2005

Stabilizing Inter-Domain Routing in the Internet

Yu Chen
University of Nevada, Las Vegas

Ajoy K. Datta
University of Nevada, Las Vegas

Sebastien Tixeuil
University of Paris

Journal of High Speed Networks 14 (2005) 21–37
IOS Press

Stabilizing inter-domain routing in the Internet [1]

Yu Chen [a], Ajoy K. Datta [a] and Sébastien Tixeuil [b,*]
[a] *School of Computer Science, University of Nevada, Las Vegas, USA*
[b] *LRI-CNRS UMR 8623 and INRIA Grand Large, Université Paris Sud, France*

Abstract. This paper reports the first self-stabilizing Border Gateway Protocol (BGP). BGP is the standard inter-domain routing protocol in the Internet. Self-stabilization is a technique to tolerate arbitrary transient faults.

The routing instability in the Internet can occur due to errors in configuring the routing data structures, the routing policies, transient physical and data link problems, software bugs, and memory corruption. This instability can increase the network latency, slow down the convergence of the routing data structures, and can also cause the partitioning of networks. Most of the previous studies concentrated on routing policies to achieve the convergence of BGP while the oscillations due to transient faults were ignored.

The purpose of self-stabilizing BGP is to solve the routing instability problem when this instability results from transient failures. The self-stabilizing BGP presented here provides a way to detect and automatically recover from this type of faults. Our protocol is combined with an existing protocol to make it resilient to policy conflicts as well.

Keywords: Border Gateway Protocol, routing, routing instability, self-stabilization

1. Introduction

Self-stabilization. The concept of self-stabilization was first introduced by Edsger W. Dijkstra in 1974 [4]. It is now considered to be the most general technique to design a system to tolerate arbitrary transient faults. A self-stabilizing system guarantees that starting from an arbitrary state, the system converges to a legal configuration in a finite number of steps, and remains in a legal state until another fault occurs (see also [6]). It is desirable that even if the error occurs rarely in the system, the networks should recover from those faults automatically [29].

In the context of computer networks, resuming correct behavior after a fault occurs can be very costly [21]: the whole network may have to be shut down and globally reset in a good initial state. While this approach is feasible for small networks, it is far from practical in large networks such as the Internet. Self-stabilization provides a way to recover from faults without the cost and inconvenience of a generalized human intervention: after a fault is diagnosed, one simply has to remove, repair, or reinitialize the faulty components, and the system, by itself, will return to a good global state within a relatively short amount of time.

Routing instability. Routing instability in the Internet has a number of origins including route configuration errors, transient physical and data link problems, software bugs, and sometimes memory corruption. The routing instability may increase the packet loss, resource overhead, and the delays in network convergence [19,28]. Moreover, in a recent work of Varghese and Jayaram (see [30]), it is proven that the crash failures of routers can lead every other router in the network to an arbitrary state, and that if links can reorder and lose messages, then any incorrect global state is reachable. Therefore, it is very important to have a self-stabilizing [4] routing protocol which can recover from any arbitrary faults without any external intervention.

[1] An abstract of this paper appeared in [3].

[*] This author is supported in part by the French Ministry Grant SR2I.

The Border Gateway Protocol (BGP) is the standard inter-domain routing protocol [24] in the Internet. It is used to exchange network reachability information among autonomous systems (ASs). The BGP gained a lot of popularity in the last few years due to the significant growth in the number of ISPs, many of which must use BGP to connect to other ISPs [22,25]. The availability and stability of the connection has a large effect on the stability of the Internet.

Related work. The behavior and dynamics of the Internet routing stability has not been extensively studied with the exception of [19,28]. In 1997 [19], the Internet routing instability was investigated based on the data collected from BGP routing messages generated by border routers at five Internet public exchange points during a nine month period. Overall, the study showed that the Internet continued to exhibit high levels of routing instability despite the increased emphasis on aggregation (combining smaller IP prefixes into a single route announcement) and the deployment of route dampening technology (refusing to believe the updates that exceed certain parameters of instability). The instability significantly contributes to poor end-to-end network performance and degrades the overall efficiency of the Internet infrastructure. The high level of Internet instability can lead to increased network latency, and slower convergence of the routing data structures. At the extreme, the high-level Internet instability can lead to loss of internal connectivity in wide-area national networks. This definitely will be a disaster for all e-commerce business and countless end-users all over the world.

So far, the research about BGP focused on the *stability* of the protocol. Assume that a computer network is started from an initial coherent state (where known paths to the destination do exist, where routers are properly configured, etc.), then a particular BGP instance (i.e., a particular route policy configuration of BGP in the network) is stable if it reaches a final global state where all nodes have a stable path towards the destination. Unfortunately, knowing if a particular instance of BGP is stable is an NP-complete problem (see [14]). Thus, proposals to guarantee stability include a global sufficient condition on the system (see [13]), a local condition on the BGP routers' policies (see [9]), and a dynamic additional algorithm that is run on each BGP router (The Safe Path Vector Protocol as defined in [15]).

In this paper, we rather focus on the *self-stabilizing* properties of the protocol. Assume that a computer network is started from an arbitrary initial configurations, (where known paths can be arbitrary and where routers can be completely misconfigured), then a self-stabilizing BGP solution guarantees, as soon as faults cease, that the network eventually reaches an initial coherent state, and then a stable final configuration (assuming a stable BGP is run). As self-stabilizing BGP subsumes stable BGP, the self-stabilizing BGP problem is also NP-complete. None of the approaches we mentioned (that guarantee BGP stability) is self-stabilizing. The algorithm we propose in this paper is a self-stabilizing BGP. Our work provides a way to detect and automatically recover from transient faults.

Network topology maintenance is an important component of Internet routing. A lot of research has been done in this area [10,17,26]. Nodes/links failures directly cause the network topology changes, which implicitly introduce the routing instability. Since network topology maintenance protocol is the underlying protocol for most of the routing protocols, its stability is very important. The topology update problem has been discussed in [1,5,7,12,20].

The correct routing information in an autonomous system helps BGP achieve a stable routing among the autonomous systems. Numerous intra-domain routing schemes were developed, and the Open Shortest Path First [10,22] routing protocol is one of them. It has been used as the routing protocol in the autonomous system in most of the systems due to its fast and loop-less convergence property. OSPF constructs a spanning tree to reliably flood the link state packets, and a shortest path tree to compute the best neighbor to reach any other node from the source node. An algorithm for self-stabilizing shortest path tree construction is presented in [27]. The self-stabilizing spanning tree construction algorithms appeared in [2,8].

Contributions. None of the previous work on BGP is self-stabilizing. In this paper, we present the first self-stabilized Border Gateway Protocol (called SBGP hereafter). The key advantage of BGP protocol being self-stabilizing is that regardless of the current state of the processors and channels, after the faulty components are removed, replaced, or recovered, the protocol will resume behaving normally within a reasonably short time. Our approach in this paper consists of two layers:

– A self-stabilizing synchronization mechanism (the counter-flushing mechanism) ensures that starting from any initial configuration, the system eventually reaches a configuration where messages are sent and received in the right order. Thus, routers eventually benefit from a reliable communication scheme to communicate with their peers. Special care is taken to enable interoperation of our protocol with non self-stabilizing BGP routers (legacy routers).
– The Safe Path Vector protocol of [15] is run on top of the first protocol (thus assuming reliable communications) to ensure the stability of the routing metrics. The dynamic approach of [15] was preferred over the static approaches of [13] and [15] since it is suited for dynamically evolving networks, while the two static approaches require some knowledge about the system policies.

Our algorithm requires the time $O(IDiam)$ to stabilize, where $IDiam$ is the maximum diameter of an autonomous system. If we assume that the degree of each router and the capacity of each data link is significantly smaller than the number of autonomous systems, then the memory used at each router is $O(NBR \log NBR)$, where NBR is the number of autonomous systems. We assume that the SBGP underlying routing algorithm within autonomous systems is OSPF, since the Link State Protocol it is based upon which was proven self-stabilizing by Lynch (as reported in [21]). However, other similar algorithms (such as RIP) would be acceptable as long as they retain the self-stabilizing properties of OSPF.

Outline of the paper. The remainder of the paper is organized as follows. In Section 2, we present the system model and definitions that will be used throughout the paper. We also define the inter-domain routing problem there. In Section 4, we describe (both informally and formally) our self-stabilizing version of the BGP protocol, along with the proof of correctness. Section 5 provides some concluding remarks.

2. Model

Distributed system. A *distributed system* is an undirected connected graph, $S = (V, E)$, where V is a set of nodes ($|V| = n$) and E is the set of edges. Nodes represent *routers*, and edges represent *bidirectional communication links*. We will use "nodes" and "routers" interchangeably in this paper. A communication link (p, q) exists iff p and q are neighbors. Nodes communicate only by message passing [18]. The message delivery time is arbitrary and finite but unbounded. We assume FIFO channels (in real networks, such channels can be implemented through reliable transport protocols, such as TCP).

The *state* of a router is defined by the value of its variables. The *state* of a system is a vector of $n+1$ components where the first n represent the state of n routers, and the last component refers to the set of messages (denoted by a multi-set $\mathcal{M}$) in transit in the links. In the following, we refer to the state of a router and system as a (*local*) *state* and *configuration*, respectively. Let a distributed protocol $\mathcal{P}$ be a collection of binary transition relations denoted by $\mapsto$, on $\mathcal{C}$, the set of all possible configurations of the system. A *computation* of a protocol $\mathcal{P}$ is a maximal sequence of configurations $e = \gamma_0, \gamma_1, \ldots, \gamma_t, \gamma_{t+1}, \ldots$, such that for $t \geqslant 0, \gamma_t \mapsto \gamma_{t+1}$ (a single *computation step*), if γ_{t+1} exists, or γ_t is a terminal configuration. *Maximality* means that the sequence is either infinite, or it is finite and no action of $\mathcal{P}$ is enabled in the final configuration. All computations considered in this paper are assumed to be maximal. The set of computations of a protocol $\mathcal{P}$ in system S starting with a particular configuration $\alpha \in \mathcal{C}$ is denoted by $\mathcal{E}_\alpha$. The set of all possible computations of $\mathcal{P}$ in system S is denoted as $\mathcal{E}$.

Program. During a computation step, we assume that at least one router executes at least one of its possible composite actions. A composite action of a router p consists of three phases: (i) p receives a message or is waken up by a timeout mechanism (this enables the composite action), (ii) p executes some internal statements, and (iii) p sends at least one message. The composite actions of each router are assumed to be fairly scheduled: if enabled (either by the receipt of a message of by a timeout), it is eventually executed by the router.

The timeout mechanism is needed for stabilization purpose. In [11], it was proven that any self-stabilizing protocol where processors communicate by a message passing mechanism must assume at least one timeout action

in at least one processor. The timeout action is used to recover from different types of faults in the network – messages may be lost and the system may start with no messages in any channel.

We use the notations introduced in [10] to describe the routers. The definition of a router consists of two parts. Messages, inputs, variables, parameters, macros, and procedures of the router are declared in the first part. In the second part, actions of the router are defined. The statements of a router are of four types: assignment, sending, selection, and iteration. An assignment statement of p is of the form: $x_p := E_p$ where x_p is a variable of p and E_p is a constant or expression of the same type as x_p. A sending statement of p is of the form: SEND $< message\_type >$ to $< receiving\_process\_name >$. A selection statement of p is of the form: **if** ... **endif**. An iteration statement of p is of the form: **for** ... **endfor**.

Self-stabilization. Let $\mathcal{X}$ be a set. $x \vdash P$ means that an element $x \in \mathcal{X}$ satisfies the predicate P defined on the set $\mathcal{X}$. A predicate is non-empty if there exists at least one element that satisfies the predicate. We define a special predicate true as follows: *for any* $x \in \mathcal{X}$, $x \vdash$ true. We use the following term, *attractor* in the definition of self-stabilization.

Definition 1 (Attractor). Let X and Y be two predicates of a protocol $\mathcal{P}$ defined on $\mathcal{C}$ of system $\mathcal{S}$. Y is an attractor for X if and only if the following condition is true: $\forall \alpha \vdash X : \ \forall e \in \mathcal{E}_\alpha : \ e = (\gamma_0, \gamma_1, \ldots) :: \ \exists i \geqslant 0, \forall j \geqslant i, \gamma_j \vdash Y$. We denote this relation as $X \triangleright Y$.

Definition 2 (Self-stabilization). The protocol $\mathcal{P}$ is self-stabilizing for the specification $\mathcal{SP}_\mathcal{P}$ on $\mathcal{E}$ if and only if there exists a predicate $\mathcal{L}_\mathcal{P}$ (called the legitimacy predicate) defined on $\mathcal{C}$ such that the following conditions hold: (i) $\forall \alpha \vdash \mathcal{L}_\mathcal{P} : \ \forall e \in \mathcal{E}_\alpha :: \ e \vdash \mathcal{SP}_\mathcal{P}$ (correctness), and (ii) true $\triangleright \mathcal{L}_\mathcal{P}$ (convergence).

3. BGP overview

Unlike many IGPs built reliably on the datagram services, BGP uses TCP as its transport protocol. Implementing this way provides reliable communication and hides all the details of the network being transmitted over. Two BGP routers establish a TCP connection by exchanging messages, confirm the connection parameters, and report any route changes to each other. These two BGP routers are known as peers or neighbors. BGP allows each autonomous system to enforce its routing policies independently. These policies are related to political, security, or economic consideration [23], and will affect the route selection and redistribution of the route information.

For BGP routers that are located in different ASs, they usually share a common physical data link, which means that they are directly connected. BGP routers within an AS do not have to be directly connected. BGP routers within the same AS exchange update messages by running Internal BGP (IBGP), and BGP routers located in different ASs run External BGP (EBGP) to exchange the reachability information. Figure 1 shows an example of IBGP and EBGP, presenting three autonomous systems A, B, and C. Inside an autonomous system are routers (e.g., A_0, A_3 in AS A) that are either directly connected (denoted by a full line as between A_2 and A_3) or virtually connected (denoted by a dotted line as between A_0 and A_3) using IBGP. Among autonomous systems, routers need to be directly connected (denoted by a full line as between A_3 and C_0), and they communicate using EBGP.

IBGP. In Fig. 1, Routers A_0 and A_3 belong to the same autonomous system, A. There must exist an IBGP connection between them if two routers need to exchange the routing information. But, the routes heard from an IBGP peer cannot be advertised to other IBGP routers to prevent the looping route announcement within the AS [25]. Therefore, in order for all the BGP routers within an AS to exchange routing information, there must be an IBGP connection between every pair of routers. It is obvious that the full-mesh connections among the IBGP routers scale very poorly. Two approaches are employed so that the IBGP connections become more scalable: router reflection and AS confederation [25].

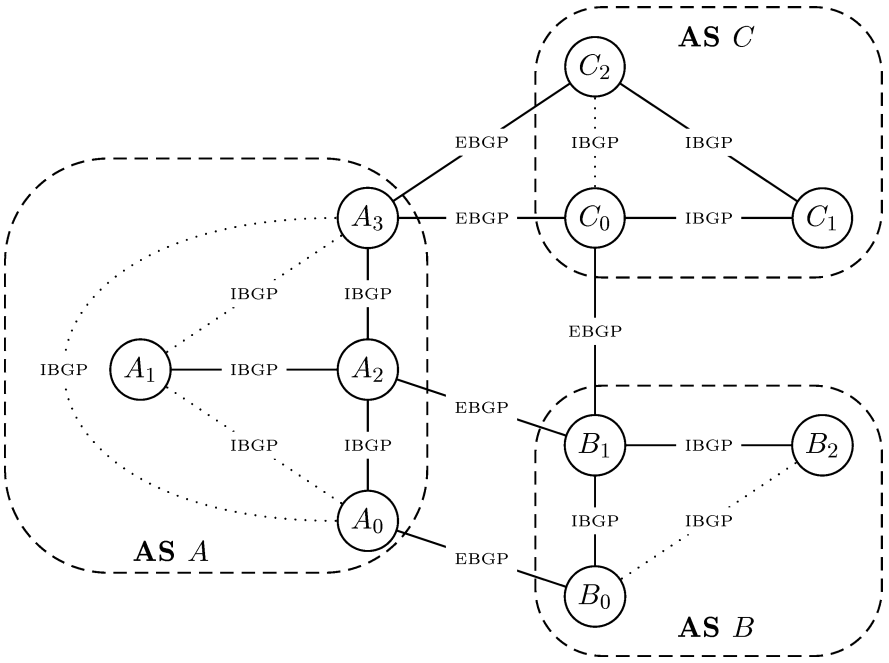

Fig. 1. IBGP and EBGP.

EBGP. Two BGP routers that are in different ASs are generally directly connected, and this EBGP connection carries the transit traffic between them. Unlike IBGP connections, the routes collected from an EBGP peer can be advertised to an IBGP neighbor and vice versa [24]. Before it announces a route to an external autonomous system, a BGP router must ensure that networks within the AS are reachable. This is done by a combination of IBGP peering within the AS and by redistributing BGP routing information to IGPs (as is done in OSPF) that run within the ASs. Therefore, when an AS carries traffic from one AS to another, it will not advertise a route until all the routers within the AS have learned about it via an IGP.

Specification of inter-domain routing problem. We define NEXT_BR as the IP address of the border router that should be used as the next closest neighbor towards a specific destination.

Specification 1. *We consider a computation* e *of the inter-domain routing problem to satisfy the specification* $\mathcal{SP}_{\mathcal{IDR}}$ *when the routing tables* (i) *do not contain any information about the unreachable nodes in the system and* (ii) *contain the IP address of* $NEXT\_BR$ *from which a specific destination can be reached.*

4. Stabilizing BGP

In Section 3, we discussed the possibility of routing instability in the Internet and the importance of a self-stabilizing routing protocol. We now present a self-stabilizing BGP algorithm, called Algorithm $SBGP$. The algorithm starts in an arbitrary state without any initialization. A node in Algorithm $SBGP$ adds to any BGP UPDATE message an integer counter and uses the well-known Counter Flushing scheme (see [29]) to synchronize with other BGP routers. Acknowledgments are sent for each UPDATE message received by a node. These acknowledgment messages also follow the counter flushing scheme to synchronize with the UPDATE messages. A part of our algorithm that eliminates the possibility of the routing oscillation is an adaptation of the $SPVP$ algorithm proposed in [13,14,16]. Since the core of our protocol guarantees that the routing information is eventually

correctly transmitted among the $SBGP$ routers (by the self-stabilizing property of our algorithm), it satisfies the condition for a good initial configuration of the $SPVP$ protocol. Then the main theorem of [16] guarantees that route oscillation cannot occur. Thus, overall we get a self-stabilizing and route oscillation-free BGP-like protocol.

We organize this section as follows: We first present an outline of Algorithm $SBGP$. Then we introduce the data structure of the algorithm, along with the helper procedures and macros, followed by the actual algorithm code. A simple algorithm for user packet routing is also proposed. Finally, we provide the proof for Algorithm $SBGP$ and its complexity analysis.

4.1. Overview of algorithm $SBGP$

The abstract version of the $SBGP$ algorithm is presented as Algorithm 4.1. A router p may have two different types of neighbors: internal and external. An internal neighbor q may not be directly connected to p, but is in the same autonomous system as p. We call q an $Internal\_Peer$ of p. An external neighbor q' is directly connected to p, but resides in a different autonomous system than p. We refer to q' as an $External\_Peer$ of p.

In the BGP standard [25], it is possible that external peers are not directly connected because having a static route to each external peer is sufficient. Our SBGP algorithm could handle this case provided that those static routes to external peers are accurate all the time. However, this hypothesis is unrealistic most of the time since manual intervention is required to define static routes. When the network topology changes, a manual intervention is required, and the assumption of our algorithm is not satisfied. Then the existence of a self-stabilizing protocol is compromised until the manual intervention occurs. For that reason, we assume the simpler case where external peers are directly connected. A change of topology is typically handled by an underlying self-stabilizing topology maintenance protocol.

When an $SBGP$ router wants to send a message to its internal peers or external peers, it checks its $Interal\_Peers$ or $External\_Peers$ sets to make sure that the corresponding peer router is active. We now explain how each $SBGP$ router p maintains the sets $Internal\_Peers_p$ and $External\_Peers_p$. From the underlying OSPF protocol, p gets the set $Internal\_Peers_p$. From the local topology maintenance algorithm (e.g., of [7]), p obtains the set of its neighbors, and hence obtains the set $External\_Peers_p$.

Algorithm 4.1 ($SBGP$) Abstract Version of Stabilizing BGP

```
ASBGP.01 upon receipt of an update message from neighbor q
ASBGP.02     if my counter value and the received counter value are different then
ASBGP.03         Save the message in the waiting queue;
ASBGP.04         Send an acknowledgment back to q;
ASBGP.05         if the previous broadcasting cycle is done then
ASBGP.06             Update the set of routes received from q;
ASBGP.07             Decision Process;
ASBGP.08         endif
ASBGP.09     else
ASBGP.10         Send an acknowledgment back to q;
ASBGP.11     endif
ASBGP.12 upon receipt of an acknowledgment message from external peer q
ASBGP.13     Record the acknowledgment;
ASBGP.14 upon receipt of an IP packet from internal peer q
ASBGP.15     if I am the destination then
ASBGP.16         Decapsulate the IP packet;
ASBGP.16         Run the update part or the acknowledgment part of the algorithm;
ASBGP.17     else
ASBGP.18         Forward the IP packet to the best neighbor towards the destination using OSPF routing scheme;
ASBGP.19     endif
```

Algorithm $SBGP$ is message reactive (apart from the timeout mechanism needed as shown by [11] and that is not presented in Algorithm 4.1) and aware of three kinds of messages. Update and acknowledgment messages (Lines ASBGP.01–ASBGP.13 of Algorithm 4.1) are used to implement a self-stabilizing *cycle of broadcasting* by using the counter flushing mechanism of [29]. sends update messages to keep the routing tables up-to-date, and expects acknowledgment from all external peers. A counter is added to each message so that eventually, messages that do not belong to the current cycle of broadcasting are removed from the network. The third type of message is the IP packet (Lines ASBGP.14–ASBGP.19 of Algorithm 4.1) that may be used within an AS to carry update or acknowledgment messages among routers that do not run Algorithm $SBGP$. Thus, this message ensures interoperability with legacy equipment. From here on, a cycle of broadcasting is simply referred to as a cycle.

4.2. Detailed description

4.2.1. Messages

There are three kinds of messages exchanged among the $SBGP$ routers: the UPDATE message, the ACK message, and the encapsulated IP packet. An $SBGP$ router may receive different types of messages at the same time. The reason we use the IP packet is the following: $SBGP$ peers inside one AS may not be directly connected with each other. In order to prevent looping of the routing announcements within the AS, when an $SBGP$ router receives a route from another $SBGP$ peer located in the same AS, it cannot re-advertise the route to other $SBGP$ peers in the same AS. Thus, a full-mesh connection is needed among all $SBGP$ routers in the same AS. An $SBGP$-level message may traverse among the non-SBGP routers before it reaches the final destination. The encapsulation hides the details of the actual message so that the encapsulated messages look like normal IP packets to the non-$SBGP$ routers. For the sake of simplicity, we assume that the UPDATE and ACK messages are used to exchange information among the external peers, and the encapsulated IP packet to exchange information among the internal peers. These messages can be further described as follows:

UPDATE contains the following fields: c and RIB. c contains the counter of the current cycle initiated by the sender. RIB consists of the routes being broadcast. The UPDATE message piggybacks the RIB to reduce the traffic in the system.

ACK contains one field: c (as defined above).

IP is an encapsulated message. It has two fields: sd and rv, the id of the sender and destination, respectively. The SBGP router encapsulates the UPDATE and ACK messages into ordinary IP packets so that the non-$SBGP$ routers will be able to receive them. Since the final destination of a message or a packet is always an $SBGP$ router, the (intermediate) non-$SBGP$ routers do not have to interpret these messages at all. They just need to forward the IP packets to their best neighbors towards the specific destinations.

4.2.2. Router variables

Inputs. We assume that several inputs (constants for our purpose) are available at every SBGP router p. The inputs MAX and NBR (representing two bounds as described below) need to be known for the counter flushing mechanism [29] to stabilize. The $Internal\_Peers$ variable is a projection of the result of the (stabilized) OSPF algorithm which is run in the autonomous system. The $External\_Peers$ variable is a projection of the result of the (stabilized) Update algorithm of [7] which is run between SBGP nodes that belong to neighboring autonomous systems. The available inputs are as follows:

$External\_Peers_p$. The set of neighbors of p (i.e., the nodes directly connected to p), but not in the same autonomous system as p.

$Internal\_Peers_p$. The set of routers in the same autonomous system as p. Note that these routers may not be directly connected to p.

MAX. The upper-bound of the link capacity.

NBR. The upper bound of the number of routers.

$MyAs$. The id of the autonomous system.

Variables. In addition, every $SBGP$ router p maintains the following local variables: (Note that these variables may be corrupted due to arbitrary transient faults.)

$Counter_p$. List of tuples (id, $Counter$). id is the node identifier and $Counter$ is an integer in the range $0 \ldots MAX$. It keeps track of the counter values used in a different cycle initiated by a specific node. $Counter_p(i, 5)$ denotes that the current cycle initiated by node i has the counter value 5.
$EAcked_p$. Set of external $SBGP$ routers from which an acknowledgment was received at node p in the cycle initiated by p.
$IAcked_p$. Set of internal $SBGP$ routers from which an acknowledgment was received at node p in the cycle initiated by p.
$Finished_p$. Boolean variable. $Finished_p =$ true indicates that the cycle initiated by p is finished successfully and a new cycle at p can start, if necessary.
Msg_Queue_p. Queue to store the new UPDATE messages from the external SBGP peers. The queue is implemented in a FIFO fashion to ensure the fairness.
$Best\_Routes_p$. Set of routes chosen as the preferred routes for every distinct destination in $INTERNAL\_UPDATE$ procedure.
$Routes\_Selected_p$. Set of routes chosen as the preferred routes for every distinct destination in $LOCRIB\_UPDATE$ procedure.
$Adj\_RIB\_In_{qp}$. Set of routes received from peer q.
$Adj\_RIB\_Out_{pq}$. Set of routes to be broadcast to peer q.
LOC_RIB_p. Set of routes selected by the local $SBGP$ router's Decision Process. (See lines SBGP.07–SBGP.12 in Algorithm 4.7.)
P. Route in RIBs.
$P.NLRI$. IP address of the destination.

4.2.3. Macros and helper functions

Macros. Every $SBGP$ router p uses the macros described in Table 1 locally. We omit the specific details of the macros because they are simple and not critical to our work. Every $SBGP$ router p locally uses helper procedures described below.

$INTERNAL\_UPDATE$. This procedure (Algorithm 4.2) will be invoked only when a new route is received from an external peer. After the Adj_RIB_In buffer is updated (Procedure $UPDRIB$), each route to a specific destination is assigned a value (LOC_PREF), based on the routing policies and other constraints. The route with the highest LOC_PREF will be chosen and advertised to all the SBGP routers inside the autonomous system.

$UPDRIB$. This procedure (Algorithm 4.3) updates Adj_RIB_In buffer. When an $SBGP$ router p receives a route update message from an external peer, it compares the received routes with the routes in its Adj_RIB_In buffer. The routes in the buffer are replaced only when there are new routes to the same destination. p also updates the attributes of the accepted routes. If the route update message is from an internal peer, p only updates Adj_RIB_In, if necessary, without changing the attributes of the routes.

Table 1
$SBGP$ macro description

Macro	Description
ENCAP	This macro encapsulates a $SBGP$ level UPDATE or an ACK message into a regular IP packet.
DECAP	This macro decapsulates an IP packet into an $SBGP$ level UPDATE or ACK message.
IGP_LOOKUP	This macro returns the best neighboring router in the autonomous system towards the specific destination. It takes the IP address of the destination as the input, and then uses the OSPF's routing information (which can be found in $Best\_Nbr$ in every OSPF router) to decide the IP address of the best neighbor.
ENQUEUE	Adds a new UPDATE message to the tail of the message queue.
DEQUEUE	Removes an UPDATE message from the head of the message queue.

Algorithm 4.2 ($INTERNAL\_UPDATE$) Advertise Selected Routes Within the AS

Variables: br_p: temporary buffer to store the selected route
Macros: BEST: selects the best route for a certain destination from all the routes received from the external peers
CAL_PREF: assigns the degree of preference based on the routing policiess

IUPDATE.01 $Adj\_RIB\_In_{qp}.P.LOC\_PREF :=$ CAL_PREF($Adj\_RIB\_In_{qp}.P.NLRI$);
IUPDATE.02 $br_p :=$ BEST(); $br_p := SPV(br_p)$;

Algorithm 4.3 ($UPDRIB$) Update Adj_RIB_In Buffer

Variables: $P.NEXT\_BR$: IP address of the best neighboring route to the destination
$P.AS\_PATH$: a sequence of integers to represent the ASs the route has traversed
Macros: APPEND: appends its own AS number to the attribute AS_PATH

UPDRIB.01 **if** $q \in External\_Peers_p$ **then**
UPDRIB.03 **for each** $Msg.RIB_{qp}.P.NLRI$ **do**
UPDRIB.04 **if** $(Msg.RIB_{qp}.P \neq Adj\_RIB\_In_{qp}.P) \vee (Msg.RIB_{qp}.P.NLRI \notin Adj\_RIB\_In_{qp})$ **then**
UPDRIB.06 $Adj\_RIB\_In_{qp}.P := Msg.RIB_{qp}.P$; $Adj\_RIB\_In_{qp}.P.NEXT\_BR := p$;
UPDRIB.08 $Adj\_RIB\_In_{qp}.P.AS\_PATH := APPEND(MyAS_p, Adj\_RIB\_In_{qp}.P.AS\_PATH)$;
UPDRIB.09 **endif**
UPDRIB.10 **endfor**
UPDRIB.11 **elseif** $q \in Internal\_Peers_p$ **then**
UPDRIB.13 **for each** $Msg.RIB_{qp}.P.NLRI$ **do**
UPDRIB.14 **if** $(Msg.RIB_{qp}.P \neq Adj\_RIB\_In_{qp}.P) \vee (Msg.RIB_{qp}.P.NLRI \notin Adj\_RIB\_In_{qp})$ **then**
UPDRIB.16 $Adj\_RIB\_In_{qp}.P := Msg.RIB_{qp}.P$;
UPDRIB.17 **endif**
UPDRIB.18 **endfor**
UPDRIB.19 **endif**

$LOCRIB\_UPDATE$. This procedure (Algorithm 4.4) takes all the routes stored in Adj_RIB_In. These routes were received from two sources: $SBGP$ routers in the same autonomous system as p and $SBGP$ routers in the neighboring autonomous systems. This procedure selects the best route for every destination, and stores them in LOC_RIB, replacing the current route to the same destination.

$OUTMSG$. This procedure (Algorithm 4.5) formats the outgoing messages. When a message is sent to $SBGP$ peers inside the autonomous system, the routes can be simply copied into the messages. But, when a message is sent to $SBGP$ peers outside the autonomous system, some attributes of the routes, such as LOC_PREF and MED, need to be changed. In both cases, the counter of the sent message is updated to reflect the current value of the router counter.

SPV. This procedure (Algorithm 4.6) captures and suppresses routes that contain cycles due to routing policy conflicts. Details of this protocol can be found in [13,14,16].

4.2.4. Algorithm

As described in Section 4.1, our $SBGP$ algorithm is message-driven, i.e., the processors execute upon receiving UPDATE, ACK, or IP messages. But, in order to satisfy the stabilizing property, the algorithm must use a timeout action (presented as Algorithm 4.8) (see [11] for more explanation of the timeout action). We now present the behavior of a $SBGP$ router p upon receiving the following three types of messages.

Algorithm 4.4 ($LOCRIB\_UPDATE$) Update LOC_RIB Buffer

Variables: $Route_p$: temporary storage for a route
Macros: SELECTION: selects the best route to a certain destination from all the routes received from both the internal and external peers according to the route selection rules

LOCRIBUPD.01 **for each** $P.NLRI$ **do** /* For each route with a distinct destination */
LOCRIBUPD.03 $Route_p := SPV$(SELECTION());
LOCRIBUPD.04 **if** $Route_p \notin LOC\_RIB_p$ **then**
LOCRIBUPD.06 $LOC\_RIB_p := LOC\_RIB_p \cup \{Route_p\}$;
LOCRIBUPD.07 **endif**
LOCRIBUPD.08 $Adj\_RIB\_Out_{pq} := Adj\_RIB\_Out_{pq} \cup \{Route_p\}$;
LOCRIBUPD.09 **endfor**

Algorithm 4.5 ($OUTMSG$) Format Outgoing Messages

Inputs: RIB: set of routes that need to be advertised
Variables: $P.LOC\_PREF$: integer, the degree of preference of the route
$P.MED$: integer, the preference of an entry/exit point of an AS

OUTMSG.01 **if** $q \in External\_Peers_p$ **then**
OUTMSG.03 **for each** $RIB.P$ **do**
OUTMSG.04 $Msg.c := Counter_pp$; $Msg.RIB_{pq}.P.LOC\_PREF := dlp$; $Msg.RIB_{pq}.P.MED := 0$;
OUTMSG.07 **endfor**
OUTMSG.08 **elseif** $q \in Internal\_Peers_p$
OUTMSG.09 **for each** $RIB.P$ **do**
OUTMSG.10 $Msg.c := Counter_pp$; $Msg.RIB_{pq}.P := RIB.P$;
OUTMSG.12 **endfor**
OUTMSG.13 **endif**

UPDATE Message. When a router p receives an UPDATE message from a router q, it identifies q as an $SBGP$ peer located in a different AS. It compares the counter value c in the UPDATE message with its own counter value $Counter_pq$ corresponding to the cycle initiated by q. If they are the same, p simply sends an acknowledgment back to q. If not, p saves the message in the message waiting queue. If the value of the variable $Finished_p$ is true, the previous cycle initiated by p is finished and a new cycle can start. p then removes one message from the head of the queue and starts to process the message. Assuming that this message is from an external peer k, p changes the counter value as $Counter_pk = c$, sends an acknowledgment to the sender of the UPDATE message, and updates (Procedure $UPDRIB$) the $Adj\_RIB\_In_{kp}$ based on the RIB contained in the UPDATE message. p then invokes the Decision Process (Lines SBGP.07–SBGP.12) for the route selection. In Lines SBGP.07–SBGP.09, p calculates the degree of preference for each route and advertises the route with the highest degree of preference for each distinct destination to all the $SBGP$ peers in the same AS (Procedures $INTERNAL\_UPDATE$ and SPV). In Lines SBGP.10–SBGP.12, p selects the best route out of all the routes stored in the Adj_RIBs_In (the routes received from $SBGP$ peers located in its own AS and in the neighboring ASs), and stores them into LOC_RIB_p (Procedure $LOCRIB\_UPDATE$). In Line SBGP.11, the newly selected routes in LOC_RIB_p are stored into the corresponding entries in the associated Adj_RIBs_Out, and advertised to each peer located in a neighboring AS. p sets the variable $Flag_p$ to Internal so that the new changes will be sent to all its internal peers first.

ACK Message. When a router p receives an acknowledgment from one of its external peers, it first checks if the counter value in the message is the same as that in p. If they are the same, p records the id of the sending router,

Algorithm 4.6 (SPV) Simple Path Vector

Inputs: br_p: the route that is chosen as the best route to a certain destination
Variables: Old_p: the previous route to a certain destination
New_p: the new route that is chosen as the best route to a certain destination
$History\_New$: the new path history
Bad_Path: set of routes whose history lists contain a cycle
$Route_p$: temporary buffer to store the selected route

Macros: HIST: computes a new path history
CYCLE: detects if a certain history list contains a cycle
BEST: computes the best route to a certain destination

SPV.01 **if** $br \in LOC\_RIB \wedge br \neq LOC\_RIB_p.P$ **then**
SPV.03 $Old_p := LOC\_RIB_p.P$; $New_p := br$; $History\_New_p :=$ HIST($LOC\_RIB_p.P$);
SPV.06 **if** CYCLE($History\_New_p$) = true **then**
SPV.08 $Bad\_Path_p := Bad\_Path_p \cup \{New_p\}$; $New_p :=$ BEST();
SPV.10 **if** $New_p \neq Old_p$ **then**
SPV.12 $History\_New_p := (-, Old_p)$;
SPV.13 **endif**
SPV.14 **endif**
SPV.15 **if** $New_p \neq Old_p$ **then**
SPV.17 $Route_p := (New_p, History\_New_p)$; $LOC\_RIB_p := LOC\_RIB_p \setminus \{ Old_p\}$;
SPV.19 **endif**
SPV.20 **endif**

and checks if it has received acknowledgments from all its current active external peers. If so, p sets $Finished_p$ to true so that a new cycle can start, and sets $Flag_p$ to Internal (Line SBGP.22).

IP Message. When a node p receives an IP packet from a node q, it recognizes that the packet is from either an $SBGP$ peer or a non-$SBGP$ router located in its own AS. p checks if the packet is designated for itself. If not, p simply forwards the packet to its best neighbor towards the destination (Lines SBGP.41–SBGP.43). If p itself is the actual receiver of the packet, p can interpret the packet by decapsulating it. The message contained in the IP packet may be an UPDATE message or an ACK message. If it is an UPDATE message, p needs to distinguish the new message from the old message by comparing the counter value in the message with its own value. If the message is new, p accepts the message and updates its $Adj\_RIB\_In_{qp}$. In this case, p will not readvertise the updates to its internal peers. p simply encapsulates the acknowledgment message and sends the IP packet to the actual sender of the IP packet (Lines SBGP.28–SBGP.31). If it is an acknowledgment message, p records the actual sender and checks to see if it has received all the acknowledgments (Lines SBGP.34–SBGP.36). Variable $Flag_p$ is set to External (Line SBGP.37) so that p can broadcast the new changes to its external peers.

Upon timeout expiration. When a timeout expires at router p, no new internal computation is done, but messages that reflect the current status of the router are being sent to every internal peer and every external peer. This costly behavior is to be executed only once after a transient failure and only in the special case where no message is initially present in the system. If there exists at least one message initially, no timeout mechanism is used. The timeout period needs to be adjusted so that it does not interfere with the normal behavior of the algorithm. However, the detail discussion of this implementation is beyond the scope of this paper.

4.2.5. User data packet routing

User packets are transmitted using TCP which is a reliable transport protocol. So, we can safely assume that a message will eventually arrive at its destination.

Algorithm 4.7 ($SBGP$) Border Gateway Protocol for Process p (message responses)

```
SBGP.01  upon receipt of UPDATE(c, RIB_qp) from q
SBGP.02      if q ∈ External_Peers_p ∧ Counter_pq ≠ c then
SBGP.03          ENQUEUE(UPDATE(c, RIB_qp));
SBGP.04          SEND ACK(Counter_pq) to q;
SBGP.05          if Finished_p = true then
SBGP.06              DEQUEUE(); Counter_pk := c; UPDRIB(k);
SBGP.07              for each Adj_RIB_In_kp.P.NLRI do
SBGP.08                  Best_Routes_p := INTERNAL_UPDATE() ∪Best_Routes_p;
SBGP.09              endfor
SBGP.10              for all Adj_RIB_In_kp do
SBGP.11                  Routes_Selected_p := LOCRIB_UPDATE() ∪Routes_Selected_p;
SBGP.12              endfor
SBGP.13              Finished_p := false; Flag_p := Internal; Counter_pp := (Counter_pp + 1) mod MAX;
SBGP.14          endif
SBGP.15      else
SBGP.16          SEND ACK(Counter_pq) to q;
SBGP.17      endif

SBGP.18  upon receipt of ACK(c) from q
SBGP.19      if Counter_pp = c then
SBGP.20          EAcked_p := EAcked_p ∪ {q};
SBGP.21          if EAcked_p = External_Peers_p then
SBGP.22              Finished_p := true; EAcked_p := NULL; Flag_p := Internal;
SBGP.23          endif
SBGP.24      endif

SBGP.25  upon receipt of IP(sd, rv) from q
SBGP.26      if p = rv then
SBGP.27          if DECAP(IP) = UPDATE then
SBGP.28              if Counter_psd ≠ c then
SBGP.29                  Counter_psd := c; UPDRIB; IP := ENCAP(ACK(Counter_psd));
SBGP.30                  SEND IP(p, sd) to IGP_LOOKUP(sd);
SBGP.31              endif
SBGP.32          endif
SBGP.33          if DECAP(IP) = ACK then
SBGP.34              if Counter_pp = c then
SBGP.35                  IAcked_p := IAcked_p ∪ {sd};
SBGP.36                  if IAcked_p := Internal_Peers_p then
SBGP.37                      IAcked_p := NULL; Flag_p := External;
SBGP.38                  endif
SBGP.39              endif
SBGP.40          endif
SBGP.41      else
SBGP.42          SEND IP(sd, rv) to IGP_ LOOKUP(rv);
SBGP.43      endif
```

When an $SBGP$ router receives a user packet, it first checks if the destination belongs to the same autonomous system or it can be reached via any one of its directly connected external $SBGP$ peers. If it is true, the $SBGP$ router consults the routing table maintained by the OSPF algorithm to get the IP address of the best neighbor via which the destination can be reached. Otherwise, it checks its own forwarding table and forwards the packet to the best neighboring $SBGP$ towards the destination. The user data packet routing protocol is presented as Algorithm 4.9.

Algorithm 4.8 ($SBGP$) Border Gateway Protocol for Process p (Timeout)

```
SBGP.44  if Flag_p = Internal then
SBGP.45      c := Counter_p;
SBGP.46      for each s ∈ Internal_Peers_p do
SBGP.47          OUTMSG(s, Best_Routes_p); IP := ENCAP(UPDATE(c, RIB));
SBGP.48          SEND IP(p, s) to IGP_ LOOKUP(s);
SBGP.49      endfor
SBGP.50  endif
SBGP.51  if Flag_p = External then
SBGP.52      c := Counter_p;
SBGP.53      for each n ∈ External_Peers_p do
SBGP.54          OUTMSG(n, Routes_p);
SBGP.55          SEND UPDATE(c, RIB) to n;
SBGP.56      endfor
SBGP.57  endif
```

Algorithm 4.9 ($SBGPROUTING$) Routing User Packet Using SBGP

Messages: DATA(c : counter; sd: sender's IP address; rv: receiver's IP address, $data$: user's data)
Macros: IN_AS: checks if the IP address belongs to any subnetwork in the autonomous system
input: IP address; output: true or false
REACHABLE: checks if the IP address can be reached via a directly connected external neighbor
input: IP address; output: true or false
OSPF_BEST_NBR: gets the IP address of the best neighbor for a destination using OSPF routing table
BGP_NEXT_HOP: gets the IP address of the next hop for a destination using SBGP routing table

```
SBGPROUTING.01  upon receipt of DATA(c, sd, rv, data) from q
SBGPROUTING.02      if IN_AS(rv) or REACHABLE(rv) then
SBGPROUTING.04          SEND DATA(c, sd, rv, data) to OSPF_BEST_NBR(rv);
SBGPROUTING.05      else
SBGPROUTING.06          SEND DATA(c, sd, rv, data) to BGP_NEXT_HOP(rv);
SBGPROUTING.07      endif
```

4.3. Proof of correctness

Self-stabilization treats memory corruptions and topology changes in the same uniform way. A topology change is simply a kind of memory corruption (the state of the network is not reflected in the routers memory). A self-stabilizing algorithm upon detecting this inconsistency would trigger a message.

In our case, the self-stabilizing Update algorithm of [7] maintains the set $External\_Peers_p$ of router p, while the (self-stabilizing) underlying OSPF protocol maintains the $Internal\_Peers_p$ set of $SBGP$ Router p. Assuming all protocols run under the fair composition mechanism of [6], it follows that we can safely assume that the sets $External\_Peers_p$ and $Internal\_Peers_p$ of $SBGP$ Router p are constant and up-to-date.

We define the legitimacy predicate $\mathcal{L}_{\mathcal{SBGP}}$ for Algorithm $SBGP$ as follows:

$$\mathcal{L}_{\mathcal{SBGP}} \equiv \{\mathcal{L}_{\mathcal{OSPF}} \wedge \mathcal{L}_{\mathcal{UPD}} \wedge \mathcal{SP}_{\mathcal{IDR}}\},$$

where $\mathcal{L}_{\mathcal{OSPF}}$ denotes the set of legitimate configurations for OSPF, $\mathcal{L}_{\mathcal{UPD}}$ denotes the set of legitimate configurations for the local topology update algorithm, and $\mathcal{SP}_{\mathcal{IDR}}$ represents the set of configurations where the BGP routing tables are correct.

Lemma 1 (Correctness). *Starting from a configuration γ which satisfies $\mathcal{L}_{\mathcal{SBGP}}$, if the network topology in the autonomous systems and among the autonomous systems remains the same, then all configurations reachable from γ in any possible executions of Algorithm $SBGP$ satisfy $\mathcal{L}_{\mathcal{SBGP}}$.*

Proof. Assume that starting from a configuration γ, no network topology changes (and the routing policies do not change). This implies that $\mathcal{L}_{\mathcal{OSPF}}$ and $\mathcal{L}_{\mathcal{UPD}}$ hold and that the inputs ($External\_Peers_p$ and $Internal\_Peers_p$) contain up-to-date information. Procedure $UPDRIB$ in Algorithm $SBGP$ guarantees that if a received route to a certain destination already exists in the Adj_RIB_In, the route will be discarded. Thus, no changes will be made to Adj_RIB_In. Obviously, LOC_RIB and Adj_RIB_Out will remain the same. □

Lemma 2 (Convergence). *Starting from an arbitrary configuration, Algorithm $SBGP$ eventually reaches a legitimate configuration per Specification* 1 (*i.e.,* $\mathsf{true} \triangleright \mathcal{L}_{\mathcal{SBGP}}$).

Proof. Starting from an arbitrary state, the stabilizing property of OSPF and local topology update algorithm guarantees that eventually, the inputs ($External\_Peers_p$ and $Internal\_Peers_p$) of a $SBGP$ router p contain up-to-date information. Formally, we have:

$$\mathsf{true} \triangleright (\mathcal{L}_{\mathcal{OSPF}} \wedge \mathcal{L}_{\mathcal{UPD}}).$$

We now assume that we start from a configuration where predicate $\mathcal{L}_{\mathcal{OSPF}} \wedge \mathcal{L}_{\mathcal{UPD}}$ holds. It is obvious that in Algorithm 4.8, either the condition in Line SBGP.44 or the condition in Line SBGP.51 in all the nodes is true. Thus, the sending statements (Line SBGP.48 or Line SBGP.55) will be enabled. As a result, the receiving guard (Line SBGP.01 or Line SBGP.25 in Algorithm 4.7) becomes active. Line SBGP.16 will also be enabled so that variable $Finished_p$ will be eventually set to true. In this case, the routes reflecting topology changes can be processed (Lines SBGP.07–SBGP.12 in Algorithm 4.7). Variable $Finished_p$ is set to false and $Flag_p$ to Internal guarantees that in this configuration, no more new messages can be processed. The sending statement (Line SBGP.42) is forced to be active so that the result of the processing can be advertised to every $SBGP$ router in the autonomous system. Eventually the new routes are distributed to $SBGP$ routers in different autonomous systems, and the routing table in every router will be changed accordingly. Formally, we have:

$$(\mathcal{L}_{\mathcal{OSPF}} \wedge \mathcal{L}_{\mathcal{UPD}}) \triangleright \mathcal{L}_{\mathcal{IDR}}.$$

By transitivity of the $\triangleright$ relation, we get the lemma's result. □

Our final result follows from Lemmas 1 and 2:

Theorem 1. *Algorithm $SBGP$ is self-stabilizing.*

4.3.1. Complexity analysis
Time complexity. We assume that $IDiam$ is the maximum diameter of an autonomous system.

Proposition 1. *The Algorithm $SBGP$ requires $O(IDiam)$ time to stabilize.*

Proof. Starting from an arbitrary state, the local topology maintenance protocol (of [7]) takes $O(IDiam)$ time to stabilize. $SBGP$ also utilizes the routing information in the autonomous system. After the global topology is stabilized, OSPF takes a constant time to get the correct routing information since the shortest path computation is a local operation. When an $SBGP$ router receives a new message, it needs $(IDiam + 1)$ time to send the update to all of its internal and external neighbors. Thus the total time needed is $O(IDiam)$. □

Space complexity. The space complexity is measured here by the total number of memory bits used to implement the algorithm. We assume that NBR is the upper bound of the number of $SBGP$ routers in the system.

Proposition 2. *Algorithm $SBGP$ requires*

$$O(NBR \log NBR + NBR \log MAX + \triangle_{ep} \log \triangle_{ep} + \triangle_{ip} \log \triangle_{ip})$$

at each $SBGP$ router.

Proof. A route consists of several attributes such as AS_PATH, $Next\_Hop$, LOC_PREF, and MED. Thus a route requires $\log NBR + C$(C is a constant). $SBGP$ has three types of messages. The UPDATE message contains the counter and the RIB buffer. Assuming that the maximum size of RIB buffer is MAX_b, it requires $\log MAX\ MAX_b \log NBR$. An acknowledgment only contains a counter value. Thus, it requires $\log MAX$. An IP message contains the identifier of the sending node and the receiving node. Therefore, the message requires $2 \log NBR$ bits. We also assume that the maximum size of the message queue at every router is MAX_q.

Every node maintains its set of internal and external neighbors. Let us denote them as Ex and In, respectively. They are maintained by some local topology maintenance algorithm. The degree of p is the number of neighbors of p. The degree of p considering only its $External\_Peers$ is denoted by $\triangle_{ep}$ and is equal to $|Ex|$. The degree of p including only its $Internal\_Peers$ is denoted by $\triangle_{ip}$ and is equal to $|In|$. Every SBGP router has the following variables: $Counter_p$, $EAcked_p$, $IAcked_p$, $Finished_p$, Msg_Queue_p, $Best\_Routes_p$, $Adi\_RIB\_In_{qp}$, $Adi\_RIB\_Out_{pq}$, and LOC_RIB. $EAcked_p$ contains a set of ids of external peers and requires $\triangle_{ep} \log \triangle_{ep}$. $IAcked_p$ contains a set of ids of internal peers and uses $\triangle_{ip} \log \triangle_{ip}$. $Counter_p$ contains a node id and a counter value for all the nodes in the system. Thus, it requires

$$NBR(\log NBR + \log MAX).$$

$Best\_Routes_p$, $Adj\_RIB\_In_{qp}$, $Adj\_RIB\_Out_{pq}$, and LOC_RIB are buffers containing the individual routes. Thus, they require

$$4MAX_b \log NBR.$$

Msg_Queue_p contains three types of messages and requires

$$MAX_q(\log MAX\ +\ (\log MAX + MAX_b \log NBR)\ +\ 2 \log NBR).$$

Therefore, Algorithm $SBGP$ requires

$$O(NBR \log NBR + NBR \log MAX + \triangle_{ep} \log \triangle_{ep} + \triangle_{ip} \log \triangle_{ip})$$

at each $SBGP$ router. □

If we assume that the degree of each router and the capacity of each data link is significantly smaller than the number of autonomous systems, then the previous complexity result is $O(NBR \log NBR)$, where NBR is the number of autonomous systems. This is not surprising since each router maintains a direction (of size $\log NBR$ bits) towards each autonomous system in the network, and there are NBR of them.

5. Conclusion

In this paper, we presented a self-stabilizing BGP algorithm, Algorithm $SBGP$, which is based upon a practical (Internet) protocol. The algorithm takes $O(IDiam)$ to stabilize after the underlying local topology maintenance protocol is stabilized, where $IDiam$ is the diameter of an autonomous system that has the largest diameter. Our solution makes use of the counter flushing scheme [29] to ensure the reliable delivery of the control messages. Our dynamic data structures make sure that the invalid nodes are removed from the network quickly. This implies a fast convergence of the algorithm.

Self-stabilizing algorithms have a variety of applications. It gained a lot of attention in the past two decades due to its uniform mechanism to deal with the various types of faults. Our improved algorithms capture the underlying semantics of BGP, and at the same time, they are robust enough to deal with different transient faults. While this paper does not bring new algorithms to the theory of self-stabilization, it shows that the practical cost of self-stabilization in actually deployed protocols is in fact low and does not add much complexity to the task of designing such protocols. In our case, a simple 32 bit counter is sufficient to provide self-stabilization, while the set of errors that can be handled using this technique is significantly larger than the set of the currently implemented BGP protocol.

Since BGP-4 became the standard inter-domain routing protocol, a lot of new features, such as AS confederations, route flap damping, communities, etc, have been developed. Those extensions make the BGP a more scalable and robust routing protocol. A future research topic would be to design a self-stabilizing BGP with all the new features. Also, in our implementation of BGP, we avoided the interaction between BGP and the underlying IGP. We simply use the message encapsulation method to carry a transit packet through one autonomous system. This method might add some overhead on the performance of the algorithm. Future work can investigate the alternatives of carrying the transit traffic, such as propagation of BGP information via IGP.

References

[1] B. Awerbuch, B. Patt-Shamir and G. Varghese, Bounding the unbounded (distributed computing protocols), in: *Proceedings IEEE INFOCOM 94 The Conference on Computer Communications*, 1994, pp. 776–783.

[2] N.S. Chen, H.P. Yu and S.T. Huang, A self-stabilizing algorithm for constructing spanning trees, *Information Processing Letters* **39** (1991), 147–151.

[3] Y. Chen, A. K. Datta and S. Tixeuil, Stabilizing inter-domain routing in the Internet, in: *Proceedings of Europar 2002*, 2002, pp. 749–752.

[4] E.W. Dijkstra, Self stabilizing systems in spite of distributed control, *Communications of the Association of the Computing Machinery* **17** (1974), 643–644.

[5] S. Dolev, Self-stabilizing routing and related protocols, *Journal of Parallel and Distributed Computing* **42**(2) (1997), 122–127.

[6] S. Dolev, *Self-stabilization*, MIT Press, 2000.

[7] S. Dolev and T. Herman, Superstabilizing protocols for dynamic distributed systems, *Chicago Journal of Theoritical Computer Science* **3**(4) (1997).

[8] S. Dolev, A. Israeli and S. Moran, Self-stabilization of dynamic systems assuming only read/write atomicity, *Distributed Computing* **7** (1993), 3–16.

[9] L. Gao, T. Griffin and J. Rexford, Inherently safe backup routing with bgp, in: *IEEE INFOCOM '01*, Volume 1, 2001, pp. 547–556.

[10] M.G. Gouda, *Elements of Network Protocol Design*, Wiley, 1998.

[11] M.G. Gouda and N. Multari, Stabilizing communication protocols, *IEEE Transactions on Computers* **40** (1991), 448–458.

[12] M.G. Gouda and M. Schneider, Maximum flow routing, in: *Proceedings of the Second Workshop on Self-Stabilizing Systems*, 1995, pp. 2.1–2.13.

[13] T. Griffin, F.B. Shepherd and G. Wilfong, Policy disputes in path-vector protocols, in: *Proceedings of ACM SIGCOMM 1999*, 1999.

[14] T. Griffin and G. Wilfong, An analysis of bgp convergent properties, in: *Proceedings of ACM SIGCOMM 1999*, 1999.

[15] T. Griffin and G. Wilfong, A safe path vector protocol, in: *IEEE INFOCOM '00*, 2000, pp. 490–499.

[16] T. Griffin and G. Wilfong, A safe path vector protocol, in: *Proceedings of IEEE INFOCOM 2000, Book 2*, 2000, pp. 490–499.

[17] C. Huitema, *Routing in the Internet*, Prentice Hall, 1995.

[18] S. Katz and K.J. Perry, Self-stabilizing extensions for message-passing systems, *Distributed Computing* **7** (1993), 17–26.

[19] C. Labovitz, G.R. Malan and F. Jahanian, Internet routing instability, in: *Proceedings of ACM SIGCOMM 1997*, 1997.

[20] T. Masuzawa, A fault-tolerant and self-stabilizing protocol for the topology problem, in: *Proceedings of the Second Workshop on Self-Stabilizing Systems*, 1995, pp. 1.1–1.15.

[21] R. Perlman, *Interconnections: Bridges, Routers, Switches and Internetworking Protocols*, Addison-Wesley, Longman, 2000.

[22] L. Peterson and B.S. Davie, *Computer Networks : A Systems Approach*, Morgan Kaufmann, 2000.

[23] Y. Rekhter and P. Gross, Applications of the border gateway protocol in the Internet, Technical report, Network Working Group, 1995.

[24] Y. Rekhter and T. Li, A border gateway protocol 4(bgp-4), Technical report, Network Working Group, 1995.

[25] L.W. Stewart, *BGP4, Inter-Domain Routing in the Internet*, Addison-Wesley, 1998.

[26] A.S. Tanenbaum, *Computer Networks*, Prientice Hall, 1996.

[27] M.S. Tsai and S.T. Huang, A self-stabilizing algorithm for the shortest paths problem with a fully distributed demon, *Parallel Processing Letters* **4** (1994), 65–72.

[28] K. Varadhan, R. Govindan and D. Estrin, Persistent routing oscillations in inter-domain routing, Technical Report ISI96-631, USC/Information Science Institute, 1996.

[29] G. Varghese, Self-stabilization by counter flushing, in: *PODC94 Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, 1994, pp. 244–253.

[30] G. Varghese and M. Jayaram, The fault span of crash failures, *Journal of the ACM* **47**(2) (2000), 244–293.