YAMR: Yet Another Multipath Routing Protocol

Igor Anatolyevich Ganichev
Dai BIn
Philip Brighten Godfrey
Scott Shenker

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2009-150
http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-150.html

October 30, 2009

YAMR: Yet Another Multipath Routing Protocol

Igor Ganichev, Bin Dai, P. Brighten Godfrey, and Scott Shenker

ABSTRACT

As the Internet is now a critical component of our information infrastructure, several recent papers have proposed using multipath routing for increase the Internet's reliability, and to give users greater control over the service they receive. However, the paths chosen by these protocols are not guaranteed to have high diversity. In this paper, we propose yet another multipath routing scheme (YAMR) for the interdomain case. YAMR provably constructs a set of paths that is resilient to any one inter-domain link failure, thus achieving high reliability in a systematic way. Further, even though YAMR maintains more paths that BGP, it actually requires significantly less control traffic, thus alleviating instead of worsening the Internet scalability. This reduction in churn is achieved by a novel hiding technique that automatically localize failures leaving the greater part of the Internet completely oblivious.

1. INTRODUCTION

In recent years, a growing chorus of researchers have advocated the multipath routing paradigm, in which the routing infrastructure makes multiple paths available, allowing senders to select among them. This approach gives users access to the paths that best suit their needs (low latency, high bandwidth, low loss, low jitter), thereby improving reliability and increasing competition among ISPs ([16], [3]). It is hard enough to design multipath routing algorithms for the intradomain case, but the interdomain case is even more challenging because of policy constraints and scaling requirements. There have been several proposals for interdomain multipath routing (see, for example, [15, 13]), and they have made admirable progress in grappling with these two issues; to wit, they have demonstrated that it is possible to provide a set of alternate interdomain paths in a scalable and policy-compliant manner.

The only disquieting aspect of these approaches (and many other multipath proposals in the intradomain case) is that the set of alternate paths is somewhat *ad hoc*; they cannot systematically compute a set of alternate paths that have a high degree of path diversity.[1] That is, while they provide a tunable number of alternate paths, these paths may have significant overlap, thereby leaving the possibility that a single failure could take out the entire set.

In this paper we present the Yet Another Multipath Routing (YAMR) protocol that systematically provides high path diversity. There are two components to the YAMR approach.

(1) An efficient BGP-like mechanism for computing a diverse family of policy-compliant paths: This component of YAMR (which we call YAMR Path Construction, or YPC) computes a set of alternate paths that are deviations from BGP's default path.[2] Each alternate path is computed assuming that a link in the default path is down. Considered as static set of paths, there is no single failure that can break all the paths simultaneously, unless that failure disrupts *all* policy-compliant paths between the source and receiver. When protocol dynamics are taken into account, the story is more complicated (because when BGP recovers from a link failure, it can break paths that did not contain the failed link). We present simulation results on the actual resilience achieved under full dynamics, which show that YAMR imroves the reliability of BGP in single link failures by almost three orders of magnitude.

However, computing this family of paths involves higher control plane messaging overhead than BGP. We therefore added another component to YAMR.

(2) A technique for reducing churn by localizing routing updates: Much of the churn created by BGP is due to the fact that every change in a path must be disseminated to all nodes that use that path. YAMR hides some of these updates, and it turns out that this "update hiding" technique not only reduces YAMR's churn, it also increases (by an order of magnitude) YAMR's resilience, by largely avoiding BGP's

[1] The theory literature has many such algorithms, but they do not lend themselves to scalable, policy-compliant implementation.

[2] This idea is borrowed from [6], which computes the cost of the cheapest path that avoids each link on shortest path.

problem of recovery causing functioning paths to break.

The rest of the paper is organized as follows. In Section 2 we present the YAMR Path Construction algorithm and, in Section 3, we present hiding techniques in YAMR. We describe simulation results in Section 4. In the appendix, we formally prove the theorems cited in this paper.

2. YAMR PATH CONSTRUCTION

2.1 Overview

The core of YAMR is a policy-based multipath construction mechanism that is very similar to BGP. Paths are described by a series of ASes, such as $[A, B, C, D]$; this also defines a series of interdomain links, such as (A, B), (B, C), and (C, D) in the preceding example. For convenience, we use a failure model where these links are the unit of failure. We can easily generalize the algorithm to cover both link failures and domain failures (where all links $(*, A)$ and $(A, *)$ fail for some A), but our notation is cumbersome enough as is, so we opt for clarity over generality in this short paper.

The goal of YPC is to compute a *default* path p_d (that is identical to what BGP would compute) and for each link L in p_d also compute (if one exists) a policy-compliant *alternate* path p_L that does *not* contain the link L. It turns out that we can only guarantee this under a set of restrictive conditions.

We say that the network is in a *canonical condition* when the network has converged and all ASes follow *next-hop* [7] and *widest-advertisement* [11] policies (these policies include the customer-peer-provider policy [8]), and there are no dispute wheels [10]. Under these conditions, we can show:

Theorem 1. *Assuming the canonical condition, for any destination D, AS A, and interdomain link e, if there is a policy-compliant path from A to D that avoids e, then YPC computes a policy-compliant e-avoiding path to that destination.*

We now describe YPC in more detail, starting with the control plane and then moving to the data plane.

2.2 Control Plane

Similar to BGP, ASes in YAMR construct their default and alternate paths from the paths advertised by their neighbors, applying local policies, import and export filters and actions. These multiple paths to the same destination are differentiated using *labels.* Default paths have a special label which we denote by d, while alternate paths are labeled by the link they avoid.

Within this framework, YAMR achieves Theorem 1 by selecting paths as described in Procedure 1, where p_L denotes an L-labeled path, $best_A(U)$ denotes the A's most preferred path from the set U, U_L denotes the set of L-labeled paths A knows from its neighbors, and U_d^L denotes the set of default paths A knows that avoid link L. AS A first selects its default path from the default paths it knows from its neighbors. Because default paths are selected only from default paths, the default paths in YAMR are exactly the same as in BGP. Then, for each interdomain link L on the default path, A selects an L-labeled alternate path from the set of default paths avoiding L and alternate L-labeled paths. To clarify terms, we use *RIB_IN* to describe the set of routes learned from neighbors and *RIB_LOCAL* to describe the set of routes used for forwarding.

Procedure 1: YPC path selection run by AS A.

```
/* select the default path */
$p_d := best_A(U_d)$
foreach link $L$ in $p_d$ do
    /* select the $L$-labeled alternate path */
    $p_L := best_A(U_d^L \cup U_L)$
end
```

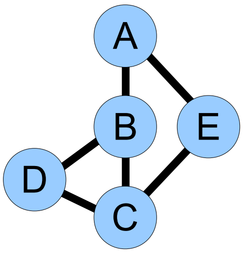
We now walk through a complete run of YPC shown in Figure 1. First, C announces its default path $[C]$ to its neighbors, which then construct their default paths. None of the neighbors is able to construct an alternate path yet. Next, B and D send their default paths to each other. Upon processing these messages each is able to construct the alternate path it needs. Next, B and E send to A the updates to their RIB_LOCALs. A can construct its default paths either from $[B, C]$ or $[E, C]$. A prefers to have $[A, B, C]$ as its default path and now needs to construct alternate paths avoiding links (A, B) and (B, C). For the (A, B)-avoiding path A has the path $[A, E, C]$ as the only choice because the path $[A, B, C]$ goes through (A, B) and the path $[A, B, D, C]$ cannot be considered because of its label (and would be unsuitable anyway, since it does not avoid (A, B)). Finally, A sends updates to its RIB_LOCAL to E, which is now able to pick its alternate path.

Putting all this together, we can show:

Theorem 2. *If there are no dispute wheels, YPC always converges to a unique final configuration that has no loops.*

2.3 Data plane

YAMR requires a single addition to the IP header: a 32-bit field for the path label. A packet arriving at AS A destined to D with label L is forwarded along the L-labeled path towards D if A has such a path in its RIB_LOCAL. Otherwise, the packet is forwarded along the default path towards D (without overwriting the label). If A does not have a default path towards D, the packet is dropped. Once the control plane has



This figure presents a simple run of YPC on the topology shown on the left. AS C announces a single prefix and other ASes build their paths to this prefix. In the table below, the first column shows the messages send by the protocol. The other five columns show the state of the routing tables after all the messages in the first column have been processed. Messages are denoted by "src->dst : msg", where msg contains a number of paths. Each path is denoted by "label:AS-path". In this figure, we denote the default path label by (0,0). Messages that don't result in changes to the routing tables are omitted. Also, note that we picked a particular order of the messages. If another order were picked, intermediate routing tables would have been different.

Messages	A	B	C	D	E
C->D: (0,0):[C] C->B: (0,0):[C] C->E: (0,0):[C]	none	(0,0):[B,C]	local path	(0,0):[D,C]	(0,0):[E,C]
D->B: (0,0):[D,C] B->D: (0,0):[B,C]	none	(0,0):[B,C] (B,C):[B,D,C]	local path	(0,0):[D,C] (C,D):[D,B,C]	(0,0):[E,C]
B->A: (0,0):[B,C], (B,C):[B,D,C] E->A: (0,0):[E,C]	(0,0):[A,B,C] (A,B):[A,E,C] (B,C):[A,B,D,C]	(0,0):[B,C] (B,C):[B,D,C]	local path	(0,0):[D,C] (C,D):[D,B,C]	(0,0):[E,C]
A->E: (0,0):[A,B,C], (A,B):[A,E,C], (B,C):[A,B,D,C]	(0,0):[A,B,C] (A,B):[A,E,C] (B,C):[A,B,D,C]	(0,0):[B,C] (B,C):[B,D,C]	local path	(0,0):[D,C] (C,D):[D,B,C]	(0,0):[E,C] (C,E):[E,A,B,C]

Figure 1: A complete run of YPC on a simple topology.

converged, this algorithm is guaranteed to produce no loops.

For each destination, a YAMR router needs to have a forwarding entry for the default path and for each alternate path whose next-hop is different from the next-hop of the default path. Thus, the state requirements of YAMR are roughly $1 + k$ times that of BGP, where k is the average interdomain path length. Recent measurements suggest that this is around 3.6 [2].

2.4 Discussion

As Theorem 1 shows, YPC is guaranteed to give each AS a policy-compliant path that avoids any given interdomain link (if such a path exists), thus greatly improving reliability. Moreover, users can use all of the paths simply by inserting the appropriate path label into their packets. (The default path lists all the AS links, and so the edge will know which labels will produce different paths; YAMR does not include mechanisms to tell the edge which of these AS links might be providing subpar service.) The paths are constructed and made available to users with moderate increases in the control messaging (or churn) as we will see in Section 4, and in RIB and FIB sizes.

BGP scalability is considered a critical challenge [12] and YPC makes it worse. Among the many dimensions of scalability, churn appears to be the most intractable. Indeed, the comparison of technology trends and projected growth of RIB and FIB sizes in [1] suggests that technology advances are expected to satisfy RIB and FIB memory requirements at a constant cost. We now present a method that reduces YAMR's churn below that of BGP. This churn reduction method involves hiding path withdrawals, leaving most of the Internet completely oblivious to the failure.

3. HIDING ROUTE UPDATES

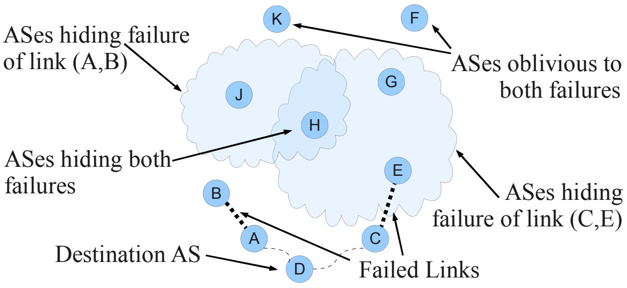

ASes hiding failure of link (A,B)
K
F
ASes oblivious to both failures
J
G
H
ASes hiding both failures
E
B
ASes hiding failure of link (C,E)
A
C
Destination AS
D
Failed Links

Figure 2: An illustration of hiding bubbles.

3.1 Overview

YAMR's hiding technique is a set of distributed mechanisms that can be applied to either YPC or BGP to confine the effects of a link failure to a small neighborhood around the link. Hiding ASes do not propagate information about the link failure to their neighbors if they can safely reroute around the failure. For example, in Figure 1, if link (B, C) fails, B can reroute around this failure by deflecting all traffic onto $[B, D, C]$ with-

Procedure 2: YAMR default path selection.

```
while $U_d$ is non-empty do
    $p_d := best_A(U_d)$
    if $p_d$ is lame then
        $p_d.defl := best\_non\_lame_A()$
        if $p_d.defl$ is null then
            delete $p_d$ from RIB_IN
            continue
        end
    end
    break
end
```

out telling A that path $[B, C]$ has failed. We call B a *hiding* AS, path $[B, D, C]$ a *deflection* path, and path $[B, C]$ (the failed path being hidden) a *lame* path.

In the example above, B is able to completely hide the failure so that all other ASes remain oblivious to it. However, in general topologies and policies, B might be able to hide the failure only from a subset of its neighbors, but can't hide it from others because it does not have a suitable path it can export to them. In such a case, B withdraws the failed path from the neighbors for whom it can't hide the failure. These neighbors then try to hide the failure from their neighbors, recursively. This process continues until the failure is completely hidden. In other words, a single failure is hidden by a dynamically determined bubble of hiding ASes (see Figure 2).

When hiding is combined with YPC to produce the full YAMR protocol, the following results hold:

THEOREM 3. *If there are no dispute wheels, YAMR always converges.*

THEOREM 4. *In the converged state, YAMR has no forwarding loops or dead ends. Moreover, if ASes follow next-hop policies, all forwarding paths are policy-compliant.*

THEOREM 5. *Assuming canonical conditions, for each AS A, if there is a policy-compliant path from A to the destination, A has a policy-compliant path to the destination in YAMR.*

THEOREM 6. *When a failed link recovers, all hiding caused by it stops and routing returns to normal.*

These theorems are prooved in the appendix. Next, we present the four mechanisms that comprise hiding.

3.2 Hiding Path Selection

The fundamental mechanism of hiding is to pretend that a withdrawn path is available. When a path currently in the AS A's RIB_IN is withdrawn from A, A does not delete it from the RIB_IN as it would in BGP. Instead, A marks the path as *lame* and calls path selection. In path selection, if A selects a lame path, it tries to choose a deflection path (from among the set of other default and alternate paths) for it. If there is no suitable deflection path, the lame path is deleted from the RIB_IN and no hiding occurs. YAMR's selection of the default path is presented in Procedure 2. Alternate paths are selected analogously.

After path selection, each lame path in the RIB_LOCAL has a deflection path associated with it. As in BGP, after the RIB_LOCAL has been updated, YAMR announces the changes to its neighbors. Export filters and actions are applied to non-lame paths in exactly the same way as in BGP. However, for lame paths, export filters are applied to the corresponding deflection paths and export actions are applied to the lame path. If export filters allow the deflection path to be advertised to a neighbor, the lame path is actually advertised. Otherwise, a withdrawal message is sent to the neighbor.

3.3 Hiding Forwarding

Forwarding in YAMR is the same as in YPC except that the forwarding entries for lame paths are built based on the corresponding deflection paths. If the lame path's label is different from the deflection path's label, the labels of packets forwarded along the deflection path are replaced by the deflection path's label.

3.4 Tokens

In the two previous sections, we described that YAMR advertises lame paths, but forwards on deflection paths. When there is a single failure, this lie is harmless, but with multiple failures and multiple ASes hiding, lying can cause forwarding loops or leave an AS unnecessarily disconnected. These two problems originate from the following two fundamental issues.

Consider an AS B that advertises a path p to its peer A. Because of hiding, the actual forwarding path p' can be different from p. The difference between p and p' leads to problems in two cases:

1. If A is in p' but not in p, A can select p and create a forwarding loop.
2. If A is in p but not in p', A will not accept p and can become disconnected if p is the only path it was offered.

Hiding solves the two problems by introducing a new message type we call a *token*. There are two types of tokens. *Loop tokens* solve the first problem, while *disconnection tokens* solve the second. We call the messages *tokens*, because their processing resembles passing a physical token - no AS remembers any state for any token and when a token is received it is either dropped or passed on to a single other AS.

At a high level the goal of both token mechanisms is to cause some AS to stop hiding. Informally, hiding tries to hide as much as possible, but it recognizes that sometimes too much hiding can cause problems. The cure for these problems is to reduce the level of hiding in the network by asking some AS(es) to stop hiding. In the distributed environment of the network, the AS that is hurt by a problem is usually not the AS that is causing the problem. Moreover, the AS that is hurt by a problem might not know that it is hurt just by looking at its local state. In the loop detection mechanism, an AS sends a token when there is a possibility that it might be hurt. If indeed so, the token is guaranteed to find the causing AS and to cause it to stop hiding. In the disconnection token mechanism, the AS creates a token when it knows that it is hurt. The purpose of the token is to find the causing AS and to cause it to stop hiding. Next, we describe both mechanisms in details.

Loop Tokens. The loop tokens are created in a single case - when a particular path is picked as a deflection path for a particular lame path for the first time. In this case, the AS that picked the deflection paths creates a loop token and forwards it along the deflection path.

Each loop token T contains two pieces of data: a destination prefix p and a list S of (B, N) pairs, where B is a label and N is an AS number. Each AS that forwards T appends to S a pair (B, N) containing the label of the path on which the token is forwarded and its own AS number. Let L denote the label in the last pair of S.

When a token arrives at an AS A, A first looks up the path P along which it would forward a packet destined to p with label L. Let Q be the label this packet would have when leaving A. Then, A checks how many times (Q, A) is present in S. If twice, A drops T. If none, A forwards T along P. If once, A checks if P is lame. If so, A deletes P from the RIB_IN, thus ceasing to hide it, and drops T. If P is not lame, A forwards T along P.

Disconnection Tokens. The disconnection token mechanism requires a small change to the path selection process: ASes must not use sender side loop detection and the import filters must not filter out loopy paths allowing them to be inserted in RIB_INs. Next, we describe the creation and processing of disconnection tokens.

An AS A creates a disconnection token when its path selection cannot pick a default path (because there are no candidates) but its RIB_IN contains at least one loopy path. A sends the created token along any loopy path and schedules a timer to retransmit the token if the condition persists at the timer expiration.

Disconnection token contains the same data as the loop token and this data is updated the same way. Only the processing at the routers different. As with loop tokens, A first looks up P and Q. If P is lame, A deletes P from the RIB_IN and drops T. Otherwise, A checks if S contains (Q, A). If so, A drops T. Otherwise, A forwards T along P.ath.

3.5 Failed Link Propagation

To ensure that hiding does not inadvertently attempt to hide a permanent change and that the network returns to normal when the failure recovers, we need one last mechanism we call Failed LInk Propagation (FLIP) and a single rule to control when an AS can hide.

FLIP is a path-vector based dissemination protocol, which can essentially be obtained from BGP by removing all policies and replacing IP prefixes with link failures. The only distinctive feature of FLIP is that link failures are propagated only in withdrawal messages. This ensures that only a few ASes that need to know about the failure get notified by FLIP.

The hiding control rule is the following. An AS can hide a failure of a path $[A_1, A_2, \ldots, A_n]$ as long as it knows (from FLIP) that for some $1 \leq i \leq n-1$ link (A_i, A_{i+1}) has failed. Using this rule and FLIP we can summarize when an AS can hide and when it has to stop.

When to Hide. When a failure occurs, FLIP disseminates the failure to some ASes thus allowing them to hide the failure. If another type of change occurs, no failure information is disseminated, thus ensuring that no AS will try to hide this change.

When to Stop Hiding. Each AS is free to stop hiding at any time for whatever reason, but it is required to stop hiding a path when FLIP revokes all link failures relevant to the path. When a failed link recovers, FLIP is guaranteed to revoke the failure information from all the ASes it originally disseminated the failure information to. Thus, when a failed link recovers, all ASes that were hiding this failure are guaranteed to stop hiding and return the network to the original state.

3.6 Discussion

Recalling the high level picture, YPC is able to efficiently construct a set of paths with provable static diversity, but incurs higher messaging overhead than BGP. To decrease the overhead, we developed a hiding technique that, as we will see in the next section, brings the churn level of YAMR below that of BGP. The surprising result, again to be discussed in the next section, is that hiding also substantially improves *resilience*. Hiding localizes the impact of any routing update, and decreases the chance that the convergence process will interfere with any functioning paths.

However, hiding deprives YAMR of some of YPC's valuable properties. First, the set of YAMR paths might not be one-failure resistant because the set of paths might already be hiding failures (so another failure would

	BGP	HBGP	YPC	YAMR
Percent Discon	9.05	8.43	0.12	0.01
Ave Conv Time	23.8	16.7	44.9	1.16

Table 1: Average percentage of ASes experiencing transient disconnectivity (top row) and average convergence time in seconds (bottom row) following a single link failure in a 1000 node topology.

cause the path to fail). Second, YAMR's advertised paths can be different from the forwarding paths. Because ASes cannot be sure about the paths beyond the first hop, they cannot implement policies beyond next-hop policies with 100% confidence. The question is whether the benefits (described in the next section) of YAMR's increased resilience and substantially lower churn compared with YPC are worth these two disadvantages.

4. EVALUATION

Recall that YAMR is composed of the path construction algorithm YPC and a hiding technique. This hiding technique can be applied to BGP, forming what we call HBGP. To understand the contributions of these components to various metrics, we run each experiment for all four protocols: BGP, HBGP, YPC, and YAMR.

4.1 Methodology

We implemented a message-level event driven simulator that includes important features like MRAI timers (with average value of 30 seconds), router processing delay, and message propagation delay. For simplicity, we represent each AS as a single router. We used annotated Internet-like topologies generated using [4].

Our basic experiment is the following. Given a topology and a multihomed stub AS, we make the AS announce a prefix, let the network converge, fail one of the provider links from this AS, and let the network reconverge. This basic experiment is repeated for all multihomed stub ASes and all of their provider links. We use a 1000 node topology for most metrics. To study scalability, we use topologies of sizes from 500 to 5000 in increments of 500.

We selected this initial experiment, which focuses on failures close to the edge, because internal failures are substantially less common and more amenable to recovery, even in BGP. Thus, these edge failures are the most interesting case, and are the dominant case in reality. We also note that this simulation is similar to the live deployment experiment of [14]. Our future work will study a broader class of failure models.

4.2 Results

We present and discuss our preliminary simulation results for reliability, churn and path stretch.

Reliability Table 1 shows the average number of ASes that experience any disconnectivity during the convergence process. We consider an AS to experience disconnectivity if there is ever a moment when none of the paths in its forwarding table are working. The table shows YAMR is almost 1000 times more resilient than BGP. The table also includes the average convergence time of the network. Note that YAMR converges more than 20 times faster than BGP, because of its localization of failures. In both cases, the hiding aspect helps YPC far more than BGP. Presumably this is because YPC provides many more potential deflection paths than BGP.

Because the simulation evaluations of other interdomain multipath routing proposals [17, 15, 13] were done with different methodologies, we have not yet been able to accurately compare YAMR's reliability with them; however, we can give a very rough comparison of YAMR to path splicing [13]. Recall that static reliability means that a routing protocol, at the time of the failure, has an alternate path that avoids the failure. This is an easier quantity to measure than what we studied in our dynamic simulations, but it ignores the fact that when a routing protocol is converging to route around the failure, it can disrupt this functioning alternate path. Nonetheless, it does provide some measure of reliability. Eyeballing Figure 7 in [13], we see that path splicing is able to improve static BGP reliability by about a factor of 15 with 5 forwarding entries per router per destination (that is, there are 15 times more unnecessary disconnections in BGP than there are in path splicing). YAMR, on the other hand, has no unnecessary disconnections when a single link fails, and even in our dynamic simulations which allow routing recovery to disrupt these alternate paths, it has almost 1000 times fewer unnecessary disconnections than BGP.

R-BGP [11] is another promising approach, achieving both perfect static and dynamic reliability when a single link fails. However, it is not a canonical multipath algorithm because it does not make multiple paths available to the users; it only invokes them upon network-detected failure. It also does not have perfect policy compliance.

Churn Figure 3 shows the CDFs of the number of messages following a link event. We present two graphs with and without tokens because token processing is a much lighter operation than update message processing and because separating them shows how many updates hiding saved and how much extra communication it introduced.

In both graphs, YAMR and HBGP significantly outperform YPC and BGP, reinforcing the conclusion that hiding is effective in reducing the messaging overhead. If tokens are ignored, YAMR reduces the message overhead by a factor of 6.2 compared to BGP, and by a

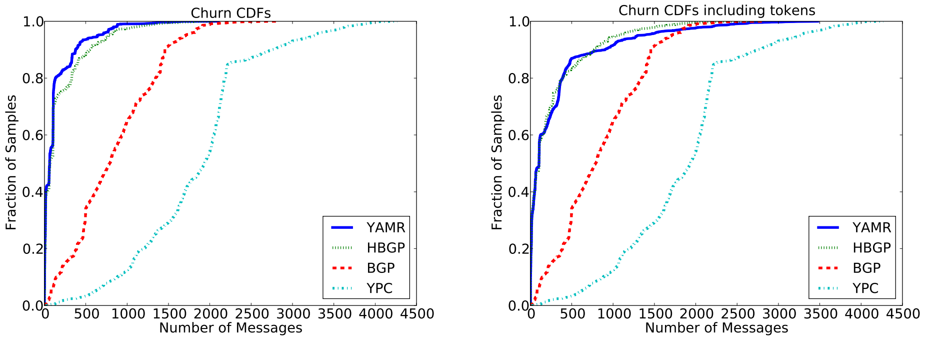


Figure 3: CDFs of number of messages following a link event. On the left side, only update messages are included. On the right side, all messages are included. The averages are BGP: 829, YPC: 1828, HBGP: 178 and 249, YAMR: 134 and 286.

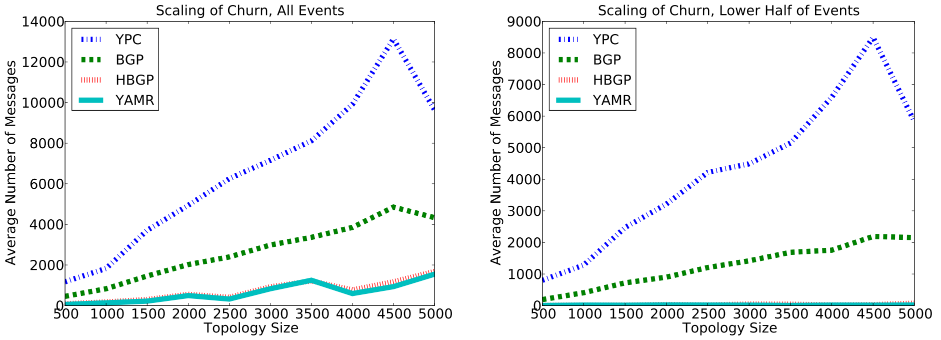


Figure 4: Average number of messages following a link event versus topology size. On the left side, all link events are included. On the right side, only half of the events with lowest number of messages are included, separately for each protocol.

factor of 2.9 if tokens are counted despite the fact that YAMR constructs more paths.

Note that compared to the protocols without hiding (BGP and YPC), the protocols with hiding (HBGP and YAMR) perform relatively better in the lower percentiles than in the higher percentiles. For example, at 20^{th} percentile, YAMR has only 5 messages while BGP has 388 messages. For every failure, BGP requires many messages to converge. YAMR, in contrast, is more bimodal: when the failure occurs in a richly connected portion of the network, it recovers with very few messages, but if connectivity is sparse then recovery is expensive (sometimes more so than BGP). Exploring this bimodal distribution, Figure 4 demonstrates that the size of the smaller convergence events are independent of network size, but the average size of all convergence events grows linearly with network size.

Path Stretch After each of our basic experiments with a single provider link failure, we measured the average path stretch for all 3 protocols and found that the average path stretch was negligible. For example, the average path stretch of YAMR was only 1.02. The reason is that in the Internet-like topologies, it is almost always possible to find an alternate path that has the same length as the shortest path.

Forwarding Table Let F be the average number of forwarding entries per router per destination. As noted in Section 2.3, the pessimal F is $1 + k$, where k is the average path length, and this occurs when every alternate path is different. In our 1000 node topology the average path length is 2.86, so the pessimal F is 3.86, but in our simulations $F = 2.21$, 43% less than the pessimal value. If this holds for the Internet, F for the Internet would be roughly 2.62.

5. REFERENCES

[1] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable internet protocol (aip). In *SIGCOMM*, pages 339–350, 2008.

[2] Routing table report. http://thyme.apnic.net/ap-data/2009/07/23/0400/mail-global.

[3] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: defining tomorrow's internet. *IEEE/ACM Trans. Netw.*, 13(3):462–475, 2005.

[4] X. Dimitropoulos, D. Krioukov, A. Vahat, and G. Riley. Graph annotations in modeling complex network topologies. In *NSDI*, August 2007.

[5] N. Feamster, R. Johari, and H. Balakrishnan. Implications of autonomy for the expressiveness of policy routing. In *SIGCOMM*, pages 25–36, New York, NY, USA, 2005. ACM.

[6] J. Feigenbaum, C. H. Papadimitriou, R. Sami, and S. Shenker. A bgp-based mechanism for lowest-cost routing. *Distributed Computing*, 18(1):61–72, 2005.

[7] J. Feigenbaum, R. Sami, and S. Shenker. Mechanism design for policy routing. *Distributed Computing*, 18(4):293–305, 2006.

[8] L. Gao and J. Rexford. Stable internet routing without global coordination. In *SIGMETRICS*,

pages 307–317, 2000.

[9] M. G. Gouda. *Elements of network protocol design*. John Wiley & Sons, Inc., New York, NY, USA, 1998.

[10] T. Griffin, F. B. Shepherd, and G. T. Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Trans. Netw.*, 10(2):232–243, 2002.

[11] N. Kushman, S. Kandula, D. Katabi, and B. M. Maggs. R-bgp: Staying connected in a connected world. In *NSDI*, 2007.

[12] D. Meyer, L. Zhang, and K. Fall. Report from the IAB Workshop on Routing and Addressing. RFC 4984 (Informational), Sept. 2007.

[13] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala. Path splicing. In *SIGCOMM*, pages 27–38, 2008.

[14] F. Wang, Z. M. Mao, J. Wang, L. Gao, and R. Bush. A measurement study on the impact of routing events on end-to-end Internet path performance. In *ACM SIGCOMM*, 2006.

[15] W. Xu and J. Rexford. Miro: multi-path interdomain routing. In *SIGCOMM*, pages 171–182, 2006.

[16] X. Yang, D. Clark, and A. W. Berger. Nira: a new inter-domain routing architecture. *IEEE/ACM Trans. Netw.*, 15(4):775–788, 2007.

[17] X. Yang and D. Wetherall. Source selectable path diversity via routing deflections. In *SIGCOMM*, pages 159–170, 2006.

APPENDIX

Here we formally prove YAMR's properties that were mentioned in the paper.

A. PRELIMINARIES

First, we define and discuss several preliminaries.

A.1 Policy Assumptions

The proofs of some of our results require two assumptions about ASes' routing policies. We assume that ASes follow *next-hop* and *widest-advertisement* policies, which we define below. Neither of these policy classes are new ([7], [11]). We adopt the definition of former class without change, but clarify the definition of the latter.

We say that an AS follows *next-hop* policies if its export filter is based solely on the destination and on the path's next-hop AS. In other words, all paths for a given destination from a given peer are announced to the same set of peers. We say that AS A's policies are *widest-advertisement* policies if the following is true for all destination prefixes p (all paths are towards p). If A is willing to advertise a path from peer B to peer C, then whenever B advertises a path to A, A has to be willing to advertise its path to C. If A uses a path through C it does not need to advertise it back to C, but the export filter has to allow A to advertise the path back to C. Finally, if the AS hosting a destination prefix follows widest-advertisement policies, it has to advertise the destination prefix to all of its peers.

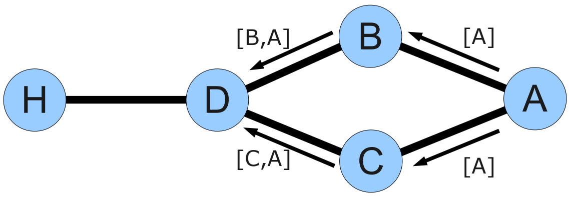


Figure 5: A announces a prefix and sends it with path $[A]$ to B and C. Both B and C choose this path and send paths $[B, A]$ and $[C, A]$ to D, respectively. D prefers path $[D, B, A]$ over path $[D, C, A]$. D's export filter allows path $[D, C, A]$ but path $[D, B, A]$ is blocked. Therefore, D does not send anything to H, which becomes disconnected even though the path $[H, D, C, A]$ is working and policy-compliant.

We make a specific set of assumptions that include valley-free customer-peer-provider policies. Our results most probably hold under a larger set of assumptions, but we do not attempt to find the precise class of policies under which our results hold. In fact, even for BGP the class of policies that are guaranteed to give each AS a path when there is a policy compliant path is unknown to the best of our knowledge. Figure 5 is an example where BGP leaves an AS disconnected when there is a working policy-compliant path.

Definition 1. *A path* $P = [A_n, A_{n-1}, \dots, A_0]$ *is a* policy-compliant *path if for each* $0 \leq i \leq n-1$, A_i *is willing to advertise path* $[A_i, A_{i-1}, \dots, A_0]$ *to* A_{i+1}.

In fact, by contemplating figure 5 and the widest-advertisement assumption, it seems that some version of the widest-advertisement assumption is required to ensure that nodes with policy-compliant paths will not be disconnected in BGP.

A.2 Ordering of Peers

The next-hop and widest-advertisement policies imply a useful categorization of peers that we define and proof here. Next-hop and widest-advertisement policies are assumed throughout this section.

Fix a destination prefix, i.e. everything in this section will refer to paths to a single destination prefix. Let $p_1, p_2, \dots, p_k$ be the peers of an AS A. For each p_i, let s_i denote the set of peers to which A would be willing to advertise a path learned from p_i. The set s_i is determined solely by the export filter and can contain p_i.

Partition peers into equivalence classes based on the equality of corresponding sets s_i and order classes in decreasing order of the size of s_i (for now, break ties arbitrarily. We will show that there is actually no ties.). Denote the peers in class j by $p_{j,1}, p_{j,2}, \ldots, p_{j,k_j}$. Denote the set s_i corresponding to the class j by S_j and let there be a total of c classes.

Lemma 1. *For any two classes $1 \leq i < j \leq c$, $S_j \subset S_i$ (S_j is a strict subset of S_i).*

Proof. Pick two peers $p_{i,m}$ and $p_{j,n}$ from classes i and j, respectively. Consider a case when A gets advertisements only from $p_{i,m}$ and $p_{j,n}$. There are two possibilities: either A chooses a path through $p_{i,m}$ or a path through $p_{j,n}$.

If A chooses a path through $p_{i,m}$, by the assumption of widest-advertisement policies, A has to advertise the path it chose (the path through $p_{i,m}$) to all of S_j. Because, the set of peers to which A advertises a path through $p_{i,m}$ is by definition S_i, S_j has to be a subset of S_i. Furthermore, S_j cannot have the same number of elements as S_i because then the two sets would be equal and would not represent two different classes. Thus, $S_j \subset S_i$

If A chooses a path through $p_{j,m}$, by the symmetric argument to the one above, we get that S_i has to be a strict subset of S_j, which is impossible because $|S_i| > |S_j|$ by the choice of ordering. Therefore, A cannot prefer a path through $p_{j,m}$ if it follows widest-advertisement and next-hop policies. The impossibility of this case implies the next lemma. □

Lemma 2. *For any two classes $1 \leq i < j \leq c$ and any two peers $p_{i,m}$ and $p_{j,n}$ from these classes, a path through $p_{i,m}$ is more preferred than a path through $p_{j,n}$. Further, we call class i a* more preferred class *than j.*

Proof. Follows from the proof of the previous lemma. □

B. YPC CONVERGENCE

In this section, we prove theorem 2. We use the framework and results presented in [10]. Unfamiliar readers should read this work if they desire to rigorously understand our arguments. Otherwise, the basic ideas should be clear.

First, we extend the SPVP definition of [10] to YPC and call it SYPC. Because the extension is an obvious one, we present only the salient differences and omit the details. The `rib` and `rib_in` of SYPC contain not a single path as in SPVP but multiple paths - one per label. Each message still contains a single path, but it also contains a label for this path. The same policies are applied to all paths, independent of their labels. The path selection process is the same as described in the YPC design above. We restate it in the next paragraph.

When a node processes a message with a default path, it updates the default path entry of the peer's `rib_in`. Then, it chooses the best default path from all the available default paths in `rib_ins`. If the best path has changed, it is sent to all the peers and the path selection is run for all the labels on the new default path. When a node processes a message with an alternative path, it inserts this path into the `rib_in` and runs path selection on the path's label. Path selection for an alternative label is the following process. The node chooses the best path from all the default and like-labeled paths in `rib_ins`. If this path is different from the current path with the same label in the `rib`, the `rib` is updated and the path is sent to the peers.

Proof of Theorem 2. First of all, notice that the processing of alternative paths results in some extra state, messages, and processing, but does not impact the dynamics and processing of the default paths. In particular, given an instance of SYPC and an analogous instance of SPVP (with the same graph, the same policies, equivalent initial configuration, and equivalent activation sequence), they will have exactly the same evolution (same state changes and messages). Therefore, because we know that SPVP converges in the absence of dispute wheels, we can conclude that the default paths of SYPC converge in the same conditions. Moreover, because we know that SPVP always converges to a unique configuration, the default paths of SYPC converge to a unique configuration. This argument is a restatement of the fact that the default paths of YPC are constructed in exactly the same way as default paths of BGP. Next, we show that alternative paths also converge.

Intuitively, after the default paths have converged, alternative paths for each label behave just like regular BGP path on a subgraph (the subgraph of nodes whose default paths go through the edge corresponding to the label) of the whole graph. The reasons for this similarity are that the processing of each label is isolated from the processing of any other label (i.e. there is no interdependence between labels), and that the nodes don't change their interest in any label (because the default paths don't change). Therefore, intuitively, all alternative paths should converge. To show this fact more rigorously, for each label e, we define an instance of SPVP that evolves in the same way as the e-labeled paths in SYPC. We call such an instance of SPVP an *e-brother* of SYPC (see figure 6 for an illustration).

Consider the state S of SYPC after all default paths have converged. Let $G = (V, E)$ be the AS graph from the given instance of SYPC. Let the fixed destination prefix p be hosted by AS D. Consider the tree T of converged default paths to D in S. Let T^e denote the subtree of T consisting of the ASes whose default paths go through e. T^e is the tree of ASes that are interested in e-labeled paths. Next, we first define the state of ASes in the e-brother, then the state of edges, and

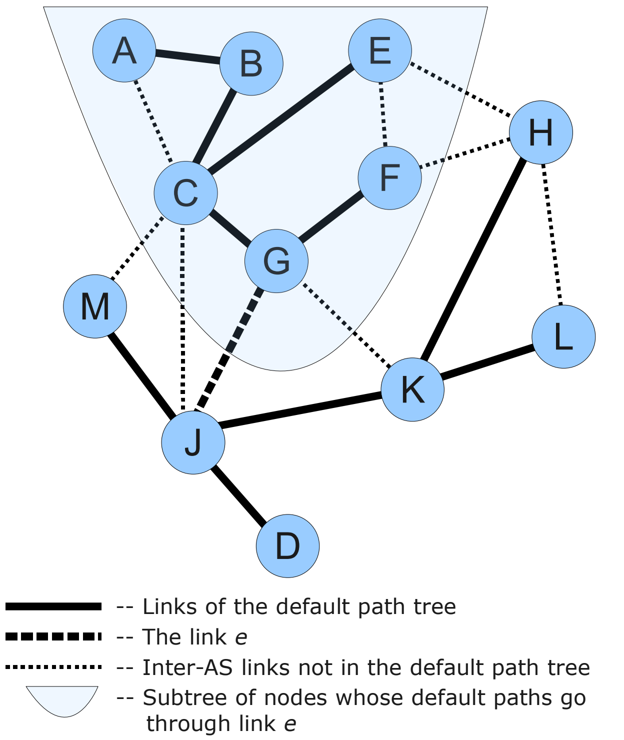

A
B
E
H
C
F
G
M
L
K
J
D
-- Links of the default path tree
-- The link e
-- Inter-AS links not in the default path tree
-- Subtree of nodes whose default paths go through link e

Figure 6: An illustration of *e-brother*. Node D is the destination. The subtree of nodes in the shaded area together with thick edges is the T^e. All the nodes together with thick edges is the default path tree T. The only change to the policies in e-brother is that nodes outside of T^e that have peers in T^e (nodes M, J, K, H), don't accept any paths from these peers. The only structural change is that link e ((G, J)) is removed. These changes cannot introduce a dispute wheel and they preserve the widest-advertisement and next-hop policies.

finally, the permitted path sets.

For each AS A in T^e, its `rib` in e-brother contains the e-labeled path from A's `rib` in S. For each peer AS B of A, the $\texttt{rib\_in}_A(B)$ in e-brother contains the e-labeled path from $\texttt{rib\_in}_A(B)$ in S, if B is in T^e. If B is not in T^e, then $\texttt{rib\_in}_A(B)$ in e-brother contains the default path from $\texttt{rib\_in}_A(B)$ in S.

For each AS A not in T^e, its `rib` in e-brother contains the default path from A's `rib` in S. For each peer AS B of A, the $\texttt{rib\_in}_A(B)$ in e-brother is empty, if B is in T^e. If B is not in T^e, then $\texttt{rib\_in}_A(B)$ in e-brother contains the default path from $\texttt{rib\_in}_A(B)$ in S. If any of the paths is not available, the corresponding entry in e-brother is empty.

The links of e-brother are the same as the links of S except for link e, which is not present in e-brother. For two ASes A and B, the link from A to B contains the following based on whether A and/or B are in T^e:

- $A \in T^e$, $B \in T^e$: the link $A \rightarrow B$ in e-brother contains all the e-labeled paths in link $A \rightarrow B$ in S.
- $A \in T^e$, $B \notin T^e$: the link $A \rightarrow B$ in e-brother is empty.
- $A \notin T^e$, $B \in T^e$: the link $A \rightarrow B$ in e-brother contains all the default paths in link $A \rightarrow B$ in S (there is actually no such paths because all default paths have converged in S).
- $A \notin T^e$, $B \notin T^e$: the link $A \rightarrow B$ in e-brother contains all the default paths in link $A \rightarrow B$ in S (there is actually no such paths because all default paths have converged in S).

Finally, we describe the permitted path sets and complete the definition of e-bother. The permitted path sets in e-brother are exactly the same as in S except that for each AS $A \notin T^e$ and all of A's peers $B \in T^e$, we exclude all paths through B from P^A. The reason for this exclusion is to prevent e-labeled paths from T^e from affecting default paths outside of T^e. This is a feature of YPC and the e-brother should simulate it.

Given the definition of e-brother, it is obvious that it will evolve in the same way as the e-labeled paths in S. Therefore, to show that e-labeled path converge in S, we need to show that e-brother converges. The e-brother is guaranteed to converge because it has the same given policies, which don't have dispute wheels. We only need to note that shrinking permitted path sets cannot introduce a dispute wheel because no new paths are allowed and no preferences between paths have changed. Moreover, since the e-brother converges to a unique final configuration, S will converge to the unique analogous final configuration. Because the argument above can be repeated for each edge e, the whole SYPC is guaranteed to converge to a unique final configuration. □

C. YPC PATH DIVERSITY GUARANTEES

In this section, we proof theorem 1 that we also call a path diversity guarantee. Our path diversity guarantee for YPC follows from a feature of widest-advertisement policies. We first state the feature in a lemma below and proof that BGP (SPVP) has this feature. Then, we apply this feature to YPC.

Recall that in section A.2 we proved two lemmas (1, 2) partitioning AS's peers into classes. We now introduce two new notions: a *nicest possible class* and a *nice path*.

DEFINITION 2. *Consider an instance Z of SPVP (or SYPC) with no dispute wheels and each AS following next-hop and widest-advertisement policies. For each AS A in Z, let C be a set of classes c such that there exists a path $p = [A, A_{n-1}, A_{n-2}, \ldots, A_0]$ such that p is policy-compliant and $A_{n-1} \in c$. Then, the* nicest possible class *for A is the most preferred class in C.*

Definition 3. *A path $p = [A_n, A_{n-1}, \ldots, A_0]$ is a* nice path *of A_n if p is policy-compliant and A_{n-1} is a peer in A_n's nicest possible class.*

Lemma 3. *Let Z be a converged instance of SPVP with no dispute wheels and each AS following next-hop and widest-advertisement policies. Then, for each AS A, if there exists a policy-compliant path from A to the destination, then A is connected in Z through a nice path.*

Proof. Note that this lemma actually makes two separate points and it can be of independent interest because it is about BGP. First, each AS that can possibly be connected (there is a policy-compiant path from it to the destination) will be connected. Second, it will actually be connected through a nice path.

First, note that nodes that have no policy-compliant paths to the destination, will obviously be isolated and won't affect any of the formed paths. Thus, without loss of generality we can assume that there are no such nodes.

We first show the following sublemma. Let A be an AS that is not connected through a nice path in Z (from now on the specification "in Z" is assumed and we don't write it explicitly). Let $p = [A, A_{n-1}, A_{n-2}, \ldots, A_0]$ be a nice path of A. Then, at least one of $A_{n-1}, A_{n-2}, \ldots, A_1$ is not connected through a nice path.

Assume the contrary, that all of the intermediate nodes have a nice path. There are two possible cases: either A is disconnected or A is connected. If A is disconnected n has to be at least 2. Consider A_{n-1}. By assumption, A_{n-1} is connected through a nice path. Because A_{n-1} is willing to advertise a path through A_{n-2} to A and it is connected through a nice path, A_{n-1} has to be willing to advertise its current path to A. Thus, A has to be connected. This contradiction proves the sublemma in the case that A is disconnected. Next, we consider the case when A is connected.

In the case that A is connected, we show that the paths of $A_{n-1}, A_{n-2}, \ldots, A_0$ has to go through A, which is a contradiction because the path of A_0 is $[A_0]$. Let p^n_X and p^c_X denote a nice and the current path of X, respectively. Further, let $p(Y)$ denote the suffix of path p starting at Y. First, note that since all A_i's have a nice path, they all must be connected. If $p^c_{A_{n-1}}$ does not go through A, A_{n-1} has to advertise $p^c_{A_{n-1}}$ to A because it is a nice path, because A_{n-1} is willing to advertise $p(A_{n-1})$ to A, and because we assume widest-advertisement policies. But then, A would have a nice path. Therefore, $p^c_{A_{n-1}}$ has to go through A.

Assume $p^c_{A_{n-2}}$ does not go through A. Because it does not go through A, it does not go through A_{n-1}. Then, A_{n-2} is advertising $p^c_{A_{n-2}}$ to A_{n-1}. Because $p^c_{A_{n-2}} \neq p^c_{A_{n-1}}(A_{n-2})$ (one goes through A and one does not) and because A_{n-1} current path is $p^c_{A_{n-1}}$, $\lambda^{A_{n-1}}([A_{n-1}]p^c_{A_{n-2}}) < \lambda^{A_{n-1}}(p^c_{A_{n-1}})$. Recall that because A's current path is not nice, $\lambda^A([A, A_{n-1}]p^c_{A_{n-2}}) > \lambda^A(p^c_A)$. These four paths and nodes A and A_{n-1} form a dispute wheel, which we assumed does not exist. Thus, $p^c_{A_{n-2}}$ goes through A.

The argument above can be repeated inductively. At the step for A_{n-i}, the following holds:

$$\lambda^{A_{n-i+1}}([A_{n-i+1}]p^c_{A_{n-i}}) < \lambda^{A_{n-i+1}}(p^c_{A_{n-i+1}})$$
$$\lambda^A(p^c_A) < \lambda^A([A, A_{n-1}, \ldots, A_{n-i+1}]p^c_{A_{n-i}})$$

and these four paths together with A and A_{n-i+1} form a dispute wheel. This finishes the proof of the sublemma. Next, we prove the lemma.

Assume the contrary, that there exists an AS A_0 whose path is not nice. Then, pick a nice path $p_1 = [A_0, B_{n-1}, B_{n-2}, \ldots, B_0]$ from A_0 to the destination. On this path, pick a node B_i, $0 \leq i \leq n-1$ such that

- B_i's path is not nice
- Path $[B_i, B_{i-1}, \ldots, B_0]$ is not a nice path of B_i.

Such B_i can be found in the following way. By the sublemma, there is an AS B_j whose path is not nice. If for this B_j, the path $[B_j, B_{j-1}, \ldots, B_0]$ is nice, we can apply the sublemma to B_j and its nice path $[B_j, B_{j-1}, \ldots, B_0]$, to find a node B_k, $k < j$, whose path is not nice. If the path $[B_k, B_{k-1}, \ldots, B_0]$ is a nice path of B_k, we can continue analogously. Because the original path $[A_0, B_{n-1}, B_{n-2}, \ldots, B_0]$ is finite and on each iteration it gets smaller, this process cannot continue forever. Therefore, there exists such a B_i. Rename it to A_1.

Define $A_2, A_3, \ldots$ analogously until some A_q is not the same as a previously found A_r. Without loss of generality, assume that $r = 0$ (because we could have started at A_r). Let p_i be the nice path of A_i we used to find A_{i+1}. Interpret the subscripts module q and let R_i be the prefix of the path p_i until and including A_{i+1}. Let Q_i be the suffix of the path p_{i-1} starting from and including A_i. Let $R = R_0, R_1, \ldots, R_{q-1}$, $Q = Q_0, Q_1, \ldots, Q_{q-1}$, and $A = A_0, A_1, \ldots, A_{q-1}$. Then, $W = (A, Q, R)$ is a dispute wheel. This contradiction, proves the lemma. □

We are now ready to proof theorem 1

Proof of Theorem 1. For the proof, we will use the e-brother defined in the proof of theorem 2. While constructing an instance of e-brother from an instance of SYPC involves many details, here we consider only converged instances of SYPC and e-brother and many of the detail become irrelevant. The only details we need are the structural and policy changes. These changes are described in the caption of figure 6.

In the proof of theorem 2, we have already noted that the structural and policy changes in e-brother cannot obviously introduce a dispute wheel, because they merely cut down on policy-compliant paths. We now show a lemma that if all ASes in an instance S of SYPC follow widest-advertisement and next-hop policies, then all ASes in S's e-brother follow widest-advertisement and next-hop policies.

The essence of widest-advertisement policies is that if an AS A is willing to advertise some path to its peer B, that A has to advertise some other paths to B. Because the changes in e-brother are that some ASes never advertise any path to some other ASes (deletion of edge e is equivalent to the ASes at e ends not advertising anything to each other), the "if" clause of the definition never happens for these pair of ASes. For other pairs of ASes, e-brother does not change their interactions at all. Therefore, widest-advertisement policies are preserved in e-brother.

The preservation of next-hop policies is obvious. Given an AS A and its path p, in S, A would advertise p to a set of peers determined by p's next-hop. In e-brother, A advertises p to the same set of peers, possibly minus some peers to whom A does not advertise anything. Thus, the set of peers to whom p is advertised in e-brother is also determined just by the next-hop of p. This completes the proof of the lemma. This lemma allows us to apply lemma 3 to e-brother.

First, we show that if BGP gives A a path p after failure of e, then there is a policy-compliant path from A to the destination in the e-brother. This is not immediately obvious because p itself might not be a policy-compliant path in e-brother. To show the existence of such a path q, we construct it based on p in the following way.

If $A \notin T^e$, then A's default path in e-brother does not go through e and is obviously a policy-compliant path. In this case, q is A's default path. If $A \in T^e$, let B be the last AS along p starting from the destination that is not in T^e. Let C be the AS after B. By definition, $C \in T^e$. Then, $q = st$ is a concatenation of two paths, where s is the prefix of p from A to B, and t is B's default path (there are actually no "alternative" paths in e-brother, we still use the qualifier "default" to indicate that the path is the same as the default path in SYPC). We only need to show that st is policy-compliant. Because we assumed next-hop policies, to show that st is policy-compliant we only need to show that B is willing to advertise t to C. By lemma 3, B's default path exists and, moreover, is a B's nice path (in SPVP, not only in e-brother). Therefore, B has to be willing to advertise t to C.

Thus, we have showed that there exists a policy-compliant path q in e-brother from A to the destination. By existence of q and lemma 3, node A will have a path p_A in the e-brother's converged state. Because paths in e-brother correspond one-to-one to the e-avoiding path of YPC, p_A is the e-avoiding path that A has in YPC before the failure of e, as desired. □

D. HIDING CONVERGENCE

In the paper, we introduced the hiding technique and applied it to BGP and YPC. We now define a formal model for hiding and prove its convergence properties. Also, the description of hiding in the paper contained many details. The model we define and study here strips many of the details exposing the core of hiding. We hope that the core hiding model can be of independent interest.

D.1 HPV Definiton

We first define the formal model for hiding, which we call Hiding Path Vector (HPV). HPV and the framework around it are based on SPVP [10], where from we borrow most of our definitions. We repeat them here for completeness of presentation.

The HPV algorithm is defined over an undirected graph $G = (V, E)$, where $V = \{0, 1, 2, \ldots, n\}$ is a set of vertices (also called *nodes* and E is a set of edges. Vertex 0 is a special destination vertex. Each edge in E represents two reliable FIFO message queues - one in each direction. For a vertex v, we denote the set of v'th neighboring vertices by $N(v)$.

Definition 4. Path $P = [u_k, u_{k-1}, \ldots, u_1, u_0]$ *is a sequence of nodes* $u_i \in V$ *such that* $(u_i, u_{i-1}) \in E$. *There is a special* empty path *denoted by* ϵ. *Path* P *is called* simple *if all of its nodes are pairwise different. Nodes* u_k, u_{k-1} *and* u_0 *are called the* first node, *the* next-hop, *and the* last node *of* P, *respectively. Paths* $P = [u_k, \ldots, u_0]$ *and* $Q = [v_m, \ldots, v_0]$ *can be* concatenated *into a path* $PQ = [u_k, \ldots, u_0, v_{m-1}, \ldots, v_0]$ *if* $u_0 = v_m$. *Concatenation with the empty path* ϵ *is the identity operation.*

Definition 5. *For each node* $v \in V$, *a set of simple paths* P^v *such that* $\epsilon \in P^v$ *is a set of* permitted paths *at node* v. *We also require that* $P^0 = \{[0]\}$. $\mathbb{P}$ *is the set of all sets* P^v.

Note that sets P^v let us model several different characteristics of BGP. First, we model both import and export filters using permitted path sets. Modeling of import filters is obvious - paths that are rejected by the import filters are not permitted. Modeling of export filters is done in the following way: the case when node u does not export path P to its neighbor v, is modeled by not including path P in P^v. Second, we model the loop detection of BGP by including only simple paths in permitted path sets.

Definition 6. *Each node* $v \in V$ *has a* ranking function λ^v *that assigns a non-negative number (preference) for each path in* P^v. *The higher the value of* $\lambda^v(P)$, *the*

Procedure 3: Procedure for node u when it is activated by a regular activator towards v

```
if there is a pending message $m$ from $v$ to $u$ then
    remove $m$ from the link queue
    $P :=$ path in $m$
    if $P \in P^u$ then
        $\texttt{rib\_in}_u(v) := P$
        if $\texttt{rib}(u) \neq best(u)$ then
            $\texttt{rib}(u) := best(u)$
            foreach $w \in N(u)$ do
                send $\texttt{rib}(u)$ to $w$
            end
        end
    else
        mark $\texttt{rib\_in}_u(v)$ as lame
    end
end
```

higher is v'th preference for P. Furthermore, for all $v \in V$, we require that $\lambda^v(\epsilon) = 0$ and for $P \in P^v$, $P \neq \epsilon$, $\lambda^v(\epsilon) > 0$. Finally, we assume that for $P_1 \neq P_2$, $\lambda^v(P_1) \neq \lambda^v(P_2)$. A weaker form of this injectivity assumption is sufficient for the proofs, but since this assumption is true for BGP, we might as well assume this strong version. Λ denotes the set of all λ^v

Each node in HPV has two data structures: `rib` - corresponding to Loc-RIB in BGP specification, and `rib_in` - corresponding to the Adj-RIB-In in BGP specification. Like BGP, $\texttt{rib}(u)$ always contains the most preferred path among the paths in $\{[u, v]P : v \in N(u), P = \texttt{rib\_in}_u(v)\}$. We denote this most preferred path by *best*(u). Unlike BGP, the $\texttt{rib\_in}_u(v)$ in can contain not only the last path received by u from v, but also the last permitted path (in P^u) received by u from v or the empty path ϵ. The case when $\texttt{rib\_in}_u(v)$ contains last path received by u from v corresponds to the regular BGP-like situation. The case when $\texttt{rib\_in}_u(v)$ contains the last permitted path received by u from v corresponds to the case when u is hiding. The case when $\texttt{rib\_in}_u(v)$ contains the empty path corresponds to the case when u was hiding, stopped, and have not yet received a permitted path from v.

As in [10] and [9], to model the distributed nature of HPV we introduce a notion of an *activation sequence*. Activation sequence specifies when which node does what. Different activation sequences model different execution orders of the true distributed version of HPV. HPV executes by processing one activator from the sequence at a time. We say that i'th activator is processed at time i. Each activator in an activation sequence is tuple $a = (u, v, t)$ where u is the node being activated, v is a neighbor of u towards which u is activated, and t is the type of the activator. HPV has two types of activators: a *regular activator* and a *revealing activator*, which are handled with procedures 3 and 4. Each procedure is run atomically.

Procedure 4: Procedure for node u when it is activated by a revealing activator towards v

```
if $\texttt{rib\_in}_u(v)$ is lame then
    $\texttt{rib\_in}_u(v) = \epsilon$
    if $\texttt{rib}(u) \neq best(u)$ then
        $\texttt{rib}(u) := best(u)$
        foreach $w \in N(u)$ do
            send $\texttt{rib}(u)$ to $w$
        end
    end
end
```

Regular activator for u towards v is handled (procedure 3) by first removing a pending message from v (if there is no message the activator results in a nop) containing a path P. If P is not a permitted path, the current path in the $\texttt{rib\_in}_u(v)$ is marked lame and P is discarded. If P is permitted, it is handled in the standard BGP fashion - it is put into $\texttt{rib\_in}_u(v)$; the current best path is computed; if the current best path is not in the $\texttt{rib}(u)$, $\texttt{rib}(u)$ is updated and the change is sent to the neighbors. Thus, regular activators are essentially, the same as the activators in [10], with the only difference that a non-permitted path is not put into `rib_in`, which is marked as lame instead.

Revealing activator for u towards v is a nop if $\texttt{rib\_in}_u(v)$ is not lame. If $\texttt{rib\_in}_u(v)$ is lame, the lame path is deleted (by setting $\texttt{rib\_in}_u(v)$ to the empty path) and the standard BGP path selection is carried out. Revealing activator essentially brings the $\texttt{rib\_in}_u(v)$ into a state that is equivalent to the one SPVP (model for BGP in [10]) would have brought it into.

We call an activation sequence *fair* if for each pair of neighboring nodes (u, v) it contains infinitely many regular activators $(u, v, "regular")$. In our presentation, all activation sequences are assumed to be fair.

In a truly distributed HPV, regular activators correspond to an arrival of a message from a neighbor, while revealing activators correspond to the node deciding to stop hiding. This decision can be caused by many factors such as a reception of a token (described earlier) or an expiration of a timer. These factors are intentionally left out of the model. The fact that our convergence result for HPV is valid for arbitrary activation sequences shows that it is safe for operators to stop hiding any path at any time.

The last piece that we need to formally talk about HPV is the state consistency. The state of HPV is said to be *consistent* if all of the following hold

1. For all $u \in V$, $\texttt{rib}(u) = best(u)$, i.e. $\texttt{rib}(u)$ con-

tains the best possible path given the values of `rib_in`'s.

2. For each pair (u, v) of neighboring nodes, if the link from v to u is not empty, the last message in this link contains the path in $\texttt{rib}(v)$.
3. For each pair (u, v) of neighboring nodes, if the link from v to u is empty, then one of the following is true
   (a) $\texttt{rib}(v) \in P^u$ and $\texttt{rib\_in}_u(v) = \texttt{rib}(v)$
   (b) $\texttt{rib}(v) \notin P^u$ and $\texttt{rib\_in}_u(v) = \epsilon$
   (c) $\texttt{rib}(v) \notin P^u$ and $\texttt{rib\_in}_u(v)$ contains the last permitted path that v sent to u.

Throughout the paper we assume that the initial state is consistent. It is easy to check that any activation sequence (not necessarily fair) takes the system in consistent state into consistent state.

Thus, we have finished defining HPV. Its inputs are a graph, a collection of ranking functions, a collection of permitted path sets, an activation sequence, and an initial state. This 5-tuple of inputs define an instance of HPV.

D.2 Dispute Wheels

Next, we restate the definition of a dispute wheel from [10].

Definition 7. *A sequence of nodes $U = u_0, u_1, \ldots, u_{k-1}$ together with two sequences of nonempty paths $Q = Q_0, Q_1, \ldots, Q_{k-1}$, $R = R_0, R_1, \ldots, R_{k-1}$ constitute a* dispute wheel *$W = (U, Q, R)$ if*

1. R_i is a path from u_i to u_{i+1}

2. $Q_i \in P^{u_i}$

3. $R_iQ_{i+1} \in P^{u_i}$

4. $\lambda^{u_i}(Q_i) < \lambda^{u_i}(R_iQ_{i+1})$

where all subscripts are to be interpreted module k. For an illustration of a dispute wheel see figure 9.a of [10].

D.3 HPV Convergence

Given an instance of HPV, we say that $\texttt{rib\_in}_u(v)$ converges if $\texttt{rib\_in}_u(v)$ does not change after some time t_0. We say that a node u converges if $\texttt{rib}(u)$ does not change after some time t_0. Finally, we say that HPV converges is all nodes converge. Next, we state the main theorem, discuss why we choose to prove this theorem, prove a number of lemmas, and finally prove the theorem.

Theorem 7. *An instance of HPV converges if it does not have a dispute wheel.*

Even through the analogous result for BGP convergence is not the sate of the art ([5]), it is well-known and relatively simple result. At the same time, it is powerful enough to guarantee convergence in at least two important classes of policies - the customer-peer-provider policies and the generalized shortest-path based policies. Finally, this result is particularly appealing because the proof exposes the effects of hiding on the dynamics of the model.

Let $value(u)$ be a set of paths that node u picks infinitely many times. C is the set of nodes that converge (whose $value()$ has a single path), R is the set of converged `rib_in`'s, and O is the set of oscillating nodes (whose $value()$ has at least two paths). It is obvious that O and C are disjoined and cover V.

Lemma 4. *Given nodes u and v, if $\texttt{rib\_in}_u(v) \notin R$, then $v \in O$.*

Proof. We prove the contrapositive - if $v \in C$, then $\texttt{rib\_in}_u(v) \in R$. By definition, because $v \in C$, $\texttt{rib}(v)$ does not change after some time t_0. Because $\texttt{rib}(v)$ does not change after t_0, v does not send any messages to u after t_0. Thus, the link from v to u is always empty after some time t_1 (because we always assume that the activation sequence is fair). Then, regular activators for u towards v in a nop after t_1 and the $\texttt{rib\_in}_u(v)$ cannot change during regular activator processing after t_1. Moreover, $\texttt{rib\_in}_u(v)$ can change at most once during a revealing activator processing after t_1. Thus, there is a time after which $\texttt{rib\_in}_u(v)$ does not change, i.e. $\texttt{rib\_in}_u(v) \in R$ as desired. □

Lemma 5. *If $u_0 \in O$ and $P_0 = [u_0, u_1, \ldots, u_{k-1}, u_k = 0] \in values(u_0)$, then for some $0 \le i \le k-1$, $\texttt{rib\_in}_{u_i}(u_{i+1}) \in R$. In other words, there is a convergent `rib_in` along P_0.*

Proof. If $\texttt{rib\_in}_{u_0}(u_1) \in R$, we are done. Assume $\texttt{rib\_in}_{u_0}(u_1) \notin R$. Then, by lemma 4, $u_1 \in O$. We next show that $P_1 = [u_1, \ldots, u_{k-1}, u_k = 0] \in values(u_1)$

First, because $P_0 \in values(u_0)$ and $\texttt{rib\_in}_{u_0}(u_1) \notin R$, P_1 has to appear in (and disappear from) $\texttt{rib\_in}_{u_0}(u_1)$ infinitely many times. The only way for P_1 to appear in $\texttt{rib\_in}_{u_0}(u_1)$ replacing another path is for u_0 to process a message containing P_1 from u_1. Therefore, u_1 sends P_1 infinitely many times, which implies that $P_1 \in values(u_1)$.

Thus, we have showed that if $\texttt{rib\_in}_{u_0}(u_1) \notin R$, then $u_1 \in O$ and $P_1 \in values(u_1)$. Applying the same argument, we can show that if $\texttt{rib\_in}_{u_1}(u_2) \notin R$, then $u_2 \in O$ and $P_2 \in values(u_2)$. Continuing analogously, if no `rib_in` along the path converges, then $u_k = 0$ has to be oscillating, which is impossible. Thus, for some $0 \le i \le k-1$, $\texttt{rib\_in}_{u_i}(u_{i+1}) \in R$. □

Proof of Theorem 7. We prove the contrapositive statement - if an instance of HPV does not converge, there is a dispute wheel. Since HPV does not converge, O is nonempty. Let $u \in O$ and $P \in values(u)$. Then, by lemma 5, there is a convergent `rib_in` along P. Let u_0 be the first node along P, whose `rib_in` from the downstream node is convergent. Then, $u_0 \in O$ because otherwise the node upstream of u_0 would have a convergent `rib_in` from u_0 (and hence u_0 would not be the first node with convergent downstream `rib_in`).

Let H_0 be a path starting at u_0 such that $P = [u, \ldots, u_0]H_0$ Because, the downstream `rib_in` from u_0 is convergent, the path H_0 is always available to u_0 after some point in time. Because H_0 is always available and u_0 is an oscillating node, H_0 has to be the least preferred path by u_0 from among all the paths in $values(u_0)$. Let $J_0 \in values(u_0)$ be a more preferred path than H_0.

Now, we have an oscillating node u_0 and a path $J_0 \in values(u_0)$. Using the same argument as we did for u and $P \in values(u)$, we can find an oscillating node u_1 and a path $J_2 \in values(u_1)$. Continuing in this fashion, for some value of k, a newly found u_k will be equal to an already found u_j. Notice that an oscillating node can have at most one path it picks infinitely many times that comes from a convergent `rib_in` - the least preferred path in $values()$ of this node. Therefore, $H_k = H_j$.

Now, lets rename u_i to a_{i-j} and H_i to Q_{i-j} for $i = j, j+1, \ldots, k-1$. Also, for $i = 0, 1, \ldots, k-j-1$, let R_i be such a path that $J_{j+i} = R_i H_{j+i+1}$. In other words, R_i is the prefix of J_{j+i} until and including u_{j+i+1}. Finally, let $m = k-j-1$. With these definitions, we now have a dispute wheel, $W = (A, Q, R)$, where $A = a_0, a_1, \ldots, a_m$, $Q = Q_0, Q_1, \ldots, Q_m$, $R = R_0, R_1, \ldots, R_m$. The first three conditions for the dispute wheel in definition 7 are obviously satisfied by construction. The last condition holds because J_i was picked to be a more preferred path than H_i for $i = 0, 1, \ldots, k-1$, and because $H_k = H_j$. This completes the proof. □

In this section, we defined a HPV and proved that it converges under any fair activation sequence as long as the policies don't contain dispute wheels. HPV can be viewed as an application of the hiding technique to SPVP. The hiding technique simply says that when a node's peer withdraws a path from it, the node can pretend that it continues to have the path until the peer announces another path or until the node decides to stop pretending. An intuitive reason why hiding does not affect convergence of SPVP is because hiding does not introduce anything new into the dynamics of SPVP - it simply *delays* the processing of a withdrawal. SPVP processes the withdrawal immediately, while HPV processes the withdrawal when it decides to (or never if the peer sends a new path before the node processes the withdrawal). In some sense, the fact that SPVP converges under any activation sequence means that SPVP's dynamics are invariant under the timing of events. Thus, it seems reasonable that if the timing of withdrawal processing is made variable (what hiding does to SPVP), SPVP will still converge.

In the next section, we show how the convergence result for HPV can be applied to YAMR to prove its convergence. To avoid boring the reader with pages of formal details, we present only the main constructs and arguements of the proof.

D.4 HSYPC Convergence

In the previous sections we defined SYPC to be the formal model for YPC. SYPC for YPC is what SPVP is for BGP. In this section, we talk about HSYPC, which is the result of applying the hiding technique to SYPC, just like HPV was the result of applying the hiding technique to SPVP. Throughout the discussion, we consider an instance of HSYPC, Z, with no dispute wheels and a consistent initial configuration. We assume that there are no link events and policy changes after some moment. Our goal is to prove that HSYPC converges. Note that HSYPC is a simplified model YAMR. Thus, in this section we come very close to proving that YAMR converges (i.e. proving theorem 3). In the next section, we actually prove theorem 3.

First of all, the default paths of HSYPC are constructed in exactly the same way as the paths of HPV. Therefore, by theorem 7, the default paths of HSYPC converge.

After convergence of default paths, the dynamics of paths with a given label are completely independent from the dynamics of other labels. Therefore, it is sufficient to show that paths with a given label e converge.

Let S^e be the set of nodes whose (converged) default paths contain e. Let S be the set of all nodes whose default paths don't contain e. If Z contains nodes that don't have default paths, these nodes are completely isolated and we can assume there are no such nodes without loss of generality. Therefore, $S \cup S^e = V$, where V is the set of all nodes as usually. Using the notation from the previous section, we can note that $S \subseteq C$ and $O \subseteq S^e$.

Note that because of hiding the set S^e does not have to be a tree. However, similar arguments that we used in the previous section apply here as well. For the remainder of a section, when we talk about the path of a node u, we mean u's default path if $u \in S$ and u's e-labeled path if $u \in S^e$. Further, when we talk about $\texttt{rib\_in}_u(v)$, we mean the default path in $\texttt{rib\_in}_u(v)$ if $v \in S$ and the e-labeled path in $\texttt{rib\_in}_u(v)$ if $v \in S^e$. In other words, $\texttt{rib\_in}_u(v)$ contains the path that we are concerned with for node v.

Thus, for each node we have at most one path that we are concerned with and there is a single path in $\texttt{rib\_in}_u(v)$ for each connected ordered pair (u, v) of nodes that we are concerned with. The interdependence of these paths is not the same as the paths in HPV - the paths at different nodes affect each other as was described in the YAMR's design. However, the dynamics are sufficiently similar to HPV that we can use the same notation and the same arguments.

Lemma 6. *Given nodes u and v, if $\textit{\textbf{rib\_in}}_u(v) \notin R$, then $v \in O$.*

Proof. The meaning of this lemma in the new context is somewhat different from the meaning of lemma 4. However, the unchanged proof of lemma 4 proves this lemma as well. □

Lemma 7. *If $u_0 \in O$ and $P_0 = [u_0, u_1, \ldots, u_{k-1}, u_k = 0] \in values(u_0)$, then for some $0 \le i \le k-1$, $\textit{\textbf{rib\_in}}_{u_i}(u_{i+1}) \in R$. In other words, there is a convergent **rib_in** along P_0.*

Proof. First of all, $u_0 \in S^e$ because $u_0 \in O$. If for any i, $0 \le i \le k$, $u_i \in S$, the statement is obvious. Thus, we can assume that for all i $u_i \in S^e$. Then, the proof of lemma 5 applies here unchanged. □

At this point, we have repeat the proof of theorem 7 verbatim (but obviously using the new meanings for paths and `rib_ins`) to show that e-labeled paths converge. As mentioned earlier, repeating the same argument for all labels, show that HSYPC converges when there are no dispute wheels.

D.5 YAMR Convergence

In this section, we finally prove theorem 3. We restate it here in a more rigorous way.

Theorem 8. *Consider an instance Z of YAMR with consistent initial configuration, no dispute wheels, and all ASes following widest-advertisement and next-hop policies. Assume there are no policy changes and link events after time t. Then, there exists a time $t' > t$ after which no update messages are sent, no tokens are sent, and no state changes occur. In other words, YAMR converges completely.*

Proof. Consider an instance X of HSYPC that corresponds to Z. In the previous section, we showed that X converges. This implies that after some time t_0 no update messages are sent and no state changes occur Z. Therefore, we only need to show that no tokens are sent after some time.

Before showing that tokens cease to be sent, we need to realize that no state changes after t_0 implies that after t_0 no AS changes what it hides. In other words, no AS can decide to stop hiding some path after t_0 by definition of t_0. ASes have complete freedom deciding what to hide and when to stop hiding, but whatever they do, they can continue forever as was shown in the previous section. Thus, there exists a time, t_0, after which no hiding changes occurs. This logical point that gives complete freedom to ASes and yet claims that all ASes stop after some point can be hard to understand. We further illustrate it with an example.

Consider a finite set of integers C. Then, in any infinite sequence S of integers from C, there exists an index after which S contains only integers that appear in it infinitely many times. Think of constructing a sequence S_0 that will contradict this statement. We have complete freedom to choose how many elements of C will appear in the sequence finitely many times. Further, we have complete freedom to choose how many times each of these elements will appear in S_0. Even further, we can put each occurrence of each of these elements as far in the sequence as we want. Yet, despite all these freedoms, there is an index after which there is none of our elements. The convergence of state in hiding has the same logic. ASes have a lot of freedom, but after some point they all stop. Next, we show that loop and disconnection tokens stop.

Recall that loop detection tokens are sent only at the end of path selection. Path selection can only be triggered by an update message or by a decision to stop hiding (because of a reception of another token or inability to find a deflection path). We know that no update messages are sent after t_0. We also argued above that no AS can decide to stop hiding after t_0. Therefore, no loop detection tokens can be sent after some time $t_1 > t_0$ when all the updates message sent before t_0 have been processed (no new ones can be sent after t_0).

Lastly, we show that disconnection tokens cannot be sent forever. Assume the contrary that there exists an AS A that sends disconnection tokens forever. In particular, A sends tokens after t_1 when all the state has converged and no update messages are sent. Because A sends disconnection tokens A must choose a loopy path. Because the state has converged and the control path is loopy, the forwarding path corresponding to it has to contain a hiding AS B. Upon reception of A's disconnection token B has to stop hiding, thus changing its state. This contradiction completes the proof. □

E. HIDING LOOP-FREENESS

In this section, we prove theorem 4 that guarantees loop-freeness of YAMR (and HBGP as a special case).

As before, we fix a destination prefix. We say AS A *has an e-labeled path* if A's `rib` contains an e-labeled path (even if this path is lame). We also introduce the notion of label L's *forwarding path* starting at node A. Given a node A and a label L, the forwarding path of L starting at A is the path on which a packet leaving A with label L will travel. The label can change along the forwarding path. The forwarding path can contain a loop, in which case it is infinite. Next, we restate theorem 4 more rigorously and prove it.

Theorem 9. *Assume after some time t_0, there are no link events and policy changes for a sufficiently long time that the network converges at time t_1. Then, if an*

AS has an L-labeled path at time t_1, the forwarding path of L starting at the AS is finite and ends at the destination. In other words, if the AS sends a packet with label L the packet will reach the destination (ignoring practicalities like data corruption).

PROOF. Assume the contrary, that there is an AS that has a path for a label but whose forwarding path for this label does not reach the destination. Note that since the network is in the converged state at time t_1, the forwarding path cannot be finite and not end at the destination. If the forwarding path ends at a node X that is not the destination and the previous nodes is Y, then Y forwarding table is not consistent with X RIB, which is impossible in the converged state. Thus, the forwarding path has to be infinite and hence contain a loop.

Let a *supernode* (A, L) be a pair, where A is a node and L is a label. Intuitively, a supernode (A, L) is a node A and its path for L. It is useful because we can talk about *the* path of a supernode and *the* forwarding entry of the supernode without specifying the label. Using supernodes, we can represent the loop F in the forwarding path as a sequence of supernodes $[(A_0, L_0), \ldots, (A_{k-1}, L_{k-1}), (A_k, L_k) = (A_0, L_0)]$, where each supernode (A_i, L_i) represents the fact that packet sent on the forwarding path arrives at node A_i with label L_i. Whenever we use indices of supernodes in F, they are to be interpreted module k. We call a supernode (A_i, L_i) a *hiding supernode* if the path for label L_i at node A_i is lame. We say that a supernode (A_i, L_i) sent a loop token we mean that node A_i sent a token along its L_i-labeled path. We say that a supernode (A_i, L_i) changed its forwarding state if the next-hop supernode of (A_i, L_i) has changed. Note that if the label changed the next-hop node can be the same.

Because the network state has converged and there is a loop F in the forwarding path, F has to contain a hiding supernode. Without loss of generality let (A_0, L_0) be the last hiding supernode in F that sent a loop token T, say at time t_2 (if there are multiple such supernodes, let (A_0, L_0) be an arbitrary one of them). Next, we show that there exists a supernode in F that changed its forwarding state after it processed T (not necessarily because of T).

Recall that a hiding supernode always sends a token after it changes its forwarding state. Because T is the last token (A_0, L_0) sent, it did not change its forwarding state after sending T and T was sent to (A_1, L_1). Now, if there is no supernode in F that changed its forwarding state after processing T, then the token must have traveled around F and must have come back to (A_0, L_0). (A_0, L_0) would then have changed its forwarding contrary to the fact above. Thus, there is a supernode that changed its forwarding after processing T, say at time t_3. Let (A_j, L_j) be the supernode that changed its forwarding last among all the supernodes in F, say at time t_4 (if there are multiple such supernodes, choose one randomly). Then, $t_4 \geq t_3 > t_2$ and hence there are no tokens sent by supernodes in F after t_4.

Because (A_j, L_j) did not send a token after updating its forwarding state at time t_4, it is not a hiding supernode. Therefore, it must have sent an update to (A_{j-1}, L_{j-1}) because no forwarding changes happen after t_4. Because (A_{j-1}, L_{j-1}) also did not send a token after receiving this update, it is not a hiding supernode and it must have sent an update to (A_{j-2}, L_{j-2}) (because no forwarding changes happen after t_4). We can continue this argument until we conclude that (A_0, L_0) is not a hiding supernode, which contradicts our choice of (A_0, L_0). This contradiction completes the proof. □

F. HIDING CONNECTIVITY

In this section, we formally prove theorem 5. Note that it is enough to prove that an AS that can be connected is connected though its default path. Thus, it is enough to prove theorem 5 for HBGP. We start with some definitions:

- *control path* - a path that the AS has in its RIB or RIB_IN. Control path can be lame. Then, it has a deflection path. For an AS A, we denote, its control path by c_A. By $c_A(u)$ we denote the control path of A were it to choose the peer u's control path as best.
- *forwarding path* - a path that the packets actually travel from an AS. For an AS A, we denote, its forwarding path by f_A. By $f_A(u)$ we denote the forwarding path of A were it to forward through peer u. Note that $f_A(u)$ can be infinite if the packet comes back to A.
- Given a forwarding path p, the corresponding control paths is denoted by $ctrl(p)$. Given a control path p, the corresponding forwarding paths is denoted by $fwd(p)$
- *nice forwarding path* - like nice path but referring specifically to a forwarding path.
- *class(p)* - is a class of a path p, which is the class of the next-hop peer in p.
- *fine forwarding class* c of some AS A is defined as follows. Let U be the set of all peers of A that announce a path to A (including peers that announce a loopy path to A). Then c is the highest class in the set $\{class(f_A(u)) : u \in U,\ f_A(u)\ is\ finite\}$. If the set is empty, we say that c is null.
- *fine forwarding path* p is a path whose class is the fine forwarding class.
- A path advertised by a peer u to A is called *misaligned* if it contains A but the suffix of u starting at A is not the same as any of A's paths from which

it could have originated. In the case of HBGP, there is a single originating path - the path in the RIB. In the case of YAMR, a default path can only originate from another default path and an alternative path can originate from the default path and from a labeled path with the same label. We also say a *misaligned* advertisement if it contains a misaligned path. If a path is not misaligned, we say it is *aligned*.

Notice that for ASes that don't hide, the next-hop ASes for corresponding forwarding and control paths are the same. Therefore, a control path is nice if and only if the forwarding path is nice. Thus, we can simply talk about a *nice path* without specifying which we mean. Also, we can talk about the *next-hop* without specifying on which path the next-hop is.

LEMMA 8. *Let Z be an instance of HBGP with no dispute wheels and all ASes following widest-advertisement and next-hop policies. Assume that there has been no policy changes and link events for a long enough time that Z has converged. Then, for each AS A, f_A is in the fine forwarding class of A. In particular, if f_A is null, the fine forwarding class is null.*

PROOF. First, consider the case when f_A does not exist. If A does not forward anywhere, either no peer advertises a path to A or there is a path advertised to A that A cannot forward on (if there are multiple such paths, choose the most preferred one). In the first case, we have nothing to show. In the second case, the only reason A cannot forward on a path advertised to it is that this path is loopy. However, because A does not have any path in its RIB, this loopy path must be misaligned and A must be sending disconnection tokens along this path. This contradicts the assumption that Z has converged and concludes the case when f_A is null.

Consider the case when f_A exists. Assume the contrary that f_A is not in the fine forwarding class c of A and let the next-hop along f_A be B. If f_A is not in c, then class of f_A must necessarily be less preferred that c, by definition of c. Let f'_A be a forwarding path of A that is in c and let C be the next-hop along f'_A. Because f'_A is in the more preferred class than f_A, A would prefer any path through C to any path through B. However, since A does not prefer the path p_C advertised by C (the control path whose corresponding forwarding path is f'_A), p_C must be loopy. Moreover, p_C has to be aligned. Indeed, if p_C were misaligned, it would be considered in A path selection, would be preferred over p_B (the control path whose corresponding forwarding path is f_A) and A would send a disconnection token contradicting our assumption that Z converged.

Because p_C goes through A but f'_A does not, there is an AS along f'_A that hides. Let D be the first AS along f'_A starting from A that hides. Then, D prefers the path $p_C(D)$ more than q - the control path corresponding to the suffix of f'_A starting at D. Now, we can identify a dispute wheel.

Let us rename some paths and nodes to illustrate the dispute wheel. Let A be u_0. Let D be u_1. Let the control path corresponding to f_A be Q_0. Let q be Q_1. Let $[A, C, \ldots, D]$, the prefix of f'_A until D be R_0. Let the prefix of $p_C(D)$ until and including A be R_1. Now, the dispute wheel is $W = (U = u_0, u_1, Q = Q_0, Q_1, R = R_0, R_1)$. Verifying that W is indeed a dispute wheel is straightforward. This contradiction finishes the proof of the lemma. □

In the presence of hiding nodes, advertised control paths are different from the forwarding paths. Therefore, ASes don't have enough knowledge to choose the best available forwarding path. However, as the lemma 8 states, the disconnection token mechanism ensure that ASes end up with almost the best available paths. Even though this result is quite promising, under the general assumptions we have been using, the small imperfection the mechanism allows can prevent some ASes from getting a nice forwarding path. To guarantee that each AS gets a nice forwarding path, we have to narrow the class of possible AS policies to those for which these imperfections cannot cause any harm. We call this class of policies *dispute circlet free* policies, or in other words, policies that don't have dispute circlets. Luckily the common customer-peer-provider policies don't have dispute circlets, if we assume that there are no cycles made entirely from provider-to-customer links.

The only difference between dispute circlets and dispute wheels is the path preference relation they use. For dispute wheels, the preference relation is the same relation that ASes use to rank paths based on the ranking function. For dispute circlets, the preference relation R_A of an AS A is a general binary relation that is not necessarily a partial order defined as follows. Given the AS graph, the ranking functions of all ASes, an AS A, and two paths p_1 and p_2 from A to the destination, $(p_1, p_2) \in R_A$ if and only if $class(p_1) \leq class(p_2)$. If $(p_1, p_2) \in R_A$, we write it as $p_1 \succ^A p_2$.

If $\lambda^A(p_1) \geq \lambda^A(p_2)$, $class(p_1) \leq class(p_2)$ and $(p_1, p_2) \in R_A$. Therefore, all preference relations based on ranking function are included in R_A, from which it immediately follows that every dispute wheel is a dispute circlet.

LEMMA 9. *If there are no dispute circlets, each AS in Z has a nice forwarding path.*

PROOF. We first show the following sublemma. Let A be an AS that does not have a nice forwarding path in Z (from now on the specification "in Z" is assumed and we don't write it explicitly). Let $p = [A, A_{n-1}, A_{n-2}, \ldots, A_0]$ be a nice path from A. Then, at least one of $A_{n-1}, A_{n-2}, \ldots, A_1$ does not have a nice forwarding path.

Assume the contrary, that all of the intermediate nodes have nice forwarding paths. There are two possible cases: either A is disconnected or A is connected. If A is disconnected n has to be greater or equal to 2. Consider A_{n-1}. By assumption, A_{n-1} has a nice forwarding path. Because (1) A_{n-1} is willing to advertise a path through A_{n-2} to A (namely p), (2) A_{n-1} has a nice forwarding path, A_{n-1} has to be willing to advertise its control path to A. Therefore, the fine forwarding class of A is non-null. By lemma 8, A is connected. This contradiction proves the sublemma in the case that A is disconnected. Next, we consider the case when A is connected.

In the case that A is connected, we show that the forwarding paths of $A_{n-1}, A_{n-2}, \ldots, A_0$ has to go through A, which is a contradiction because the path of A_0 is $[A_0]$. Let p^n_X and p^c_X denote a nice and the current forwarding paths of X, respectively. Further, let $p(Y)$ denote the suffix of path p starting at Y. First, note that since all A_i's have a nice forwarding path, they all must be connected. Because A_{n-1} forwards through a nice peer, it has to advertise its control path to A. If $p^c_{A_{n-1}}$ does not go through A, A's fine forwarding class is equal to the nice forwarding class. By lemma 8, A's forwarding path is a nice forwarding path. This is a contradiction. Therefore, $p^c_{A_{n-1}}$ has to go through A.

Assume $p^c_{A_{n-2}}$ does not go through A. Because it does not go through A, it does not go through A_{n-1}. Then, A_{n-2} is advertising $p^c_{A_{n-2}}$ to A_{n-1}. Because $p^c_{A_{n-2}} \neq p^c_{A_{n-1}}(A_{n-2})$ (one goes through A and one does not) and because A_{n-1}'s current forwarding path is $p^c_{A_{n-1}}$, $[A_{n-1}]p^c_{A_{n-2}} \prec^{A_{n-1}} p^c_{A_{n-1}}$. Recall that because A's current forwarding path is not nice, $[A, A_{n-1}]p^c_{A_{n-2}} \succ^A p^c_A$. These four paths and nodes A and A_{n-1} form a dispute circlet, which we assumed does not exist. Thus, $p^c_{A_{n-2}}$ goes through A.

The argument above can be repeated inductively. At the step for A_{n-i}, the following holds:

$$[A_{n-i+1}]p^c_{A_{n-i}} \prec^{A_{n-i+1}} p^c_{A_{n-i+1}}$$
$$p^c_A \prec^A [A, A_{n-1}, \ldots, A_{n-i+1}]p^c_{A_{n-i}}$$

and these four paths together with A and A_{n-i+1} form a dispute circlet. This finishes the proof of the sublemma. Next, we prove the lemma.

Assume the contrary, that there exists an AS A_0 whose forwarding path is not nice. Then, pick a nice path $p_1 = [A_0, B_{n-1}, B_{n-2}, \ldots, B_0]$ from A_0 to the destination. On this path, pick a node B_i, $0 \leq i \leq n-1$ such that

- B_i's forwarding path is not nice
- Path $[B_i, B_{i-1}, \ldots, B_0]$ is not a nice forwarding path of B_i.

Such B_i can be found in the following way. By the sublemma, there is an AS B_j whose forwarding path is not nice. If for this B_j, the path $[B_j, B_{j-1}, \ldots, B_0]$ is nice, we can apply the sublemma to B_j and its nice path $[B_j, B_{j-1}, \ldots, B_0]$, to find a node B_k, $k < j$, whose forwarding path is not nice. If the path $[B_k, B_{k-1}, \ldots, B_0]$ is a nice forwarding path of B_k, we can continue analogously. Because the original path $[A_0, B_{n-1}, B_{n-2}, \ldots, B_0]$ is finite and on each iteration it gets smaller, this process cannot continue forever. Therefore, there exists such a B_i. Rename it to A_1.

Define $A_2, A_3, \ldots$ analogously until some A_q is not the same as a previously found A_r. Without loss of generality, assume that $r = 0$ (because we could have started at A_r). Let p_i be the nice path of A_i we used to find A_{i+1}. Interpret the subscripts module q and let R_i be the prefix of the path p_i until and including A_{i+1}. Let Q_i be the suffix of the path p_{i-1} starting from and including A_i. Let $R = R_0, R_1, \ldots, R_{q-1}$, $Q = Q_0, Q_1, \ldots, Q_{q-1}$, and $A = A_0, A_1, \ldots, A_{q-1}$. Then, $W = (A, Q, R)$ is a dispute circlet. This contradiction, proves the lemma. □

To show theorem 5, we simply apply lemma 9 to default path of YAMR.

G. HIDING RECOVERY

The Theorem 6 is obvious because of the failed link propagation mechanism. Recall that having a failed link information gives an AS a permission to hide the failure. When the link recovers, all failure information is guaranteed to be withdrawn from every AS that had it. When the failure information is withdrawn, all ASes stops hiding and routing returns to normal.