Differentiated BGP Update Processing for Improved Routing Convergence

Wei Sun, Zhuoqing Morley Mao, Kang G. Shin
University of Michigan
{*wsunz, zmao, kgshin*}*@eecs.umich.edu*

Abstract— **Internet routers today can be overwhelmed by a large number of BGP updates triggered by events such as session resets, link failures, and policy changes. Such excessive updates can delay routing convergence, which, in turn, degrades the performance of delay- and jitter-sensitive applications. This paper proposes a simple and novel idea of *differentiated processing* of BGP updates to reduce routers' load and improve routing convergence without changing the protocol semantics. Based on a set of criteria, BGP updates are grouped into different priority classes. Higher-priority updates are processed and propagated sooner, while lower-priority ones, not affecting routing decisions, can be delayed to both reduce routers' load and improve routing convergence. We first present a general methodology for update classification, update processing, and priority-state inference. By analyzing real BGP data obtained from Route Views, we show that our update classification is feasible and beneficial. We further propose two differentiated update processing (DUP) algorithms and evaluate them using the SSFNet BGP simulator on several realistic network topologies. The algorithms are shown to be very effective for large networks, yielding 30% fewer updates and reducing convergence time by 80%. Our scheme is simple and light-weight with little added processing overhead. It can be deployed incrementally, since BGP messages are not modified and every BGP router makes routing decisions independently.**

I. INTRODUCTION

Real-time, multimedia applications such as IPTV, VoIP and Internet gaming are becoming popular. These delay- and jitter-sensitive applications impose more stringent requirements on the underlying Internet routing system. In light of this trend, BGP (Border Gateway Protocol) routing issues have attracted significant attention from both the research and operator communities. A key problem associated with BGP is the excessive number of BGP updates possibly triggered by routing changes, such as session resets, link failures, and policy changes. For example, a recent study of a large tier-1 ISP shows that within just one minute, a "rich peering" router can experience hundreds of routing updates all at once partly due to the interaction between intra- and inter-domain routing [1].

There are several well-known schemes deployed to address this problem, including Minimum Route Advertisement Interval (MRAI), flap damping [2], Sender Side Loop Detection (SSLD), and Withdrawal Rate Limiting (WRATE) [3]. MRAI is a rate-limiting mechanism, enforcing a minimum inter-update interval between two neighbors (and for a specific destination prefix), in the hope that such a delay may help consolidate multiple related updates into fewer updates. Flap damping targets longer-term unstable routes, blocking routes changing too frequently over a relatively longer time period. Using a path-vector routing approach, BGP routers detect routing loops by checking if its own AS number appears in the AS path upon receiving a new route. SSLD, on the other hand, detects routing loops before sending the route to a neighbor BGP router. A rate-limiting mechanism, such as MRAI, is usually applied only to announcements, but not to withdrawals. However, some router vendors implement WRATE by applying MRAI to withdrawals as well, even though this is not recommended [3].

A related issue is the long convergence time caused by BGP path exploration. The authors of [4]–[6] have shown that the BGP convergence time is surprisingly long and depends on the length of the longest backup path. The convergence time is also shown to be proportional to the number of alternative routes to a given destination [7]. The prevalence of multi-homing in AS relationships (e.g., a customer AS peering with multiple providers ASes) [8]–[10] increases the number of backup routes in the Internet significantly, which, in turn, prolongs BGP convergence. Our recent study shows that the convergence time for one BGP Beacon [11] prefix is still surprisingly long—more than 30 minutes.

All the existing mechanisms, including MRAI and flap damping, are intended for all updates, except that in general, withdrawals are not subject to the influence of MRAI. In this paper, we introduce the concept of *differentiated BGP update processing*, which classifies BGP updates and treats them according to their importance, which determines the update sending order and delay. We observe that BGP updates can be divided into two classes: the first class affects the routing decisions of the receiving routers, possibly triggering more updates, while the second class does not affect the routing decisions of the receiving nodes, i.e., the best routes used are not changed. We regard the first class more important; but it is non-trivial to determine which updates belong to which class, since the routing decision and local policy of the receiving nodes are not directly available. We first define a method for classifying BGP updates. Updates in different classes will then be processed with different priorities. We also explore ways to process updates differently and propose two *Differentiated Update Processing* (DUP) algorithms.

In summary, we propose differentiation of updates depending on whether they are used in the forwarding tables of routers for related destination prefixes. If they are, they will more likely be processed with higher priority. The key ideas of our DUP algorithms are: (i) *Locally inferred routing preference*: when sending updates to a neighbor, a BGP router checks if the neighbor has sent updates for the same prefixes to itself. If so, it sends the updates with low priority. (ii) *Difference-based route selection*: when failure occurs and the best route to a destination is withdrawn, instead of selecting the next best available route, a BGP router first selects an interim route that shares the shortest common sub-route with the withdrawn route. The intuition behind the first idea is that if the neighbor also advertises a route, it must have an alternate route. The justification behind the second idea is that usually routes dissimilar to the withdrawn route are more likely valid during convergence.

The proposed scheme reduces the number of low-priority updates and the routing convergence time, thus reducing router (message processing, bandwidth) overhead, especially at an overloaded router in the core with rich peering. It reduces convergence time not only after a failure, but also during a new route propagation. At the same time, the scheme does not require any change to BGP protocol semantics (including the format of BGP messages and the final best route selection), thus facilitating incremental deployment. Moreover, it does not compromise reachability.

The authors of [12] proposed a Routing Control Platform (RCP), which uses a centralized routing control server to make route selection on behalf of each BGP router within a single AS, and distributes the routing decision to it. This RCP is to replace IBGP (Internal BGP) and solve many problems caused by its inefficiency. The authors showed that a prototype of such a system can be effectively implemented on a software router. Our scheme is simple and light-weight, and can be integrated into the RCP platform. Also, it can be easily implemented on software routers running XORP [13] or Zebra [14].

The rest of the paper is organized as follows. Section II discusses related work on BGP routing. Section III presents the general idea of DUP and updates classification. Section IV presents potential benefits of DUP using the Route Views data. Based on the general framework, Section V introduces the basic DUP algorithm. In Section VI the basic algorithm is combined with a new route selection algorithm to further reduce the convergence time and the number of updates. Section VII evaluates the DUP algorithms using simulation and compares their performance with the current BGP protocol and other BGP improvement schemes. Finally, Section VIII concludes.

II. Related Work

It is reported in [1], [15] that the current BGP may generate an excessive number of updates, especially when the network is overloaded, and the convergence of BGP may take too long to meet the requirements of real-time applications.

A. BGP Processing Overhead

The authors of [9] analyzed the rapid growth of BGP routing tables, as a result of several factors, such as load-balancing, the prevalence of multi-homing of small networks, and address fragmentation. The authors of [8] pointed out that not only is the Internet growing fast in size, it also becomes densely meshed at the inter-AS level. All these changes increase the BGP routing table size, thus increasing routers' processing overhead.

Studies have also shown that under certain (abnormal) conditions, there could be an excessive number of BGP updates. For instance, the authors of [16] studied the BGP behavior under the Slammer worm outbreak, and showed that during the attack, the number of BGP updates increased ten-fold, compared to that under normal condition. For some prefixes, the increase was by about 100 times. The authors of [15] studied the BGP behavior under heavy load and observed several weaknesses of the current BGP: its sensitivity to data congestion, global propagation of small local changes, and slow convergence. The authors of [17] observed that in the time scale of minutes, BGP updates do not affect router's CPU load significantly, but in a shorter time scale of seconds, BGP can consume up to 100% of CPU cycles. The authors of [18] also showed that high update rates from multiple peers are harmful, prolonging the transit times of packets.

In addition to MRAI and flap damping, BGP Graceful Restart [19] is another mechanism deployed today that can reduce updates, but has limited applicability as it works only for short-lived session resets.

B. BGP Convergence Time

Using simulation, the authors of [20] demonstrated the effectiveness of MRAI in reducing the convergence time. They also observed that for a given topology, there exists an optimal MRAI timer value which minimizes the convergence time. However, the optimal value depends on the topology, so there is no universal optimal setting for the MRAI timer applicable for all routers and all types of routing changes.

In practice, Cisco routers use 30 and 5 seconds as the default MRAI timers for EBGP and IBGP sessions, respectively, while Juniper routers disable MRAI timer by default [21]. The study above showed that disabling MRAI timer may lead to large number of updates and long convergence time.

To reduce BGP update traffic and the associated overhead, and to decrease network convergence time, the authors of [22] proposed to add some *consistency assertion* checks to BGP update processing. From this checking, many updates are observed to contradict one another, and not all of them are valid. A set of assertion rules are defined to check the validity of updates, and block propagation of information on those routes that violate these rules, to other BGP peers. They have shown that by employing these rules, the number of BGP updates can be significantly reduced, and the route convergence time can be improved drastically.

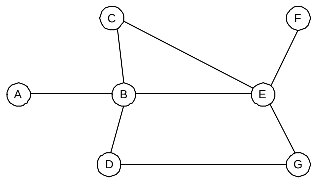
The authors of [23], [24] extended the idea in [22] by embedding the root cause of a failure in updates, so a receiving node knows which candidate routes in its routing table are invalidated by the root cause, dramatically reducing the number of invalid routes and hence the convergence time. However, this scheme requires modification of BGP updates to embed the root cause information and slows deployment.

The authors of [25] proposed a different algorithm called "Ghost Flushing." The routes invalidated by a failure are called "Ghosts." For speedy removal of such invalid routes from the network, when a route is replaced by a less preferable one, if the route cannot be propagated because the MRAI timer has not expired, a withdrawal is sent immediately. By sending extra withdrawals, it is shown that invalid routes are removed much sooner and the convergence time is greatly reduced,

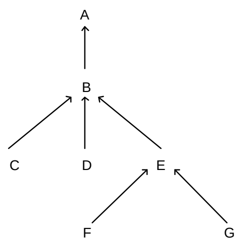
TABLE I

COMPARISONS OF BGP-ENHANCEMENT SCHEMES

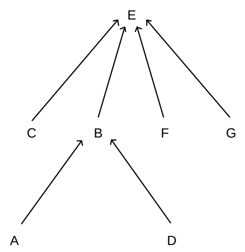
	Consistency assertion	**Root-cause based schemes**	**Ghost flushing**	**Our scheme**
change format of BGP messages	yes	yes	no	no
introduce extra messages	no	no	yes	no
reduce convergence time of updates for new prefixes	no	no	no	yes



(a) Network topology



(b) Forwarding-path tree for AS A



(c) Forwarding-path tree for AS E

Fig. 1. Forwarding-path tree of an AS

especially when the original route is the only way to reach the destination—no valid alternative routes exist after the failure.

Table I compares our scheme with three existing schemes mentioned above. Our scheme does not require any modification to the BGP protocol semantics such as BGP message format, nor does it send extra messages. Moreover, unlike the above schemes which mainly focus on reducing convergence time after a failure, our scheme can also reduce convergence time during a new route propagation.

The author of [26] discusses various schemes (at different layers of network hierarchy) to achieve sub-second convergence and maintain high routes availability. The mechanisms capture the same differentiated processing idea as ours, but work mainly at the intra-domain level. In contrast, our scheme focuses on the inter-domain level.

III. GENERAL METHODOLOGY

A. Assumptions and Notations

Before delving into the details of our proposed approach, we introduce the following widely-used assumptions. Violation of these assumptions can be accommodated, especially in a single network where the policy information is known.

A1. In a peer-peer AS relationship, if ASes A and B are peers, A only sends B updates pertaining to itself and its customers; so does B. Routing updates learned from one peer will not be forwarded to other peers.

A2. In a customer-provider AS relationship, if A is a customer of B, A sends B only the updates pertaining to itself and its customers; B sends A the updates learned from all neighboring ASes.

A3. A BGP router prefers routes learned from customer to those learned from peers; it also prefers routes learned from peers to those learned from providers.

For simplicity, in addition to A3, we also assume that routing decisions are based on the AS path length by preferring shorter paths.

The following notation is used throughout the paper. For a given BGP update, the sending router is called *sender*, and the receiving router *receiver*. If a router has multiple routes to reach a given destination AS, the one currently used is called *primary route*; other alternative routes are called *backup routes*. To avoid confusion, the term *peers* is used to indicate the two ASes with a peer-peer relationship between them, while neighboring BGP routers/ASes are called *neighbors*.

B. Per-prefix Forwarding-Path Tree

Before discussing BGP update classification, we introduce the concept of per-prefix *forwarding-path tree*[1]. When all the BGP routers in a network reach steady state, for a given destination prefix, at router level there is a forwarding-path tree inside the network. The destination itself is the root of the tree, and its immediate neighbors are the first-level children, and so on. Directional links between different levels of nodes are trunks of the tree. Data packets heading for this destination prefix flow from the leaves to the root, along the trunks; while routing updates flow in the opposite direction, from the root to the leaves (Fig. 1). If updates are received through existing tree trunks, then we call them "on-tree" updates; otherwise, they are called "off-tree" updates. For each tree node, there is only one trunk reaching it from its parent node. Note that the tree structure is dictated by the routing in the network; when routing changes, the tree structure changes accordingly.

The key observation is that a BGP router can have many neighbors, thus receiving many updates regarding alternative routes to a given destination. However, for each destination prefix, there is only one on-tree update (regarding its primary route) from its parent node. Other updates are off-tree updates (regarding backup routes). For example, seven ASes are connected as shown in Fig. 1(a), each node representing an AS and also a BGP router. The forwarding-path trees for ASes A and E are illustrated in Figs. 1(b) and (c), respectively. In both trees the D-G link is an off-tree link.

In the current BGP, children nodes on the forwarding-path tree always know their parents, while parent nodes have no information about their children nodes (or no such information is used even if it is known at all). Our scheme attempts to gather such information and use it for BGP update processing.

[1]Here we assume that a router always send traffic for one prefix to the same neighbor. If this does not hold, the resulted structure would be a forwarding-path DAG (Directional Acyclic Graph). But this does not affect the following analysis.

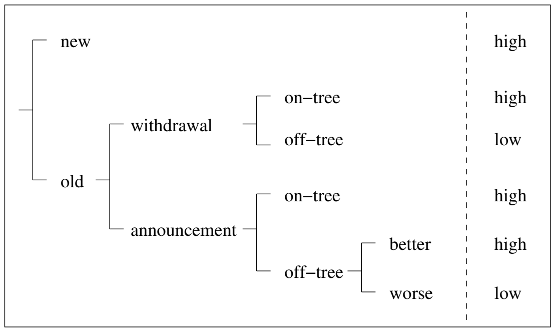


Fig. 2. BGP update classification

C. Update Classification

The BGP update classification method is shown in Fig. 2. The updates are classified from the receiver's point of view. If the updates are about new destinations, which the receiver did not know before, they belong to a high-priority class. Otherwise, the updates are classified depending on whether they are on-tree in the forwarding-path tree of the associated destination prefix. We consider on-tree updates to be more important as they affect routing decisions.

As shown in Fig. 2, when an update is a withdrawal, then if it is on-tree, meaning that it withdraws a primary route of the receiver, then the update has high priority; otherwise, it has low priority since it withdraws a backup route which is not used by the receiver. When an update is an on-tree announcement, it implicitly replaces the primary route with a new route. Thus the update has high priority. If the update is off-tree, it has high priority only when it contains a better route than the primary route; otherwise, the update has low priority—since the route is not better than the primary route and will not be chosen.

D. Update Processing

Once updates are classified, the next question is how to differentiate their processing based on their priority class.

1) Receiver side—priority queues: Since the update classification is done from a receiver's perspective, the natural way is to process BGP updates using a priority queue on the receiver side. The advantage is that the receiver knows which class an update belongs to, by simply checking its forwarding table. However, to classify the updates into different priority queues, a BGP router has to check the content of the updates and do some pre-screening. This checking and pre-screening in a large part is repeating the default BGP update processing, thus incurring additional overhead. Also, this receiver-side method alone does not directly reduce the number of updates, as all updates are still transmitted to the receiver. Because of this, we focus on the sender-side scheme in the following discussion.

2) Sender side—different delays: An alternative way is to differentiate updates at the sender, using different delay timers (such as MRAI) for different classes of updates. There are two basic methods with different benefits. The first method is to process high-priority updates with default timers, while processing the low-priority ones with longer timers. This method can potentially reduce the number of low-priority updates transmitted; the second method is to process the low-priority updates with default timers, while processing the high priority ones with shorter timers. This method can potentially reduce the convergence time, since the high-priority updates experience a shorter delay at each hop.

However, since update classification is done from the receiver's perspective, the sender has no direct way to tell whether an outgoing update will affect the receiving router's forwarding table (or whether the update is an on-tree update for the receiver). Thus, the sender has to infer the class of updates. This class inference may incur some overhead on the sender side.

E. Update Class Inference

As mentioned above, when the sender side is involved in differentiated update processing, it has to infer the class of each update for the receiver. Discussed below are possible ways to achieve correct inference.

First, a router can infer the information externally: it can obtain the information from the data it receives from other routers, either implicitly (e.g., monitoring the data traffic passing through to determine if a neighbor is sending data packets to the related destination through it) or explicitly (e.g., letting the receivers of its updates send some feedback messages saying if the updates are being used as their primary routes). Note that inside a single AS, and hence for IBGP sessions, this information can be trivially obtained since the routing policy is consistent inside an AS.

A router can also infer the information internally, e.g., by checking its configuration and/or routing table. For example, when a router R has an update for a destination prefix, it can examine if there is a route entry for the same prefix received from a neighbor in the routing table. If there is, then the neighbor is using a different route; otherwise, it is using the route learned from R. However, if routers filter out some outgoing updates based on local policies, then this method will overestimate the number of high priority updates. On the other hand, this scheme does not cause any reachability problem.

Since the external inference methods in general incur more overhead and require extra memory to store the inferred data, in the following discussion, we focus on internal (or local) inference.

IV. Empirical Data Analysis

To examine empirically the amount of BGP updates that can be classified as low priority, we analyze the routing data collected by the Route Views project [27]. Each Route Views router peers with BGP routers in many ASes to collect BGP routing data. The data are collected in two forms: RIB files and update files. The RIB files contain the contents of the forwarding tables of all the BGP neighbors, and are collected every two hours; the update files contain new updates from the BGP neighbors every 15 minutes. By analyzing the Routeviews data, we found that (i) excluding duplicates, a large proportion of announcements (about 50% on average) can be classified as low priority, and thus processed separately. This confirms the value of our DUP scheme; (ii) a significant portion of withdrawals are low priority as well, meaning that

TABLE II

PSEUDOCODE OF DUP ALGORITHM: DUP_SEND()

ASP_1^{pre}:	AS path of the previous route for D sent to AS_2;
ASP_1^{new}:	AS path of the newly-chosen route for D;
ASP_2:	AS path of the route for D received from AS_2;
len ():	length of an AS path;

```
01: // AS_1 sends an update for D to AS_2:
02: If $ASP_2$ = Null, Then
03:    sends the update with shorter MRAI timer;
04: Else
05:    If $AS_1$ is a peer or provider of $AS_2$, Then
06:       sends the update with longer MRAI timer;
07:    Else // $AS_1$ is a customer of $AS_2$
08:       If $ASP_1^{pre}$ = NULL, Then
09:          sends the update with shorter MRAI timer;
10:       Else
11:          If len ($ASP_1^{pre}$) $\neq$ len ($ASP_1^{new}$) And
12:             len ($ASP_1^{new}$) + 2 $<$ len ($ASP_2$), Then
13:             sends the update with shorter MRAI timer;
14:          Else
15:             sends the update with longer MRAI timer;
```

they are withdrawing backup, not primary routes. Due to the space constraint, the details of the data analysis is omitted. See an extended version [28] for details.

V. DUP: DIFFERENTIATED UPDATE PROCESSING

Based on the general discussion in Section III, we design a *Differentiated Update Processing* (DUP) algorithm based on the sender-side scheme. To reduce overhead, a sender infers the priority of an update based on its local information. Also, since this inference is not always accurate, all the withdrawals are still treated the same, without any differentiation.[2] The algorithm takes the following two steps.

Step 1: it checks the AS relationship between a local AS_1 and a neighboring AS_2, and infers whether AS_2 is using AS_1's route to forward traffic. For example, if AS_1 and AS_2 have a peer-peer relationship, then if AS_1 has no route from AS_2 (meaning that AS_2 has no route at all or has a route via its provider/peer AS), the route has high priority; else, the route has low priority. (AS_2's current route must be via its own customer AS, which is favored over AS_1's route since AS_1 is a peer AS of AS_2.)

If the AS relationship alone can not decide the priority of AS_1's new route, it goes to Step 2 to compare the lengths of the two routes.

Step 2: the lengths of AS_1's new route and the route from AS_2 are compared. If from AS_2's point of view, the new route is shorter than AS_2's current route, then it has high priority; else, it has low priority.

The details of the algorithm are listed in Table II. Note that (i) in the algorithm we assumed known AS relationship information between a BGP router and its neighbor, which can be easily stored in a router as a configuration parameter or directly inferred from the configurations; (ii) the AS paths' length comparison is made from AS_2's point of view: if at AS_1 the two paths have length len (ASP_1^{new}) and len (ASP_2), then at AS_2 the lengths become len (ASP_1^{new})+1 and len (ASP_2)-1; (iii) two MRAI timers are used in the DUP algorithm (one more than the current BGP protocol for each BGP neighbor), one for high-priority updates ($MRAI_{short}$) and the other for low-priority updates ($MRAI_{long}$). If timer $MRAI_{long}$ expires, the router will send low-priority updates and reset the timer; if timer $MRAI_{short}$ expires, the router will send high-priority updates and reset the timer. Depending on the purpose of the algorithm, two options can be implemented. The first option is to use the default MRAI value for $MRAI_{short}$, and a larger value for $MRAI_{long}$. The goal is to reduce the number of BGP updates exchanged between BGP routers, by holding low-priority updates longer. The second option is to use the default MRAI value for $MRAI_{long}$, and a smaller value for $MRAI_{short}$. The goal is to shorten the convergence time of BGP in the network, by speeding up the propagation of high-priority updates. In this paper we focus on this second option.

[2]In practice, withdrawals generally are not subject to the control of MRAI timer anyway.

TABLE III

PSEUDOCODE OF SIMPLER DUP: DUP_SEND()

ASP_2:	AS path of the route for D received from AS_2;

```
01: // $AS_1$ sends an update for D to $AS_2$:
02: If $ASP_2$ = Null, Then
03:    sends the update with shorter MRAI timer;
04: Else
05:    sends the update with longer MRAI timer;
```

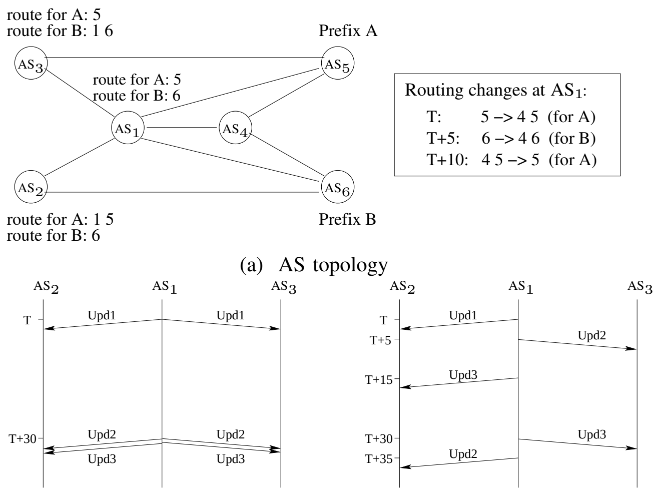
A. Simpler DUP

The DUP algorithm above takes advantage of the knowledge of AS relationship between a BGP router and its neighbor when classifying the priority of an update. The classification process appears complex and may not be always accurate, as ASes do not always choose routes based on the guideline of AS relationships.

To overcome this issue, we introduce a simpler version of DUP without relying on the knowledge of AS relationships. It considers one thing only: whether a neighbor has already propagated route for the same prefix to the local AS. If the neighbor has not already done so, the update has high priority; otherwise, it has low priority. The details of the algorithm are listed in Table III.

B. Priority misclassification

Since sender-side scheme is used, neither the DUP algorithm nor its simpler version matches exactly the update classification method in Fig 2. If a sending node has not received a route from its neighbor, it means (i) the route is for a new prefix; or (ii) the neighbor is already using the local AS's route; or (iii) the neighbor is using another route without notifying the local AS. While in the first two cases the outgoing update is on-tree, in the third case it is clearly off-tree. Thus our algorithms may set an update's priority higher and send it earlier than necessary. On the other hand, if the sending node has already received a route from the neighbor, then the outgoing update must be off-tree. The question is whether it is better than the neighbor's current route from the neighbor's perspective. The simpler version simply set the update as low priority, thus may delay the update longer than the default BGP. The DUP algorithm is more accurate, classifying the priority based on the AS relationship between



(a) AS topology

(b) Update sequence under default BGP (c) Update sequence under DUP

Fig. 3. Illustration of DUP algorithm

the two neighbors and the AS path lengths of the new route and the route from the neighbor. However, if some ASes do not follow the guidelines of AS relationships discussed in Section III, it may misclassify the priority also. On the other hand, since in this case the neighbor already has a route, it does not affect the neighbor's connectivity. The only drawback is that the neighbor may use a less preferred route slightly longer. In addition, since we focus on speeding up the high priority update while sending the low priority ones using the default MRAI value, this is not an issue at all: low priority updates are not delayed longer than the default MRAI.

This simpler version does not consider AS relationships, nor does it compare AS path lengths. Thus it incurs less overhead. From simulation studies, it works quite well. Therefore, in the following discussion, we use the simpler version to represent the DUP algorithm.

The following example illustrates the advantages of our DUP algorithm.

Example 1: As shown in Fig. 3, both AS_2 and AS_3 are connected to AS_1. Prefix A is located in AS_5; prefix B is located in AS_6. Suppose to reach destination A, AS_2 uses AS_1's path, while AS_3 does not; to reach destination B, AS_3 uses AS_1's path, while AS_2 does not. Now suppose AS_1 first changes its path to A at time T, then changes its path to B at T+5, and changes its path to A again at T+10. Under the current BGP, AS_1 uses one MRAI timer for each neighbor. It sends update for the first change (Upd_1) to both AS_2 and AS_3 at T, and sends updates for the second and third changes (Upd_2 and Upd_3) to both AS_2 and AS_3 at T+30 (after the MRAI timers expire). Under the DUP algorithm, in contrast, AS_1 uses two MRAI timers for each neighbors. It sends Upd_1 to only AS_2 at T: since AS_3 is not using AS_1's route, the update to AS_3 has low priority and thus is delayed; then for the same reason, at T+5 (not T+30: since AS_1 does not send Upd_1 to AS_3 at T, the MRAI timer for high-priority update is not reset then) it sends Upd_2 to AS_3 only. At T+10, Upd_3 replaces Upd_1, so Upd_3 is sent to AS_2 at T+15 after the MRAI timer for high-priority update expires. Upd_3 is sent to AS_2 at T+30 after the MRAI timer for low-priority update expires, and Upd_2 is sent to AS_3 at T+35 after the MRAI timer for low-priority update expires. Therefore, the DUP algorithm not only sends important updates (Upd_2 to AS_3 and Upd_3 to AS_2) sooner, it also reduces the number of updates: under DUP, only five updates are sent. Since Upd_2 (Upd_1) is not used by AS_2 (AS_3), its delay does not affect routing decision.

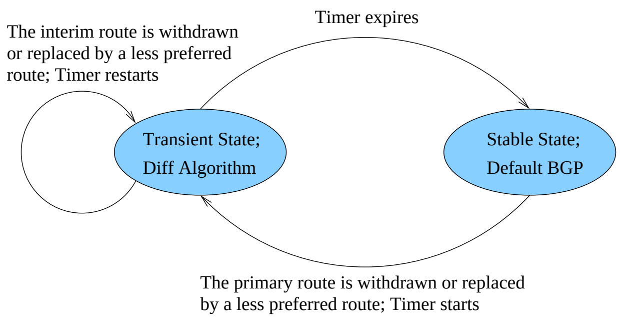

Timer expires
The interim route is withdrawn
or replaced by a less preferred
route; Timer restarts
Transient State;
Diff Algorithm
Stable State;
Default BGP
The primary route is withdrawn or replaced
by a less preferred route; Timer starts

Fig. 4. Transition diagram between Stable and Transient states

VI. DUP+: Enhanced DUP

Note that in the above discussion, we assumed that all the updates contain valid routes. While this assumption holds during the propagation of new routes, it does not hold in case of a network component failure which may invalidate some existing candidate routes, and these invalid routes may propagate through the network during the transient period after the failure. This propagation of invalid routes caused by BGP path exploration is the main culprit for the extraordinarily long convergence time after a failure, unnecessarily triggering excessive routing changes at the same time.

Also, since invalid routes are propagated, the actual best route after a failure may not be shorter than some invalid routes being sent/received earlier; picking shorter routes only as the high-priority ones (as in the Step 2 of the DUP algorithm) is suboptimal. The update classification needs to be improved. Ideally we can distinguish valid routes from invalid ones, so that valid routes can receive higher priority than invalid routes, thus speeding up the convergence.

A. Difference-Based Route Selection

To reduce the convergence time and the number of updates exchanged after a network failure, we propose a new route-selection algorithm that selects a new route based on the difference between the candidate routes and the original (withdrawn) route: when the original route is withdrawn or replaced by a less preferred route (e.g., due to a network failure), the selected route is the shortest one with the *maximum difference* from the original route, which is not necessarily the best of the remaining routes.

As shown in Fig. 4, we define transient and stable states for a prefix. After an original route is withdrawn or replaced by a less preferred route (usually resulting from a network failure), the router enters the transient state. In this state, the routing decision process for the prefix does not select the best remaining route; instead, it selects an interim route with the minimum similarity to the original route—the one that shares

TABLE IV

PSEUDOCODE OF DIFF ALGORITHM

ASP:	AS path of route R;
P:	destination prefix of route R;
ASP_{lcs}:	susceptible AS path segment;
lcs (asp_1, asp_2):	longest common subsequence of two AS paths;

```
01: // Route $R$ is withdrawn or replaced by a less preferred route :
02: If $P$ is in default stable state, Then
03:     $ASP_{lcs}$ = $ASP$;
04:     start the timer $t_p$;
05: Else
06:     $ASP_{lcs}$ = lcs ($ASP_{lcs}$, $ASP$);
07:     restart the timer $t_p$;
08: select the shortest route sharing the minimum LCS with $ASP_{lcs}$.
```

the shortest common AS path segment with the original one. Then, the AS path of the original route is stored as *susceptible AS path segment*, since the failure occurred to this path. At the same time, a timer is started to indicate how long the router has stayed in the transient state. When the timer expires, the router switches from the transient state to the stable state, and the best route is recomputed using the default BGP route-selection algorithm. However, if the existing interim route is withdrawn or replaced again during the transient state, (i) the timer is restarted, and (ii) the existing interim route is used to update the susceptible AS path segment. This new algorithm used during the transient state is called *Diff*.

The details of the Diff algorithm are given in Table IV. Currently, the length of the timer (t_p in the code) is set to 1.5 times the MRAI timer value. The idea is to set it long enough to catch consecutive updates from a single BGP neighbor. Also, during the same transient state, if the interim route is withdrawn or replaced by a less preferred route, the LCS (longest common subsequence) of the interim route and the susceptible AS path segment is computed (recursively) to get the new susceptible path segment, which has the effect of narrowing down the location of the failure. A variable ASP_{lcs} is used to store the susceptible AS path segment.

The intuition behind Diff is that during the transient state, the routes are not stable and many invalid routes are propagated. These invalid routes are closer and more similar to the original routes; potential valid routes are more different from the original routes. Thus, during the transient state, we select an interim route which is more likely to be valid. Although this route is not necessarily the best remaining route, as long as it is valid, it guarantees the reachability of the destination. The propagation of this valid route also helps other routers to converge to a valid route faster. After the transient state is exited, most (if not all) of the invalid routes are removed from the routing table, and a new best route can be selected. Therefore, the Diff algorithm skips invalid backup routes and selects the shortest one with the largest difference from the original route, which is more likely to be a valid route. This significantly shortens the path exploration process and generates fewer routing updates.

Suppose the original route is $\{AS_1 \dots AS_k\ AS_{k+1} \dots AS_m\}$, and a failure occurs between AS_k and AS_{k+1}. Then, all the invalid routes after the failure contain the path segment $\{AS_k\ AS_{k+1} \dots AS_m\}$. By using difference-based route selection, the new algorithm potentially chooses a valid route faster, favoring routes that do not contain that path segment. On the other hand, a route containing segment $\{AS_k\ AS_{k+1} \dots AS_m\}$ is not necessarily an invalid route, since there could be multiple peering sessions between AS_k and AS_{k+1}.

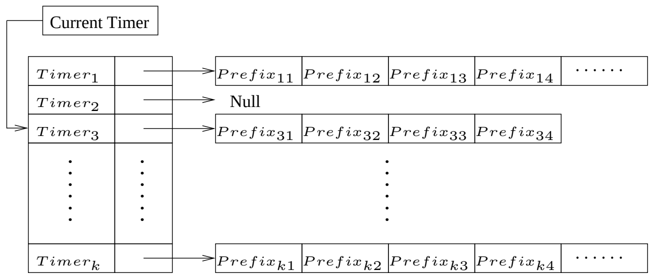


Fig. 5. The global timer structure

Note that the Diff algorithm uses two criteria to select an interim route: maximum difference (which is equivalent to shortest LCS) and shortest AS path length: it chooses the route that shares the shortest LCS with the original route first. If multiple routes share the same shortest LCS, then the one with the shortest AS path will be chosen. In addition, the LCS algorithm here is different from the classic LCS algorithm in that the resulting common subsequence must start from the origin of the AS paths and must contain contiguous ASes. For example, given two routes, $\{AS_4\ AS_3\ AS_2\ AS_1\}$ and $\{AS_3\ AS_4\ AS_2\ AS_1\}$, both originating from AS_1, according to our algorithm, their LCS is $\{AS_2\ AS_1\}$, not $\{AS_3\ AS_2\ AS_1\}$ or $\{AS_4\ AS_2\ AS_1\}$. Our algorithm compares the AS numbers at the corresponding locations only, starting from the original AS.[3] Thus the computation complexity is linear with respect to the lengths of the AS paths.

Since Diff only changes how a route is selected, all the other mechanisms of the default BGP still remain, including loop detection. Therefore, the routers will converge under Diff if they do under the default BGP.

1) Global timer scheme: In the discussion above, we assume that each prefix has a separate timer. We call it *individual timer scheme*. Although each timer is created on demand—only when a prefix is affected by changes in the network, and it is deleted whenever the routes for the prefix converge, the scheme may still incur significant overhead in practice when a failure affects a large number of prefixes.

To reduce the overhead incurred by the timers, we design a new *global timer scheme* that deploys a fixed number of global timers for all the prefixes. As shown in Fig. 5, the scheme works as follows: when a BGP router starts, it automatically starts K timers. The expiration time of the timers are set as $\frac{T}{K}, 2\frac{T}{K}, 3\frac{T}{K}, ..., (K-1)\frac{T}{K}$, and T, respectively, where T is the full length of an expiration period. The gap between two neighboring timers' expiration time is $\frac{T}{K}$. There is a pointer locating the current timer, which is always the one with the longest expiration time. For each timer, there is also a queue associated with it. Whenever a prefix is affected by a change in

[3]If prepending is used by the two routes, redundant AS numbers must be removed from their AS paths before the comparison.

TABLE V
PSEUDOCODE OF DUP+ ALGORITHM: SENDER SIDE

ASP_1^{new}:	AS path of the newly-chosen route for prefix D;
ASP_2:	AS path of the route for D received from AS_2;
ASP_{lcs}:	susceptible AS path segment;
lcs (asp_1, asp_2):	longest common subsequence of two AS paths;

```
01:  // $AS_1$ sends an update for D to $AS_2$:
02:  If the prefix is in stable state, Then
03:     call DUP_Send();
04:  Else
05:     If $ASP_2$ = Null, Then
06:           sends the update with shorter MRAI timer;
07:     Else
08:        If lcs ($ASP_1^{new}$, $ASP_{lcs}$) $<$ lcs ($ASP_2$, $ASP_{lcs}$), Then
09:           sends the update with shorter MRAI timer;
10:        Else
11:           sends the update with longer MRAI timer;
```

the network, it is put in the queue associated with the current timer until the timer expires.

When a timer expires, (i) if its queue is not empty, all prefixes in the queue go through the route recomputation process to select the best routes, after which they are removed from the queue; (ii) the timer is reset with length T; (iii) the current timer pointer points to this newly reset timer.

Since the current timer is updated whenever a timer expires, its expiration time is always about T time away (more accurately, the time value is between $\frac{(K-1)T}{K}$ and T). Also, a prefix is always put in the queue of the current timer. Therefore, for each prefix, its timer will expire in t, where $\frac{(K-1)T}{K} \leq t \leq T$.

Instead of a timer for each prefix, only a small number (K) of timers are maintained here, which are shared by all the prefixes. The larger K is, the more closely the scheme simulates the individual timer scheme. Since T equals 1.5 times of the MRAI timer value (as discussed above), we set K as 9. Thus $\frac{T}{K}$ is one sixth of the MRAI timer value. From simulation tests, the two timer schemes work almost identically.

B. DUP+

Combining Diff with DUP, we can send potentially invalid routes using a longer MRAI timer while sending valid route using a shorter MRAI timer. This new algorithm is called DUP+, which is an extension of DUP. Under DUP+, the algorithm determines whether a prefix is in stable or transient state: if it is in stable state, the DUP algorithm will be executed; if it is in transient state, it will compare LCS values instead to determine the priority of an update. The details of the sender-side DUP+ algorithm are given in Tables V. Note that DUP+ is a superset of DUP. In line 3 of Table V, the corresponding code for DUP is called.

1) overhead: DUP+ introduces some extra overhead in terms of state maintenance and processing load. In Diff algorithm, once a prefix is in transient state, it needs to maintain its susceptible AS path segment. This consumes $2L$ bytes of memory for each path segment, where L is the length of the AS path segment and is generally a single digit. On the other hand, these states are required only for the prefixes that are affected by failures, not for every prefix. By studying Route Views data, we found that typically an AS makes routing changes for less than 1000 prefixes during a 15-minute period. Thus the total required memory space is less than $2000L$ bytes. By using the global timer scheme, Diff also needs to maintain an array of global timers and their associated prefix queues; but the number of timers and their queues is a small fixed number. In addition, DUP+ maintains an extra MRAI timer for each neighbor.

TABLE VI
PARAMETERS AND THEIR DEFAULT VALUES

Parameters	Values
Link delay	0.01-0.1 (sec)
$MRAI_{long}$ (for low priority)	30 (sec)
$MRAI_{short}$ (for high priority)	15 (sec)
MIN_PROC_TIME	0.01 (sec)
MAX_PROC_TIME	0.5, 0.1 (sec)
Number of advertisers	3
Length of timer t_p (used by Diff algorithm)	1.5 * $MRAI_{long}$
Number of global timers	9

DUP+ also incurs extra processing overhead. The LCS computation in Diff only occurs during failures, and its complexity is $O(n)$, where n is the length of the longest common subsequence of the two AS paths. All the other functions are just simple modification of existing BGP, which contain only a few dozen lines of code. Thus the extra overhead is small.

VII. EVALUATION

To demonstrate the benefits of the DUP+ algorithm,[4] we evaluate it using SSFNet's BGP simulator [29], a Java-based simulator widely used for studying BGP performance (e.g., [20], [22], [24]).

A. Simulation Design

We studied two scenarios: new route propagation and link failure. The performance metrics used are valid network convergence time, average valid convergence time, and the number of updates exchanged.

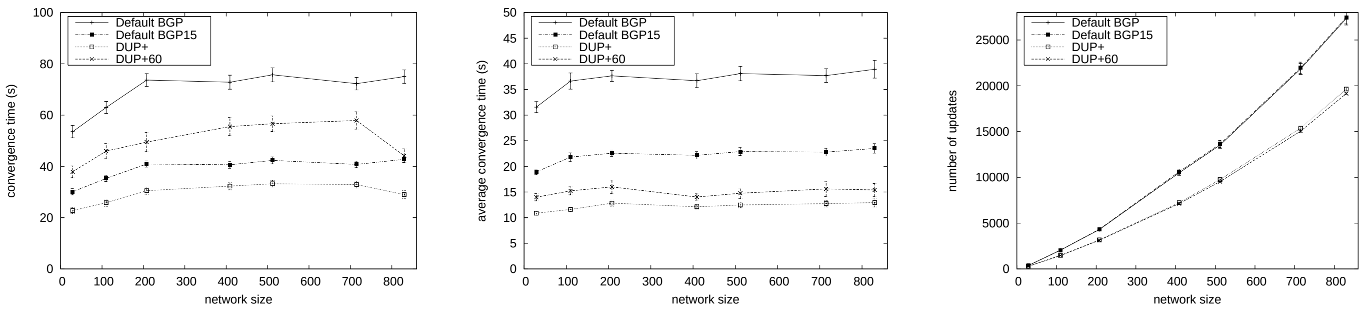
First, we define valid convergence time, based on which the other two definitions follow:

Definition 1 (Valid convergence time): The valid convergence time of a router is the length of the time interval (t_e, t_c], where t_e is the time when the origin router sends out the first update messages, and t_c is the time instant after which the router always has valid routes (through which the destination is reachable).
Note that a router may switch from one valid route to another after its valid convergence time being reached.

Definition 2 (Valid network convergence time): The valid network convergence time is the length of the time interval (t_e, t_{nc}], where t_e is the time the origin router sends out the first update messages, and t_{nc} is the time instant after which all the routers in the network always have valid routes.
The valid network convergence time is in fact the worst valid convergence time among all the nodes.

Definition 3 (Average valid convergence time): The average valid convergence time is the average over the valid convergence times of all the nodes in the network.

[4]Since DUP+ is a superset of DUP, we used only DUP+ in the evaluation.



(a) Valid network convergence time (b) Average valid convergence time (c) Number of updates

Fig. 6. Performance comparison between default BGP and DUP+: new route propagation

Compared to the commonly-used definition for convergence time [20], the valid (network) convergence time more accurately captures the reachability of a node (the network) to the destination and the impact of routing changes on application traffic. Similar definitions have also been proposed, such as *next-hop convergence time* [25] and *data-plane convergence time* [30]. Compared to them, the valid convergence time is easier to measure.

Table VI describes the simulation parameters and their default values. The link delay was randomly set between 0.01s and 0.1s. In the SSFNet simulator, the CPU processing delay for each packet is simulated to be a random value between two thresholds—MIN_PROC_TIME and MAX_PROC_TIME. They were set to 0.01s and 0.5s, respectively. (We also used 0.1s for MAX_PROC_TIME and found that the relative performance of DUP+ is very similar). For simplicity, we always set $MRAI_{short}$ to be half of $MRAI_{long}$.

To test some realistic network topologies, we used the multi-AS topology generating package from SSFNet [31], which contains seven different topologies based on the Internet BGP routing table. The number of nodes contained in the topologies ranges from 29 to 830. For each topology, we chose N nodes as advertisers, which announce their own IP prefixes to the network. Among all the nodes in each topology, we only chose those with a small number of neighboring nodes (less than or equal to four) in the simulation, meaning that they are more likely to be at the edge of the network. The default value of N is 3. For each simulation scenario, we randomly picked six instances of advertisers; for each of them, we conducted six independent runs with different random seeds. Therefore, there are 36 runs for each scenario. The results are summarized as the average value of the 36 runs, and are presented using 95% confidence interval.

B. Results

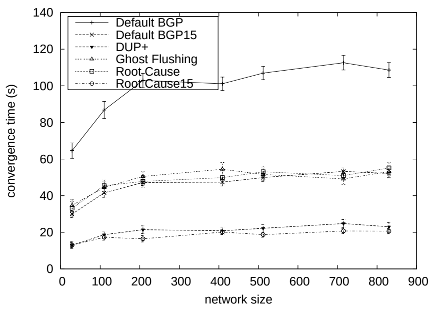
1) New route propagation: We first tested the case that new routes are propagated, and compared DUP+ with the default BGP. To study the interactions among routing changes, the three advertisers started at different times with a delay of a few seconds between them. We focused on the performance of the last advertiser; the other two were considered as the generators of cross traffic.

As shown in Fig. 6, the DUP+ algorithm yields very short valid network convergence time and average valid convergence time—only 40% or less of those for default BGP. At the same time, the number of updates is also smaller, saving about 30% of updates. This clearly shows the benefits of DUP+. For comparison, we also ran default BGP with MRAI timer of 15 seconds (called "default BGP15"), and DUP+ with MRAI timer of 30 (60) seconds for high (low) priority updates (called "DUP+60"). As the results show, DUP+60 still outperforms default BGP, and its average convergence time is even shorter than "default BGP15". The clearly shows the benefits of differentiated processing.

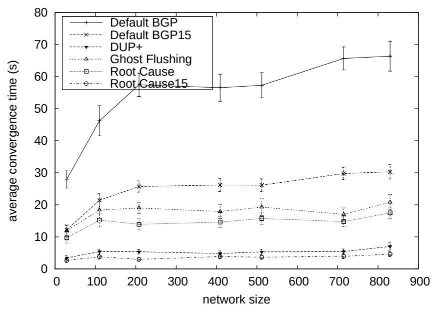
2) Link failure: We also evaluated the DUP+ algorithm under the link failure scenario, and compared it with the default BGP, Ghost Flushing, and Root-Cause based scheme. In addition to the default MRAI value of 30 seconds, we also used the value of 15 seconds for the default BGP (called "default BGP15") and Root Cause (called "Root Cause15") to see how well they perform with a shorter MRAI. For each topology, we randomly picked a node with a small number of peers as a test node. The test node first advertised some network prefixes to all its neighbors. For a given prefix, it advertised different AS path lengths to different neighbors by prepending. After the network initially converged, we broke the link between the test node and one of its neighbors, and observed how the algorithms perform. As in the new route propagation case, two additional nodes started advertising their prefixes shortly before the failure, injecting cross traffic.

As shown in Fig. 7, DUP+ has shorter valid network/average convergence time than not only both Ghost Flushing and default BGP (as much as 80%), but also Root Cause, which is a little surprising. The reason is that although Root Cause selects only valid route after failure by removing invalid route very fast, it propagates valid route the same way the default BGP does; in contrast, DUP+ propagates valid route with a shorter timer, which overcomes that fact that it is slower in selecting valid route. By using a shorter MRAI value of 15 seconds, we can see that "Root Cause15" indeed performs the best, but DUP+ is very close to it.

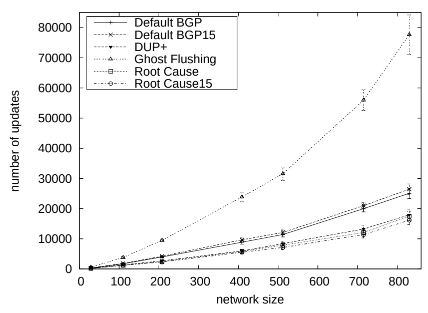
Under DUP+ the number of updates is very small as well. Again the number is very close to that for Root Cause. Although Ghost Flushing has shorter convergence time than the default BGP, the number of updates under it is surprisingly large. By checking the simulation data, we found that most of the updates are the extra withdrawals triggered by mechanisms of the Ghost Flushing algorithm.



(a) Valid network convergence time



(b) Average valid convergence time



(c) Number of updates

Fig. 7. Performance comparison between default BGP, DUP+, Ghost Flushing and Root Cause: failure case

In addition to the above three performance metrics, we also examined the number of routing changes under each algorithm and found the DUP+ algorithm to incur 50% fewer routing changes than the default BGP and Ghost Flushing, leading to more stable routes. In contrast, it is not as good as Root Cause since it is not as efficient in selecting valid routes; but the difference is rather small.

In summary, not only does DUP+ outperform the default BGP and Ghost Flushing, its performance exceeds or comes close to that of the Root-Cause based scheme as well. This shows that the performance of the current BGP can be significantly improved using our scheme without changing BGP message format required by Root-Cause based approaches.

VIII. Concluding Remarks

In this paper, we presented a simple and novel way of differentiating BGP updates based on their impact on inferred routing decisions of the receiver. It is shown to make significant improvements in both reducing the routers' overhead of processing an excessive number of BGP updates, as well as reducing routing convergence time. The proposed scheme is simple to implement, requires no modification of BGP protocol semantics, and can be deployed incrementally.

So far, we have mainly focused on EBGP (External BGP) sessions (for BGP neighbors belonging to different ASes). Inside a large AS, there are also many IBGP (Internal BGP) sessions. However, the default MRAI timer for IBGP is only 5 seconds, much shorter than that for EBGP. In addition, all the IBGP routers in the same AS are either directly peering with each other, or peering through route-reflectors. Therefore, routing delay inside an AS is generally shorter than that between ASes. Thus, our scheme does not modify the processing of updates between IBGP nodes.

References

[1] R. Teixeira, A. Shaikh, T. Griffin, *et al.*, "Dynamics of hot-potato routing in IP networks," in *Proc. of ACM SIGMETRICS'04*, June 2004.
[2] C. Villamizar, R. Chandra, and R. Govindan, "BGP Route Flap Damping," RFC 2439, Nov. 1998.
[3] Y. Rekhter and T. Li, "A border gateway protocol 4 (BGP-4)," RFC 1771, Mar. 1995.
[4] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," in *Proc. of ACM SIGCOMM'97*, Sept. 1997, pp. 115–126.
[5] C. Labovitz, A. Ahuja, A. Bose, *et al.*, "Delayed Internet routing convergence," in *Proc. of ACM SIGCOMM'00*, Aug. 2000.
[6] C. Labovitz, A. Ahuja, R. Wattenhofer, *et al.*, "The impact of Internet policy and topology on delayed routing convergence," in *Proc. of IEEE INFOCOM'01*, Apr. 2001, pp. 537–546.
[7] B. J. Premore, "An analysis of convergence properties of the Border Gateway Protocol using discrete event simulation," Ph.D. dissertation, Dartmouth College, Technical Report TR2003-452, May 2003.
[8] G. Huston, "Analyzing the Internet BGP routing table," *The Internet Protocol Journal*, vol. 4, no. 1, pp. 2–15, Mar. 2001.
[9] T. Bu, L. Gao, and D. Towsley, "On characterizing BGP routing table growth," in *Proc. of IEEE Global Internet Symposium'02*, Nov. 2002.
[10] A. Akella, B. Maggs, S. Seshan, *et al.*, "A measurement-based analysis of multihoming," in *Proc. of ACM SIGCOMM'03*, Aug. 2003.
[11] Z. M. Mao, R. Bush, T. G. Griffin, *et al.*, "BGP beacons," in *Proc. of Internet Measurement Conference (IMC)'03*, Oct. 2003.
[12] M. Caesar, D. Caldwell, N. Feamster, *et al.*, "Design and implementation of a routing control platform," in *Proc. of NSDI'05*, May 2005.
[13] M. Handley, E. Kohler, A. Ghosh, *et al.*, "Designing extensible IP router software," in *Proc. of NSDI'05*, May 2005.
[14] "GNU Zebra." http://www.zebra.org/
[15] L. Wang, X. Zhao, D. Pei, *et al.*, "Observation and analysis of BGP behavior under stress," in *Proc. of IMW'02*, Nov. 2002, pp. 217–229.
[16] M. Lad, X. Zhao, B. Zhang, *et al.*, "Analysis of BGP update surge during slammer worm attack," in *IWDC'03*, Dec. 2003.
[17] S. Agarwal, C.-N. Chuah, S. Bhattacharyya, *et al.*, "The impact of BGP dynamics on router CPU utilization," in *Proc. of PAM'04*, Apr. 2004.
[18] A. Feldmann, H. Kong, O. Maennel, *et al.*, "Measuring BGP pass-through times," in *Proc. of PAM'04*, Apr. 2004.
[19] S. R. Sangli, Y. Rekhter, R. Fernando, *et al.*, "Graceful restart mechanism for BGP," draft-ietf-idr-restart-10.txt, June 2004.
[20] T. G. Griffin and B. J. Premore, "An experimental analysis of BGP convergence time," in *Proc. of IEEE ICNP'01*, Nov. 2001, pp. 53–61.
[21] "Juniper's MRAI." https://www.juniper.net/techpubs/software/junos /junos57/swconfig57-routing/html/bgp-summary32.html
[22] D. Pei, X. Zhao, L. Wang, *et al.*, "Improving BGP convergence through consistency assertions," in *Proc. of IEEE INFOCOM'02*, June 2002.
[23] D. Pei, M. Azuma, N. Nguyen, *et al.*, "BGP-RCN: Improving BGP convergence through root cause notification," Computer Science Department, UCLA, Tech. Rep. TR-030047, Oct. 2003.
[24] J. Chandrashekar, Z. Duan, Z.-L. Zhang, *et al.*, "Limiting path exploration in BGP," in *Proc. of IEEE INFOCOM'05*, Mar. 2005.
[25] A. Bremler-Barr, Y. Afek, and S. Schwarz, "Improved BGP convergence via ghost flushing," in *Proc. of IEEE INFOCOM'03*, Apr. 2003.
[26] R. White, "High availability in routing," *The Internet Protocol Journal*, vol. 7, no. 1, pp. 2–14, Mar. 2004.
[27] "University of Oregon Route Views Project." http://www.routeviews.org/
[28] W. Sun, Z. M. Mao, and K. G. Shin, "Differentiated BGP update processing for improved routing convergence," Univ. of Michigan, May 2006. http://www.eecs.umich.edu/˜wsunz/publications/bgp-dup-tr06.pdf
[29] "Scalable Simulation Framework (SSF)." http://www.ssfnet.org/
[30] F. Hao and P. Koppol, "An Internet scale simulation setup for BGP," *ACM Computer Communication Review*, vol. 33, no. 3, July 2003.
[31] B. J. Premore, "Multi-AS topologies from BGP routing tables." http://www.ssfnet.org/Exchange/gallery/asgraph/index.html