX (December 2004)
5. Yilmaz, S., Matta, I.: 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)
6. Jacobson, V.: Congestion Avoidance and Control. In: ACM SIGCOMM '88, Stanford CA (August 1988) 314–329
7. SSFNet: Scalable Simulation Framework: http://www.ssfnet.org
8. Yilmaz, S., Matta, I.: An Adaptive Management Approach to Resolving Policy Conflicts. Technical Report BUCS-TR-2006-008, CS Department, Boston University (May 2006)
9. Rekhter, Y., Li, T.: A Border Gateway Protocol RFC 1771, 1995.