Explicit Edge Withdrawal in BGP

Yehuda Afek Anat Bremler-Barr Shemer Schwartz
Tel-Aviv University

{afek,natali}@math.tau.ac.il, shemers@xiv.co.il

Abstract— A new approach to solve the BGP convergence problem is presented. Currently, BGP updates, both announcements and withdrawals, convey the status of the routing table of the sender of the message. Through the ASpath that is attached to each announcement the connection between ASes may be deduced. However, these messages do not include conclusive information about the Internet structure. In the method presented here BGP updates succinctly include information about the topological changes that triggered the update. This makes the removal of unreachable destinations and the removal of invalid paths from the routing table much faster i.e., shorter convergence time following topological changes. Just as announcing a new path to a destination, indicates the Internet structure that enables this path, we suggest that a withdrawal message (we call them explicit edge withdrawals) will include the exact topological change that made this path invalid. The latter enables the router that receives the withdrawal message to remove all paths to destinations that rely on the invalid path. In a way our algorithm benefits the advantages of two different routing algorithms. It chooses new paths like BGP (distance-vector protocol), thus, having low computational complexity. And it checks the integrity of new and old paths, similar to link-state protocols, thus, gaining fast convergence on the occurrence of a topological change in the network.

I. INTRODUCTION

The Internet is composed of thousands of Autonomous Systems (ASes) that are using the BGP protocol to determine their best route to each destination. BGP is currently the only inter-AS routing algorithm in the Internet and was believed to support fast recovery after a link or a router failure. In [2] and [3] it is shown that BGP does not efficiently support inter-domain failovers. Moreover, it is shown in [2] and [3] that "the latency in Internet failover stems from delayed convergence, or temporary routing table oscillations formed during the operation of path selection on Internet backbone routers after a fault".

Shortening the convergence time of the BGP routing protocol is a critical step in providing QoS and highly available services, such as VoIP, on the Internet. The severity of this problem increases with the increase in the Internet size. As shown in [2] and [3], the average latency of a failover failure is 3 minutes and can be as high as 15 minutes in some cases. Furthermore, it is shown that the convergence complexity of BGP is 30(n-3) seconds, where n is the length of the longest AS path in the network and 30 sec's is the value of MinRouteAdver timer. These latencies are due to the way BGP operates following the failure or disconnection of a destination from the network. Currently BGP explores a large number of alternative routes when a failure occurs before determining that a destination is unreachable. These alternative routes are already invalid when BGP tries them, but it takes BGP a long time to realize it, and to remove the false information these routes rely on. The MinRouteAdver timer is a mechanism specified in the BGP specification [7] that limits the rate of updates a router can send. If a route to a destination has changed multiple times during MinRouteAdver period, only the last change before the expiration of the timer is announced. In [2] it is shown that without the MinRouteAdver mechanism, in the theoretically worst case, a clique network of n-ASes may explore n! possible routes before converging.

In this paper we introduce a method significantly improves the convergence time of BGP. First, our algorithm removes rapidly all invalid information from the network on the occurrence of a link or router failure. This is achieved by including an indication on the link that has caused withdrawal, in the withdrawal message. The information thus conveyed to a router enables it to purge from its routing-table, not only the route associated with the withdrawal message, but also any alternative route that is no longer valid. After the removal of invalid information from the network, the BGP can quickly converge since routers are relying only on valid information. After the fast removal of invalid information from the network and until the algorithm converges, all the routes in the network are valid, although not optimal. This prevents IP packets from traversing the network without reaching their destination. Thus the suggested algorithm achieves both faster convergence time and during most of the convergence process routes in the network are valid.

Our proposed improvement for the BGP protocol can be regarded as an intermediate protocol between a distance-vector protocol (BGP) and a link-state protocol (such as OSPF). We add to the BGP protocol qualities that are related to link-state protocols while maintaining the computation complexity of a distance-vector protocol. The algorithm of choosing a new path is similar to that of BGP, but the examining of the integrity of old and new paths after a topological change relies on knowledge of the topological structure of the network, similar to link-state algorithms. In link-state algorithms, each router holds the entire structure of the network, as retrieved from link state messages received from all the nodes in the network. Each router computes the preferred path to each destination based on this structure. The communication complexity of exchanging the link state information between every pair of routers in the network is very high in large networks. We prefer the distributed method used in BGP, where the exchanges is between BGP neighbors. However, we want to use the knowledge of the Internet structure to remove invalid routes from the routing table. Instead of knowing the

entire structure of the network as in link-state protocol, we extend the BGP protocol so each router knows the relevant portion of the network structure i.e., where the routes (AS paths) it has are passing. Thus, instead of flooding the network with update messages as in link-state protocols, we use the BGP UPDATE messages to piggyback on them the relevant information. In a withdrawal message, instead of merely listing the names of prefixes that are no longer valid, we also add the topological event that has invoked this withdrawal. This method efficiently removes invalid information from the routing-tables, while maintaining reasonable computational and messages complexity. The result is an efficient convergence of the network upon any topological event.

A. Previous Work

Our work follows a lot of the footsteps of [8] where a method to improve the convergence time is presented by using consistency assertions. This method checks the consistence of two paths to the same destination in order to determine their validity.

Paths *path(N_1,D)* (from AS N_1 to AS *D)* and *path(N_2,D)* are consistent in [8], if one of the following conditions is satisfied:

1) Two empty paths are consistent.

2) Two non-empty paths with no common nodes between them are consistent.

3) Empty path(N_1,D) is consistent with non-empty path(N_2 ,D) if $N_1 \neq Q_i$, for all i, $1 \leq i \leq n$ (path(N_2 ,D)=(N_2, Q_1, Q_2, .. Q_m, D)).

4) Two non-empty paths, path(N_1,D) and path(N_2,D) that intersect at node $P_j = Q_i$ are consistent if $path_{N1}(P_j,D) = path_{N2}(Q_i,D)$.

We explain the algorithm of [8] through the example in Figure 0(a) (taken from [8]). Suppose the (N1-D) link failed. N1 sends R an announcement that its new route to D is (N1, X, Y, D). R can conclude that N2's current route (N2, N1, D) to D is not consistent with N1's declaration. Upon N2's declaration, N1 has a route (N1,D) to D, while N1 has just declared a different route to D. R "believes" N1 about N1's routes more than N2's information about N1. Relying on the announcement from N1 it can ignore the information received from N2 and avoid sending announcements with false information. In [8] this method is expanded so it would be compliant with BGP.

The method described in [8] is based on the assumption that information on a neighbor router is more reliable when it is received directly from the neighbor than when it is received indirectly from another neighbor router. In other words it says that a router should not use a route that goes through one of its neighbors if this neighbor is not the first AS in the *ASpath*, assuming it chooses routes according to the shortest-path criteria. But this method might not handle inconsistency between routes that are not caused from differences at neighbor routers but at other routers along the *ASpath*.

Let us examine the example in Figure 0(b). R1 has two paths to DEST, through AS1 (preferred route) and through AS3 (alternative route). Suppose the link between AS0 and DEST

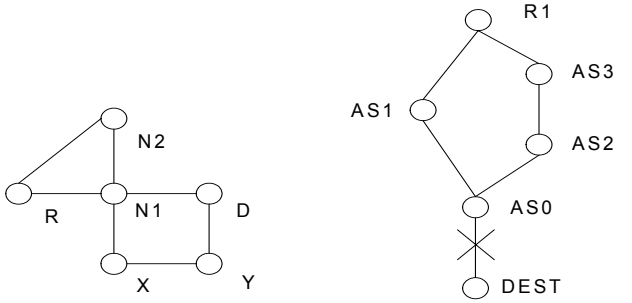


Figure 0

has failed. AS1 sends R1 a withdrawal message from DEST upon receiving a withdrawal message from AS0. The consistency checking in [8] is not sufficient for R1 to ignore the alternative path through AS3. Suppose the withdrawal message from AS3 is delayed, then, R1 will announce a new route to DEST through AS3. This exploration of an invalid backup path is the reason for BGP's delayed convergence. The method presented here prevents the exploration of invalid backup routes and thus improves the convergence time of BGP in the occurrence of a fail-down or fail-over.

Furthermore, the method in [8] relies on the fact that an AS cannot advertise different routes to the same destination to two different ASes. If an AS decides to traffic engineering by advertising different routes to the same destination, it requires an internal logic partitioning of the AS. The result is a large number of messages being sent on any modification of the internal partitioning of the AS.

Another approach to improve the BGP convergence complexity is presented in [11]. The basic idea in [11] is that the router that detects that one of its directly connected destinations has failed (i.e., it is the first router on the ASpath) adds its name to the withdrawal message. This way each router that receives the withdrawal message knows that any alternative path that uses the mentioned AS name is invalid as well. Our suggestion is very similar, the algorithm presented in [11] with the deference that our algorithm deals with any topological event that causes a convergence problem while the algorithm in [11] is intended to accelerate only the convergence due to a removal of a destination.

B. BGP background

The Internet consists of about ten thousands ASes that use the BGP as their routing algorithm. Each AS consists of many routers that exchange information about routes using interior gateway protocol (IGP). BGP neighbors from the same AS are called iBGP peers, while BGP neighbors from different ASes are called eBGP peers.

BGP is a distance and path vector routing protocol, meaning that with each destination (prefix) in the routing table an *ASpath* is associated, and the corresponding *ASpath* is sent with each update message that is sent on this destination to neighboring peers. The *ASpath* is the sequence of ASes along the preferred path from the router to the destination. The reason a route description is added to the BGP announcement is to prevent loops. A router that identifies its own AS number in an announcement it received, rejects this announcement and do

not use it to update its routing table. For each destination the router records the last announcement (with the *ASpath*) it has received from each of its peers (neighboring eBGP routers). Then, for each destination the router chooses one of its peers as the next-hop on the preferred path to that destination. Usually the router picks the peer that announced the shortest *ASpath*. However BGP is much more sophisticated and enables more complex path selection according to policy attributes. A router running BGP may also take a policy decision, such as, not to send or not to receive an announcement from different peers. For example, an AS may avoid announcing its *ASpath* to a destination, since it does not want to be a transit network for that destination.

BGP is an event driven (incremental) protocol where a router sends an update to its peers only when its preferred *ASpath* to a destination has changed, due to a topology change or a policy change. There are two types of message exchange between peering BGP routers: *announcements* and *withdrawals.* A router sends an *announcement* when its preferred *ASpath* to a destination has been changed or when it has a route to a new destination. *Withdrawal* messages are sent when a router has learned that a subnetwork (i.e. destination) is no more reachable through any of its interfaces. To avoid avalanches of messages and to limit the rate at which routers have to process updates, it is required that after sending an announcement for a destination a router waits a minimum amount of time before sending again an announcement for the same destination. It recommended setting this delay, called *minRouteAdver* to 30 seconds [7]. However the delivery of a withdrawal message is never delayed because BGP tries to avoid "black holes", in which messages are sent to destination, which is no longer reachable.

A router can announce its withdrawal from a destination explicitly and implicitly. Explicit withdrawals are withdrawals done by a *withdrawal* message. Implicit withdrawals happen when an existing route is replaced by an *announcement* by a less preferred route. When an implicit withdrawal occurs, a withdrawal message is not sent.

In this paper (same as in [2]) we consider four types of events which occur while BGP is running in the Internet:

- E_{UP} – A previously unavailable destination is announced as available at a router.
- E_{DOWN} – A previously available destination is announced as unavailable at a router.
- $E_{SHORTER}$ – A preferred *ASpath* to a destination implicitly replaces a less preferred *ASpath* (e.g. the path is becoming shorter).
- E_{LONGER} – A less preferred *ASpath* to a destination implicitly replaces a better (more preferred) *ASpath.* This happens for example, if the preferred route has failed.

It is shown in [2] that the convergence time of E_{UP} and $E_{SHORTER}$ is much shorter than E_{DOWN} and E_{LONGER}. The reason is that in E_{UP} and $E_{SHORTER}$ the length of the alternative path is increasing strictly while in E_{DOWN} and E_{LONGER} it is increasing monotonically. Upon receiving and selecting an alternative path in E_{UP} or in $E_{SHORTER}$, the router will never choose a route with a longer path. In E_{DOWN}, each router will *failover* to an alternative longer paths until it will withdrawal utterly from the destination. According to [2], if we assume bounded delays, the convergence complexity of E_{UP} and $E_{SHORTER}$ is O(1), while the convergence complexity of E_{DOWN} and E_{LONGER} is O(n).

The reason for the delayed convergence of E_{DOWN} and E_{LONGER}, which will be explained in more details in the next section, is due to BGP's exploration of alternative paths when the preferred path has been withdrew. In this exploration, routers rely on invalid information in their routing tables and in received update messages from their peers. The purpose of this work is to give E_{DOWN} and E_{LONGER} characteristics of E_{UP} and $E_{SHORTER}$ by eliminating the false information in the network.

We define route *failover* as the delay from a path failure to the establishment of a new path (E_{LONGER}).

We define route *faildown* as the delay from a destination becomes unreachable to the time it has been removed from the routing table (E_{DOWN}).

II. CONVERGENCE PROBLEM – GHOST INFORMATION

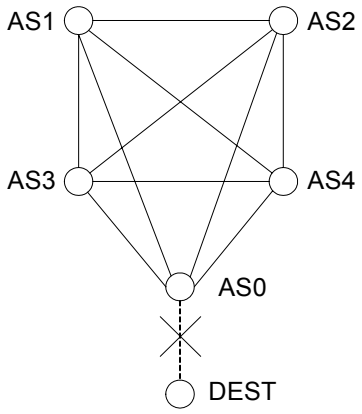
The problem with which we are dealing was presented and explained in [2] and [3]. It states that the delayed convergence caused by *failovers* last three minutes in average, and in some cases up to fifteen minutes. Here we repeat part of the explanation and try to shade light on the phenomena.

Let follow the example given that was given in [2] and [3], but we describe it in a slightly different way, thus exposing what we believe is the essence of the problem. We demonstrate the problem using topology shown in **Figure 0**. Let AS0 be connected in a clique to all other four *ASes* and all the *ASes* in the clique route to *dest* through AS0. Consider the case where AS0 withdraw its route to *dest* since the network *dest* becomes unreachable, and hence *dest* should be withdrew from all the routers' routing tables.

In [3] a detailed scenario is given, showing that the convergence time is (n-2)·*minRouteAdver* seconds with message complexity *nE,* where *n* is the number of nodes (*ASes)* and *E* is the number of links. For ease of explanation only, we give the scenario in a synchronous network, where at each round the router receives the messages sent in the previous round, calculates its new state and sends new messages to its peers if required. The duration of each round is *h* seconds, which we assume for simplicity, as 1 second. We assume that if a router has to update its *ASpath* to one of a few equal length *ASpath*es then it would prefer the *ASpath* that begins with the smallest *AS* number.

For each round (denoted by T=), and for every node (*AS*) we wrote its *ASpath* (with *ASpath*=), the node current announcement (with CA=) and the last announcement before this round (with LA=), which is its *ASpath* according to its peers. The letter W stands for a withdrawal message.

When router *dest* becomes unreachable, all the information about how to reach *dest* should be erased from the network. However, at the time *dest* becomes unreachable the false information is still in the network. Moreover, nodes start to use



LA – last announcement.
CA – current announcement.
W – withdrawal.

T=0:
AS0: CA=W - withdrawn its route from *DEST*.
AS1: ASpath=0, LA=10.
AS2: ASpath=0, LA=20.
AS3: ASpath=0, LA=30.
AS4: ASpath=0, LA=40.

T=1:
AS1: ASpath=20, LA=10, CA=120.
AS2: ASpath=10, LA=20, CA=210.
AS3: ASpath=10, LA=30, CA=310.
AS4: ASpath=10, LA=40, CA=410.

T=2:
AS1: ASpath={}, LA=120, CA=W.
AS2: ASpath=310, LA=210.
AS3: ASpath=120, LA=310.
AS4: ASpath=120, LA=410.

T=3:
AS1: ASpath={}, LA=W.
AS2: ASpath=310, LA=210.
AS3: ASpath=210, LA=310.
AS4: ASpath=210, LA=410.

T=31:
AS1: ASpath={}, LA=W.
AS2: ASpath=310, LA=210, CA=2310.
AS3: ASpath=210, LA=310, CA=3210.
AS4: ASpath=210, LA=410, CA=4210.

T=32:
AS1: ASpath={}, LA=W.
AS2: ASpath={}, LA=2310, CA=W.
AS3: ASpath=4210, LA=3210.
AS4: ASpath=2310, LA=4210.

T=33:
AS1: ASpath={}, LA=W.
AS2: ASpath={}, LA=W.
AS3: ASpath=4210, LA=3210.
AS4: ASpath=3210, LA=4210.

T=61:
AS1: ASpath={}, LA=W.
AS2: ASpath={}, LA=W.
AS3: ASpath=4210, LA=3210, CA=34210.
AS4: ASpath=3210, LA=4210, CA=43210.

T=62:
AS1: ASpath={}, LA=W.
AS2: ASpath={}, LA=W.
AS3: ASpath={}, LA=34210, CA=W.
AS4: ASpath={}, LA=43210, CA=W.

Figure 0- Example no. 1

and rely on the false information and thus the ghost information start to travel around the network building longer and longer *ASpath* until they include a cycle.

A. Formal Definition Of Ghost Information

In this section we attempt to give a formal definition of what is the ghost information.

The ghost information can be in one of the following forms:

1) The currently preferred ASpath at the router.

2) The information stored in a router about the preferred ASpath for each peer.

3) An information about a preferred ASpath in a BGP message (announcement or withdrawal) that traverse from a router to its peer.

In the formal description below, we describe the information associated with reaching *dest*.

We denote $ASpath^R$ as the preferred *ASpath* for reaching *dest* by router R.

We denote $ASpath^R{}_P$ as the last *ASpath* that P router announced according to R.

We denote $ASpath^R(m)$ as the ASpath in the last announcement concerning a route to *dest* by R.

Definition 1: $ASpath^R = \{ AS_R=AS_{R0}, AS_{R1}, AS_{R2}, \ldots, AS_{RN}\}$ is a ghost information at router R that describes a route to *dest,* if there exist 0<i<=N such that one of the following occurs:

- $ASpath^{Ri} \neq AS_{Ri}, \ldots, AS_{RN}$.
- $ASpath^{Ri+1}{}_{Ri} \neq AS_{Ri}, \ldots, AS_{RN}$.
- There exists a BGP message m (announcement or withdrawal) regarding dest that origins from Ri, which currently traverses to R(i+1), and $ASpath^{Ri}(m) \neq AS_{Ri}, \ldots, AS_{RN}$.

The scenario shown in **Figure 0** demonstrates how the convergence latency problem is due to *ghost information* that traverses the network. At T=1, AS4 after receiving a withdrawal message, chooses *ASpath*=10 according to the ghost information ($ASpath^1{}_4$ is 10). Then it sends an announcement saying that its *ASpath* to *dest* is 410. The ghost information, that origin from AS1 traverses in the network. Moreover, AS4 "knows" that AS0 does not have any path to *dest* but it chooses a route that relay on the edge (AS0, *dest*). AS4 has all the information it need to ignore all routes that relay on the invalid edge (AS0, *dest*). It can not do that in BGP since the name AS0 in BGP describes a group of routers, thus, the edge (AS0, *dest*) is not well-defined.

In the next round (T=2), AS4 would learn that if it chooses a route through AS1, the new *ASpath* should be 120, since AS1 has changed its path to 20. However, AS4 cannot update its peers about the change in its *ASpath* until at least 30 seconds have passes from the previous announcement. Here we see the negative effect of *minRouteAdver* that delays the elimination of false (ghost) information in the network. In this round AS1 also learns that all the other ASes are routing through it. It can not find a route without a loop. Hence it changes the *ASpath* to {}, and sends immediate withdrawal to all of its peers. As a result, at T=3 AS4 changes its *ASpath* to 210. Because of the *minRouteAdver* rule, AS 2 does not learn that all its peers' routes go through it until T=31, when all the *ASes* sends their next announcement. This delayed convergence yields a 62 seconds (rounds) convergence time

Observing the above scenario one may conclude that the large (30 seconds) *minRouteAdver* is the source of the problem and that by drastically reducing it the problem would be

solved. However, as shown in [9] and [2] that without this delay the message complexity of the convergence process jumps to *O(n!)* and the competition load on routers would also go much higher.

III. EXPLICIT EDGE WITHDRAWAL

In BGP when a node decides it withdraws a path to a certain destination, it sends an explicit withdrawal message in which it specifies the unreachable destinations or it sends an implicit withdrawal in which it sends an alternative path to the destination with a lower ranking value. Here we take a different approach.

To illustrate our method, we use a simplified model of the Internet as a directed graph, where each node represents an AS or a destination host/network and an edge represents the connection between them. The directed edge represents that a node at the origin of the edge advertises routes to the node at the end of the edge. In this model, there is at most one edge between two ASes. For simplicity we initially ignore the impact of policies and the effect of having multiple routers in one AS and thus might have multiple edges between two ASes.

In this model a withdrawal from a destination can happen due to two topological change in the network:

1) Node removal.
2) Edge removal.

From the perspective of a node that one of its peers has been removed or the edge to one of its peers has been removed, these two events are the same. The topological change that the node can see is that the edge to its peer is invalid. Thus, the node that its edge becomes invalid sends all its peers an *explicit edge withdrawal* that specifies the edge that was removed.

Definition 2: An *explicit edge withdrawal* is a withdrawal message that states that the sender has stopped using this certain edge for all its paths to all destinations.

The node also removes all the paths in its routing table that uses the removed edge and updates the preferred route to each destination accordingly. Then it sends advertisement messages about its new paths to destinations that their routes have been altered. Destinations that become unreachable do not need an additional withdrawal message since it can be deduced from the *explicit edge withdrawal* message. While updates about new routes are delayed in BGP routers by *MinRouteAdver, explicit edge withdrawal*s are treated as *withdrawal* messages and thus are not delayed by *MinRouteAdver*. No additional delay is required between the sending of the *explicit edge withdrawal* and the sending of the updates of the new *paths*.

In other words, we say that the current explicit and implicit withdrawal do not give the receiving router enough information. They give the status of the routing table of the sender, but they do not give enough information about the topological change that invoked this withdrawal. If an edge has failed, there are two "witnesses" to this event, the two nodes that this edge connects. If those "witnesses" can describe the event to their peers the ghost information can be removed quickly from the network.

If each node uniquely describes one router (=AS), so each edge uniquely describes as the connection between two routers. A node, upon receiving an *explicit edge withdrawal*, removes from its routing-table all the paths (including alternative paths) that use the withdrawn edge. It then sends the *explicit edge withdrawal* to all its peers. After reselecting its new preferred routes, it sends advertisement messages that describe the new paths to destinations that their paths have been altered due to the edge removal. Destinations that become unreachable, again, do not need an additional withdrawal message since it can be deduce from the *explicit edge withdrawal*. Nodes that *did not* announce any path that uses the withdrawn edge do not propagate the *explicit edge withdrawal*, since none of its peers have a path that uses both the node and the removed edge.

It is important to notice that the *explicit edge withdrawal* messages only propagates on the tree of nodes that have used the withdrawn edges in one of their preferred routes. There is no point in sending the *explicit edge withdrawal* message to a router that is not using the withdrawn edge in one of its routes.

The conceptual difference from BGP is that an *explicit edge withdrawal* message specifies now a topological change and not only a change in the routing-table of its origin. Of course we can also deduce the implication of such a withdrawal message on the routing table of its origin but its purpose is to tell which edges are no longer valid. The detailed description of the topological change in the network enables the node to remove all the ghost information from its routing-table and thus to prevent the exploration of invalid alternative route.

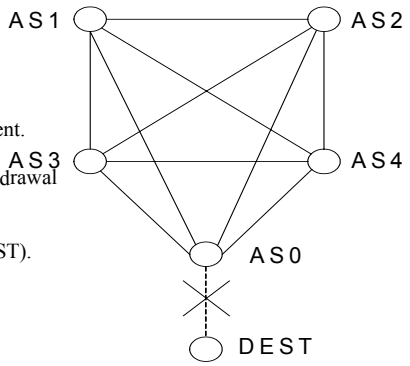
The modification is illustrated in the example of Figure 0 (b). Suppose the edge (AS0, DEST) has been removed. This could be as a result that AS0 has failed or the link between AS0 and DEST has failed. AS0 sends an *explicit edge withdrawal* message from edge (AS0, DEST). Before the removal of edge (AS0, DEST), R1 had two routes to DEST. The preferred route was through AS1 and the alternative route through AS3. When R1 receives from AS1 the *explicit edge withdrawal* from edge (AS0, DEST), it deletes both the preferred path (AS1,AS0,DEST) and the alternative path (AS3,AS2,AS0,DEST) from its routing table, since they both rely on edge (AS0, DEST). In regular BGP protocol, R1, upon receiving from AS1 a withdrawal from destination DSET, would have chosen (AS3,AS2,AS0,DEST) as its preferred route to DEST, until it would have received a withdrawal message from AS3.

B. Example no. 1 with Explicit Edge Withdrawal algorithm

We return to example no.1 that was first introduced in **Figure 0**. We use the same assumption, only now we use the *explicit edge withdrawal* algorithm.

The convergence of example no. 1 using the *explicit edge withdrawal* is straightforward, as shown in Figure 0. At T=0, AS0 recognizes that (AS0, DEST) edge has been removed. It removes its route from DEST and sends all its peers an *explicit edge withdrawal* message that specifies that (AS0, DEST) has failed. Upon receiving the *explicit edge withdrawal* message, AS1, AS2, AS3 and AS4 remove all their routes to DEST since their preferred routes and their alternative routes all rely on edge (AS0, DEST). Then they send *explicit edge withdrawal*

messages to all their peers. Their peers do not respond since they already removed all the routes with the edge (AS0, DEST).



LA – last announcement.
CA – current announcement.
W – withdrawal.
EEW – explicit edge withdrawal

T=0:
AS0: CA=EEW(AS0,DEST).
AS1: ASpath=0, LA=10.
AS2: ASpath=0, LA=20.
AS3: ASpath=0, LA=30.
AS4: ASpath=0, LA=40.

T=1:
AS1: ASpath={}, LA=10,CA=EEW(AS0,DEST).
AS2: ASpath={}, LA=20, CA= EEW(AS0,DEST).
AS3: ASpath={}, LA=30, CA= EEW(AS0,DEST).
AS4: ASpath={}, LA=40, CA= EEW(AS0,DEST).

Figure 0- Example no. 1 with *Explicit Edge Withdrawal*

IV. IMPLEMENTING THE ALGORITHM IN BGP

In this section we describe how to apply the *explicit edge withdrawal* algorithm on the currently used BGP. The basic idea is that instead of the withdrawal of a destination as is currently done in the BGP protocol, a router will announce the withdrawal of an *explicit edge* i.e., both withdrawing from destination and describing the topological change that have caused this withdrawal. If all routes in the routing tables rely on explicit edges, then an *explicit edge withdrawal* can be directly translated to a routing-table updates. Each route that uses the withdrawn edge is removed from the routing table, and the next hop is updated accordingly.

As oppose to the directed graph model that was introduced in the previous section, an AS in the Internet may have many BGP routers. As long as there is only one link between each two ASes there is no problem with our model, since the link between any two ASes can be uniquely depicted by the AS number of the two ASes. Thus, we can still explicitly describe the removal of an edge using the BGP convention, since the ASes on the route are included in the *ASpath* attribute.

In case there are several links between two ASes we call this group of links a *virtual edge.* The *virtual edge* is defined as the group of all links between BGP routers of two different ASes. The *virtual edge* is directed edge, that state the origin of the edge can propagates packets to the end of the edge. We can uniquely name each *virtual edge* with the names of the two ASes it connects.

Assumption no. 1: There is at most one edge connecting two ASes.

We will discuss in section F the validation of this assumption.

With the above assumption we have reduced the network model to that of up to a one edge between each two ASes. On this network model it is very strait-forward to implement our *explicit edge withdrawal.* Since the edges between ASes are unique, the information in the *ASpath* attribute in the BGP update is sufficient to describe any edge in the network. We do not have to assume anything about traffic-engineering or policies, since as long as a route is uniquely depicted by its edges, the BGP announcements and the *explicit edge withdrawals* describe any topological events that a router is interest with. The fact that an AS decides to advertise different routes to different ASes is irrelevant to our algorithm.

However, in order to describe an *explicit edge withdrawal* the current BGP UPDATE message format has to be modified. In the next sections we describe our proposal to adjust the current BGP UPDATE message format so it could be compatible with our algorithm.

A. Adjusting BGP Updates

In order to implement the *explicit edge withdrawal* and be backward compatible with the current BGP implementation we need to modify the BGP UPDATE message format. The *Withdrawn Routes* field (see [7]) does not allow us to describe an withdrawn edge and to be backward compatible at the same time. Furthermore, the *Path Attributes* field only exists in an UPDATE message when there is an advertised path. The strait-forward way was to add a new path attribute to a withdrawal message, but then we would not be backward compatible. Our way out of this maze is to add to the regular BGP withdrawn message an announcement of an illegal path (a path that includes a loop), and to use the *Path Attributes* field to indicate that this is an *explicit edge withdrawal* message. We call this attribute WITHDRAWN_EDGE, and its first purpose is to indicate that this is an *explicit edge withdrawal* and not regular announcement with a loop. Furthermore, the attribute WITHDRAWN_EDGE contains the information about the removed edge, as is explained below. The announcement illegality will be that the *AS* of the router that receives the message will be included in the *ASpath* in the AS_PATH attribute in the sent message, just before the AS of router that has sent it. An unmodified BGP router will discard this announcement since it will detect a loop in the *ASpath*. A modified BGP router, upon receiving an UPDATE with WITHDRAWN_EDGE path attribute, ignores the announcement in this message.

The withdrawal messages will include all the destinations that have used the removed edge. This way we achieve two goals. First, we are backward compatible with BGP. A router that does not implement the *explicit edge withdrawal* mechanism simply withdraws from the specified destinations. Second, each router that implements the *explicit edge withdrawal* has pointers to all the destinations that uses the removed edge (that went through the advertising router) and does not have search them in its routing-table. The router removes all the routes *to the withdrawn destinations* that use the removed edge, even if they were not published by the router that has sent the *explicit edge withdrawal* (ghost information).

After the *explicit edge withdrawal* has been received, it is possible that there are still remaining destinations in the routing-table that use the withdrawn edge, since these destinations were not mentioned in the *explicit edge withdrawal* so we did not removed them yet. The reason can be either because they can still use this route or because the sending router does not advertise a route to them or it uses a different route that does not use the removed edge. If indeed the removed edge is not valid for those destinations as well, we will receive an *explicit edge withdrawal* from the router that has published these routes.

We use the quality of BGP's withdrawal messages, which are not delayed by *minRouteAdver*, to our *explicit edge withdrawal* messages, in order to flood rapidly the network about the new topologic change i.e., the edge removal. This way we quickly eliminate the ghost information in the network and ensure that the *explicit edge withdrawal* propagates at least one step ahead than the announcement of alternative paths.

In order to describe the withdrawn edge we need to specify the names of the two ASes that an edge has fall in between them. We use the new WITHDRAWN_EDGE attribute to describe the removed edge. This attribute contains the names of the two ASes that the edge between them has failed.

In case that a destination has failed and not an edge between two ASes, we describe the removed edge solely with the name of the first AS, since the destination is not part of the *ASpath.*

Since a BGP UPDATE message can only contain one route to one or few destinations, BGP UPDATE can only contain one *explicit edge withdrawal* (and no announcement on new routes as well).

B. Withdrawal Caused By Policy change

A route to a destination may become invalid due to a policy change by one of two policy changes types:

1) AS1 on the ASpath decides to stop advertising the route to this destination to AS2 which is the next AS on the ASpath.

2) AS2 on the ASpath decides that it no longer accepts announcements from AS1 which is the AS that comes just before it in the ASpath.

In both cases an edge is no longer valid for a certain destination or a group of destinations. If case 1 occurs, AS1 sends AS2 an *explicit edge withdrawal* message from edge (AS1,AS2), in which the *Withdrawn Routes* attribute specifies the withdrawn destinations. If case 2 occurs, then AS2 sends all its peers an *explicit edge withdrawal* message, in which the *Withdrawn Routes* attribute specifies the destinations that it no longer uses the edge (AS1,AS2) to reach them.

C. Description of Basic Algorithm

In this section we give a full description of our basic algorithm. In the bellow description of our basic algorithm we assume that only one event can occur at any point of time i.e., the next topological change starts only after the network has converged from the first topological change. We will explain the implications of multiple events occurring concurrently in the next section.

Description of basic Algorithm for *explicit edge withdrawal*:

- The announcement of a new route to a destination goes according to BGP protocol.
- The processing of an announcement about a new route to a destination goes according to BGP protocol.
- On the detection that its preferred route to a destination is no longer valid through one of its edges due to a topological change, the router sends to all of its peers to whom it has previously announced this route, an *explicit edge withdrawal* message. The message states that there is no valid route to the destination using the specified edge. *MinRouteAdver* does not apply to the *explicit edge withdrawal* message. If the router has a valid alternative route, it announces it right after the *explicit edge withdrawal* message (or in the same message) under the restrictions of *MinRouteAdver* timer.
- On the receiving of an *explicit edge withdrawal* from its peer, the router deletes all the routes in its routing-table to the destinations specified in the *Withdrawn Routes* attribute that are using the withdrawn edge. It deletes routes even if their next hop is not the router that has sent the *explicit edge withdrawal* message. Then it propagates the *explicit edge withdrawal* to all its peers to whom it has announced a route that uses the withdrawn edge. Finally, it calculates its new preferred routes and advertises its peers if it has any new valid route to the withdrawn destinations.

D. Multiple Withdrawals and Oscillating Edge Problem

In this section we describe a problem that might arise in our algorithm due to multiple events occurring at the same time. Our discussion refers to multiple topological changes that are related one another, since if there are multiple events that are orthogonal, the basic algorithm presented in the previous section handles it properly.

We illustrate the problem that might arise when multiple topological events occur together using an example shown in Figure 0. We have a scenario in which a destination host (DEST) disconnects and reconnects several times in a time interval that is of the order of convergence time of our algorithm. R1 has two or more separate routes to that host that are all using the same oscillating edge (AS0,DEST). R1 has a problem to understand the order of the BGP UPDATEs it receives, since messages propagating on different routes suffer different and unknown latency. A message from AS1 that informs that the edge (AS0,DEST) has been restored might be received before AS3 announces that the edge was removed.

Furthermore, a contradiction can arise between announcements of two peers, in case while DEST is disconnected, another edge has failed (AS3,AS2) on the route to this destination. When the edge (AS0,DEST) is restored, AS1 will announce that it is valid, but AS3 last announcement to R1 is that edge (AS0,DEST) is invalid. To whom R1 should "belive"? Due to a load in AS3, the withdrawal message of

AS3 from *dest*, might be received in R1 after AS1 has announced that the edge was restored.

This problem is similar to that link-state algorithms (such as OSPF) have. OSPF solve this problem by applying a time-stamp to each message. A time-stamp is less practical in an inter-domain protocol since it requires synchronization between routers from different ASes. In section E we describe a modification to our basic algorithm that handles with the above consistency problem.

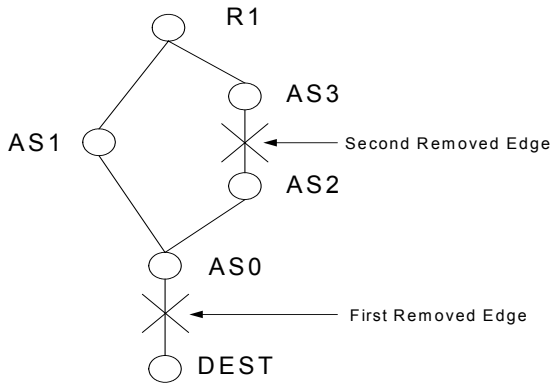


Figure 0 – Example no. 2 of multiple withdrawals

E. Enhanced Algorithm Description

To solve the problem we introduce in the previous section, an improvement to the *explicit edge withdrawal* algorithm is required. The improvement is in the way a router disqualify a route advertised from one of its peer according to an *explicit edge withdrawal* message received from another peer. Our improved algorithm goes as follows:

1) On the receiving of an explicit edge withdrawal message from one of the router's peers, the router deletes the route through this peer to the withdrawn destination. It suspends the routes from other peers if they are using the withdrawn edge. By suspension we mean that the routes are still in the routing-table but the router can not choose them as the preferred route. Further on, it propagates the explicit edge withdrawal message to all of its peers. It sets a timer *EdgeSuspentionInterval*. The *EdgeSuspentionInterval* duration is much longer than it takes a message to propagates in the Internet (~30 seconds).

2) Any receiving of a new announcement of a route to the withdrawn destination that uses the withdrawn edge will stop the *EdgeSuspentionInterval* timer and take out of suspension all the suspended routes. The deleted routes can not be restored of course.

3) On any receiving of an *explicit edge withdrawal* message from one of the other peers, the router will delete the suspended route that goes through this peer from the routing-table.

4) If the EdgeSuspentionInterval timer has elapsed and there is still a suspended routes in the routing-table then these routes go out of suspension and they can be used again as the preferred route.

The enhanced algorithm presented above can be regarded as two algorithms working together. The first is our basic algorithm that handles *explicit edge withdrawal* and announcement messages. The second is a monitoring algorithm that checks the validity of each *explicit edge withdrawal* message and its consistency with other received messages. The last influences the router's routing decisions only if there is inconsistency between messages received from different peers on the occurrence of multiple topological events that are not orthogonal.

We illustrate how our enhanced algorithm works by using a

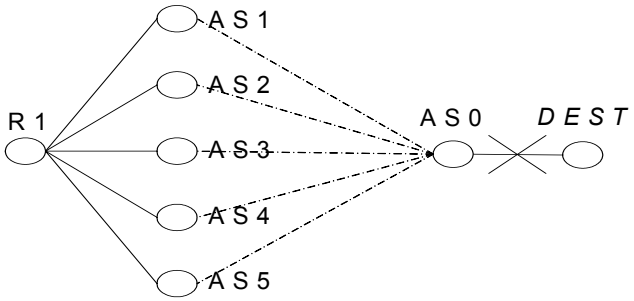


Figure 0

scenario presented in Figure 0 (the dashed-line represents a route that comprises zero to several ASes). In this scenario, at time T=0, AS1 ⇔ AS5 last announcement to R1 is that they have a route to DEST. At time T=0 the edge between AS0 and DEST was removed. We assume that R1 only receives advertisements from AS1⇔AS5 and does not send them back.

While edge (AS0,DEST) is down and all R1's peers have sent withdrawal messages, the connection between (AS0,AS1) goes down and then (AS0,DEST) goes up again. At the end of the convergence process there should be a valid route from R1 to DEST through all of R1's peers excluding AS1. In this scenario AS1, as oppose to R1's other peers, will never announce that it has a route to DEST (we assume it has no alternative pass to AS0 or DEST). In case AS1 withdrawal from (AS0,DEST) is received in R1 before the last announcement from one of the other peers is received, then R1 on the receiving of the announcement that comes after AS1 withdrawal will set all its routes to DEST out of suspension, and will choose the right route to DEST. On the other hand, if AS1's withdrawal is the last message to be received, unless we use *EdgeSuspentionInterval* timer R1 will decide that there is no feasible route to DEST. After AS1's withdrawal is received, since all other peers have sent their announcements R1 will wait until *EdgeSuspentionInterval* has elapsed. Then, since no other withdrawals have been received from the other peers, it will take AS2⇔AS5 announced routes out of suspension and R1 can choose a route to DEST.

Our improved algorithm achieves robustness when multiple topological events occurring at the same time, while preserving the fast convergence qualities of the basic algorithm. In section V.C we will prove that it converges and its convergence time.

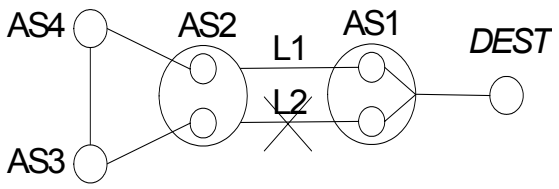


Figure 0

F. Discussion on the Virtual Edge Assumption

At the beginning of section IV, we have made an assumption regarding the *virtual edge* in our network model. We assumed that there is only one edge between two ASes. Here we discuss the validation of this assumption.

Inside an AS there is a mesh topology between all BGP routers through IBGP protocol. All the routers in an AS are synchronized about the routes of all the other routers in their AS. Thus, if one eBGP router has a link to a peer AS all other eBGP routers have a possible link through it to this peer even if they do not have a direct link to it or if their direct link to the peer AS has failed.

We might have a contradiction to our one *virtual edge* assumption in the following scenario that is shown in Figure 0.

AS1 and AS2 have two links connecting them: L1 and L2. In AS2, L1 is the link going out from NY and L2 is the link going out from SF. AS2 uses link L1 on the route from AS4 to *dest* and it uses L2 on the route from AS3 to *dest*. The scenario in which we have a problem with our assumption is when link L2 has failed and AS2 does not want to use L1 for a route from AS3 to *dest*. The motivation of AS2, is to do Hot potato, and route the traffic out of AS2 ASAP, and avoid the traversing of the traffic inside its backbone. The problem is that AS3 will announce that the edge (AS2,AS1) is not valid for *dest*, while it actually valid for other ASes. If AS4 has advertised AS3 a route to *dest* (or vises versa), or if there is another AS that receives routes that relay both on L1 and L2 it will cause a malfunction of our algorithm. If AS2 advertise AS3 a route that uses L1 (even a long *ASpath* by stuffing its AS number several times) we do not have a problem since the edge is still valid.

The scenario above is infeasible when all the policies are determined according to AS number. This scenario can implement in BGP only by using community mechanism. The updates from SF router will be colored with SF-community by the SF router, and the updates from NY router will be colored with NY-community by the NY router. SF router, would filter out any updates that come with NY-community, and the NJ router, would filter out any updates that come with SF-community.

V. Convergence Time Complexity of Explicit Edge Withdrawal

In this section we prove that the convergence time complexity of our proposed basic algorithm after *faildown* or *failover* is: D * *MinRouteAdver* + D * h, where D is the diameter of the network, and h is the maximum processing and propagation delay for all routers.

We assume the following:

1) Connections between routers can be modeled as FIFOs, meaning the first message that was sent will be the first message to be received. This quality is embedded in the BGP protocol since it runs over TCP.

2) Router processes their incoming BGP messages in an ordered way. The first received message is the first to be processed.

Claim 1: Upon an event of the removal of one edge (that causes a *failover* or a *faildown*), the first indication of the event that any router in the network will get, besides the router that has sensed the event, is an *explicit edge withdrawal.*

Claim 1 states that if a destination that is connected to the network through one edge has been removed or the connection to the destination has failed, or a connection between two routers has failed or a policy change has occurred, then the first message that any router in the network receives as a result of this event is an *explicit edge withdrawal*. Announcement about alternative routes will follow the *explicit edge withdrawal*.

Proof 1: We prove it by using an induction. For all peers of the router that detects the edge removal i.e. for d=1 (distance), we use the basic algorithm description in section C. The detecting router first sends the *explicit edge withdrawal*. The router's peers will receive the *explicit edge withdrawal* before any announcement according to assumption no. 1.

We assume that for routers, with diameter N from the detecting router, the claim is correct. We prove that the claim is correct for any router that its distance is N+1.

We prove by negation. Let assume that the router with distance N+1 receives an announcement before it receives an *explicit edge withdrawal.* We look recursively for the origin of this announcement. If the peer that has sent this announcement has a distance N from the first router then we get a contradiction according to the induction assumption and assumption no. 1 and 2. The path that this announcement had to go from the detecting router to the router with distance N+1 must be greater than N+1. We look for the first router on this path that has sent an announcement before an *explicit edge withdrawal.* Its distance from the detecting router must be greater than N. Now, if this is the first router that has sent an announcement before the *explicit edge withdrawal*, then its peer has sent it an *explicit edge withdrawal,* otherwise it is not the first. This router upon receiving an *explicit edge withdrawal* has sent an announcement and not an *explicit edge withdrawal*. This is a contradiction to the definition of our protocol or to assumption no. 1 or 2. In our algorithm a router upon receiving an *explicit edge withdrawal*, firstly, sends this *explicit edge withdrawal* to all its peers□

What can we conclude from claim 1: for all routers in the network, before receiving or announcing any alternative route, all ghost information regarding the topological change (edge failed) has been removed. This means that all announcements about a new route or withdrawals that origin from any topological change will not have any ghost information.

Thus, we can divide the convergence time of the network after an edge removal to two separate processes:

1) Removal of ghost information from network.

2) Stabilization of alternative routes.

On a *faildown* event only process 1 occurs, since there are no alternative paths. Process 2 applies only for *failover* events.

Process 1 time complexity is: D*h. D is the diameter of the network. Actually D should be the maximum distance of the detecting router to any router in the network, but we can block it by the diameter of the network. h is the maximum processing and propagation delay for all routers. Since *explicit edge withdrawal* messages are not delayed by *minRouteAdver*, D*h is the time it takes to flood the network with the *explicit edge withdrawal* message.

Process 2 time complexity is: D* *minRouteAdver*. According to *Claim 1*, process 2 begins at each router only after process 1 has finished. After the removal of all the ghost information, paths to destinations in each router can only get shorter upon receiving an UPDATE message from one of its peers. Since there is no ghost information in the network D* *minRouteAdver* is the maximum time it takes the shortest path to reach all nodes in the network. *minRouteAdver* is the maximum time message is delayed in each router.

After the fast removal of the ghost information, all the routes in all routing-tables in the network are valid routes (not optimum but valid). It prevents IP packets from traversing in the network without feasibility of reaching their destination.

B. Basic Algorithm Message Complexity

The removal of ghost information message complexity is: 2xE. E is the number of directed edges in the graph. On each edge in the network the *explicit edge withdrawal* is sent at most twice (this can be reduced to one if the router does send the *explicit edge withdrawal* back to whom it has received it from).

When *faildown* event occurs then the number of sent messages will be 2xE, since no alternative paths exists.

Upper bound of sent messages after the removal of ghost information i.e. the establishment of alternative paths: (Convergence time/ *MinRouteAdver*)*E = D * *minRouteAdver* / *minRouteAdver* * E = D*E.

This is a very weak upper bound since it assumes that in each round of *minRouteAdver* an announcement about a new path is sent on every edge of the network. This means that every *minRouteAdver* each node in the network finds a new preferred path to the destination.

C. Enhanced Algorithm Convergence Time

In this section we discuss the convergence time of the improved algorithm described in section IV.D. For this purpose we will prove the following claim:

Claim 2: In this claim we relate to the following scenario presented in Figure 0:

At time T_0, there is at least one route going from *DEST* to R1, going through router A. The route that R1 has chosen to *DEST* is called the *preferred route* (PR_{DEST}), and it is going through routers C, B and A. PR_{DEST} is the shortest route from *DEST* to R1. All edges in the network are valid at time T_0.

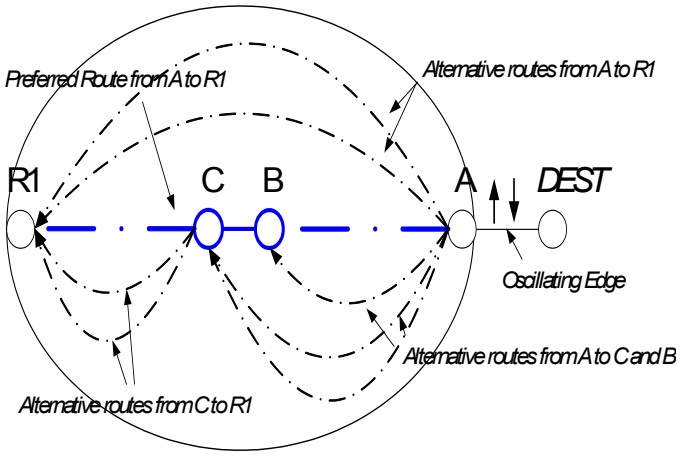


Figure 0

Between T_0 and T_1 the edge (*DEST*,A) is oscillating, meaning, the edge (*DEST*,A) is going from being valid to being invalid several times.

From time T_1, edge (*DEST*,A) is constantly valid for a route to *DEST*. Furthermore, all edges that are not on the *preferred route* from R1 to *DEST* can go from valid state to invalid state and vice-versa randomly. These edges can be on any of the alternative routes from R1 to *DEST* but not on the preferred route.

The maximum time it takes R1 to converge on PR_{DEST} as its preferred route is:

Delay(R1,A) + MinRouteAdver x Length(R1,A) + EdgeSuspentionInterval.

Delay(R1,A) is the maximum time it takes a message to propagates from A to R1. This delay is related to any *Explicit Edge Withdrawal* message or to any Announcement message, not including *MinRouteAdver* delay it can suffer.

Length(R1,A) is the number of hops from A to R1.

In order to prove the above claim we assume that Delay(~,A), the time for an *Explicit Edge Withdrawal* message that origin from A router to reach any other router in the network is smaller than *EdgeSuspentionInterval.*

Proof 2: On time T_1 all the edges on the preferred route PR_{DEST} from A to R1 are valid. What is the maximum time it takes for R1 to announce that PR_{DEST} is its preferred route to reach *DEST*.

For BGP the maximum convergences time is: Delay(R1,A) + *MinRouteAdver* x Length(R1,A). It consists of the time it takes an announcement of a new route from A to R1, while on each router on the way it might be delayed for *MinRouteAdver*. Since it is the preferred route to *DEST* for each router on the way there are no other delays. In most cases, the message will not be delayed at each router for the whole *MinRouteAdver*.

In our improved algorithm an additional delay might be added to the convergence time of regular BGP. In Figure 0, C is an arbitrary router on the preferred route from A to R1.

After T_1, C receives from B its last message that states that there is a valid route to *DEST* through edge (A,*DEST*). Suppose

that afterwards, C receives from another neighbor a delayed message that states that the edge (A,*DEST*) is not valid any more. Due to a topological change this neighbor will never re-announce that edge (A,*DEST*) is valid again. Thus, according to the improved algorithm, C's route to *DEST* through B will be suspended for *EdgeSuspentionInterval*. After *EdgeSuspentionInterval* the route through B will go out of suspension and can re-announce it.

We have described above a scenario in which an additional delay is added to the convergence time. Can there be any additional delays?

According to the above assumption, the time for an *Explicit Edge Withdrawal* message that origin from A router to reach any other router in the network is smaller than *EdgeSuspentionInterval.* Thus, after one occurrence of the above scenario, there will not be other *EdgeSuspentionInterval* delays related to edge (A,*DEST*).

Furthermore, oscillating edges or withdrawn edges that are not on the preferred route does not influence the maximum convergences time. They can cause a reset of the *MinRouteAdver* timer if there is a change on one of the alternative path or if *MinRouteAdver* is per peer and not per destination. But in our claim we already assumed that the announcement is delayed for *MinRouteAdver* at each router□

In the above claim we have proved that in the worst case scenario, when multiple event occur simultaneously, the enhanced algorithm convergence time is below: MinRouteAdver x D + EdgeSuspentionInterval (D is the diameter of the network). The enhanced algorithm adds at most EdgeSuspentionInterval to the convergence time.

VI. SIMULATION

In this section we compare the convergence time of BGP to the convergence time of the *explicit edge withdrawal* algorithm, by using a simulation. The simulation result ssupport our claim that the convergence time of our algorithm is O(D) instead of BGP's O(N).

The BGP algorithm we implemented in the simulation is a basic version of BGP as described in [7]. We assume no restriction on advertising or accepting messages from peers resulting from policies. All nodes are advertising and accepting routes from all of their peers. Each node chooses its best route to any destination according to the length of the route. Tie breaking of two equal length routes is made according to the ID (AS number) of the peer who advertises that route. Each node represents one AS (autonomous system). Each node adds one length unit to the route it advertises. In reality each AS can posses tens and hundreds of routers which exchange routing information using Intra-domain protocols. We neglected the additional length and latency caused.

The parameter *MinRouteAdver* is defined in [7] as the minimal amount of time that must elapse between advertising of routes to a particular destination from single BGP speaker. Usually this timer is implemented per peer and not per destination since it is impractical to keep a separate timer for each destination. In our simulation we set the *MinRouteAdver* to be per peer with elapse time of 30 seconds.

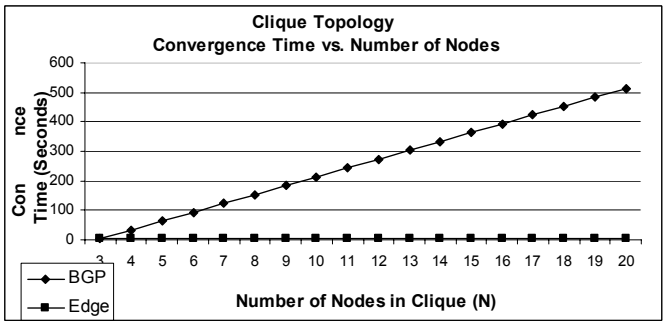


Figure 0– Clique Topology

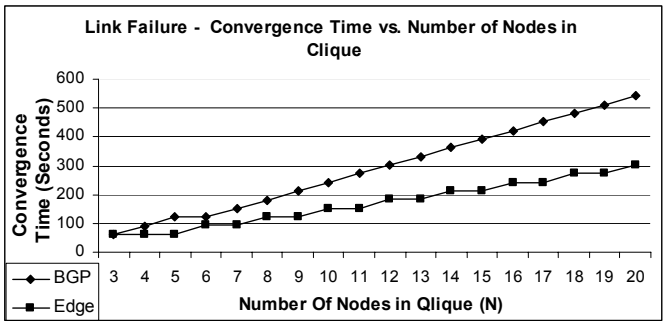


Figure 0– Link Failure

We initiated the *MinRouteAdver* timer with a random value, and we set the latencies of a packet transfer and processing of each packet to be random between 0 and 2 seconds. The results shown in the graphs are an average of multiple simulation runs under variable values of these parameters.

A. Simulation Results

1) Clique Topology

Our first scenario is depicted in **Figure 0**. A clique with N nodes is connected to node DEST through one of the nodes in the clique (AS0). At time 0, the connection between DEST and AS0 is removed. We measured the convergence time complexity of both algorithms as a function of the clique size.

Figure 0 shows that while BGP convergence time is O(n) (n – number of node in the clique), our algorithm in a clique topology is not dependent on the number of node. In a clique topology, the diameter is constant (1); thus our convergence time stays constant.

2) Link Failure Topology (failover)

In order measure the convergence time in the occurrence of a *failover* event we have used the topology shown in Figure 0. The topology consists of a clique of size N/2 and one alternative path to node DEST of length N/2. The shortest path through AS0 from DEST to the clique is removed at T=0. We have measured with both algorithms how long it takes the nodes in the clique to find the alternative path to DEST and vice versa.

BGP convergence time – 30xN-28.25.

Explicit edge withdrawal convergence time– 30xN/2-28.75.

As shown in Figure 0, while BGP convergence time depends on the number of nodes in the graph, the *explicit edge withdrawal* algorithm convergence time depends mainly on the length of the alternative path. Since we are using

MinRouteAdver timer per peer, and DEST has N/2 destinations

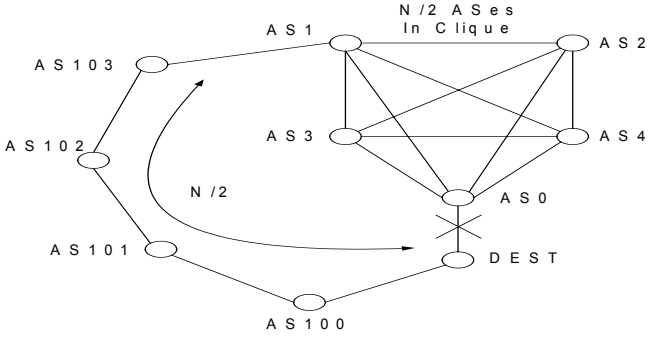


Figure 0 – Link Failue Topology

(AS0, AS1, AS2 ..) to reselect its route to them and each announcement conveys only one destination, messages are delayed on the alternative path. Once a node on the alternative path has declared a new path to any of the clique nodes, it has to wait 30 seconds before it can declare a new path to any other node. When changing the *MinRouteAdver* to be per destination, the convergence time of reduces to – (0.75xN+2.5), while BGP remains the same.

Furthermore, when using our algorithm, all the ghost information was removed from the network at time shorter than 5 seconds. The establishing of new valid path was the dominant factor in our algorithm and not the exploration of invalid paths.

B. Core Of The Internet

Finally, we compare the convergence times of the two algorithms on the core of the AS network taken from the Internet. We took the Routing Table of the route server [10] (which is a route server of 41 BGP routers) from consecutive days (4.12.00, 5.12.00, 6.12.00) and created from it the AS graph based on all the *ASpaths* of all the routes in the table. For example if there an ASpath 23, 43, 12 to destination *dest*, we add to the AS graph the directed edges (23, 43) and (43, 123). After building the AS graph, we have recursively removed nodes that their out-degree is zero. At the end we were left out with the core which includes 375 nodes with 2186 edges.

***AS* number**	**Out-degree AS**	**In-degree AS**	**BGP convergence time (seconds)**	**Our Algorithm convergence time (seconds)**
1	45	10	944	6.75
2	52	17	968	8
3	3	4	1135	7
4	112	27	997	7.75
5	61	11	1138	9
6	20	24	4	3
7	1	6	982	7.25
8	18	13	967	7.25
9	1	19	3	5.5
10	1	1	992	7.5

Table 1

We randomly chose ASes from the core, and simulated the failure of a destination (network) inside this AS. Table 1 shows comparison between the convergence time of BGP and our algorithm. It can be seen that our algorithm convergence time is much shorter than BGP. It can be explained by the fact that the core of the Internet has qualities of the clique.

Except *ASes* 6 and 9, the number of messages sent in the convergence process of BGP was above 50,000, while in our algorithm it was constant 4408 (2*E).

VII. CONCLUSIONS

In this paper we have introduced a method to cope with BGP convergence latency in a way that confronts with the essence of the problem. In the occurrence of a *faildown* or a *failover* event, the best way of eliminating the ghost information in the network is by flooding the exact description of the topology change to all the nodes that use the ghost information. The ghost information is the reason for the delayed convergence since it prevents nodes from choosing a valid path to destinations. From a node point of view the only topological changes that can be noticed is an *explicit edge withdrawal.* This withdrawal message conveys the information about the topological change and enables the routers to delete all the false information from their routing table.

We saw that using our method, the convergence time of BGP after a *faildown* or a *failover* has improved significantly. Furthermore, our method removes all ghost-information from the network in a very short time (<5 seconds in the worst case in our simulations). The fast removal of all ghost-information from the network has very strong implications. From that point on, all paths in the network are valid. No packets traverse the network without any feasibility of reaching their destination. In addition, our method decreases substantially the number of messages in the network until convergence.

Our algorithm can be regarded as an intermediate algorithm between BGP and link-state algorithms. We achieve better convergence time for the BGP protocol by using methods taken from link-state protocols.

REFERENCES

[1] T. Griffin, F. B. Shepherd, G. Wilfong, "Policy Disputes in Path-Vector Protocols," in Proceedings of ICNP '99 Conference, Toronto, Canada, October 1999.

[2] C. Labovitz, R. Wattenhofer, S. Venkatachary, and A. Ahuja, " The impact of internet policy and topology on delayed routing convergence," in Proc. Infocom, April 2001.

[3] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanianitz, "Delayed internet routing convergence," in Proc. Sigcomm, September 2000.

[4] T. Griffin, G. Wilfong, "An analysis of BGP convergence properties," in Proc. ACM SIGCOMM, Aug 2000.

[5] L. Gao and J. Rexford, "Stable internet routing without global coordination," in Proc. SGMETRICS, June 2000.

[6] T. Griffin, G. Wilfong, "A safe path vector protocol," in *Proc*. IEE INFOCOM, Mar 2000.

[7] Y. Rekhter and T. Li "A border gateway protocol 4 (bgp4)," Tech. Rep., IETF, 1999, draft-ietf-idr-bgp4-09.txt.

[8] D. Pei, Z. Zhao, L. Wang, D. Massey, A. Mankin, F. Wu, L. Zhang, "Improving BGP convergence through consistency assertions," In Proc. IEEE INFOCOM, 2002.

[9] T. Griffin, B. Premore, "An experimental analysis of BGP convergence time," in Proceedings of ICNP, Nov. 2001.

[10] Route server, "Host", Tech. Rep., Oregon, 2000, route-views.oregon.net.

[11] J. Luo, J. Xie, R. Hao and X. Li, "An approach to accelerate convergence for path vector protocol", accepted by GLOBECOM 2002.