Self-stabilizing Border Gateway Protocol

Yu Chen

University of Nevada, Las Vegas
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
SELF-STABILIZING BORDER GATEWAY PROTOCOL

by

Yu Chen

Bachelor of Arts
JiangXi University, NanChang, China
1992

Master of Education
University of Nevada, Las Vegas
1996

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science Degree
Department of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
December 2000
The Thesis prepared by

Yu Chen

Entitled

Self-Stabilizing Border Gateway Protocol

is approved in partial fulfillment of the requirements for the degree of

Master of Science

Ajoy Kumar Datta

Deem of the Graduate College

Examination Committee Member

Perry C. Chen

Examination Committee Member

Graduate College Faculty Representative

09/29/00
ABSTRACT

Self-Stabilizing Border Gateway Protocol

by

Yu Chen

Dr. Ajoy Kumar Datta, Examination Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

The Border Gateway Protocol (BGP) is currently the only inter-domain routing protocol employed on the Internet. It is designed to exchange the reachability information among the autonomous systems in the global Internet. The Internet routing instability (or the rapid fluctuation of the network reachability information) is an important problem facing the Internet engineering community. With the wide availability of the Internet, the Internet failures may not only interrupt the daily routines of countless end-users, but also generate millions of dollars of loss in e-commerce. Since BGP has an impact on routing in the global Internet, the design and implementation of a robust and fault-tolerant Border Gateway Protocol is an important research topic.

We achieve the fault-tolerance of BGP using the paradigm of self-stabilization. A self-stabilizing protocol, starting from an arbitrary state converges, within finite steps, to a state from where the system exhibits the desired behavior. In this thesis, we propose a self-stabilizing Border Gateway Protocol. Our design consists of mainly two phases: First, we investigate the Interior Gateway Protocols (IGP) which runs under the BGP. We design a self-stabilizing IGP. Because IGP provides the routing information inside an autonomous system, its stability is a crucial aspect of stabilization of the BGP. Then, we design a self-stabilizing BGP.
# TABLE OF CONTENTS

**ABSTRACT** ......................................................................................................................... iii

**LIST OF FIGURES** .................................................................................................................. vi

**ACKNOWLEDGMENTS** ............................................................................................................ vii

**CHAPTER 1  INTRODUCTION** .................................................................................................. 1
  Self-Stabilization ........................................................................................................... 1
  Routing Instability ...................................................................................................... 2
  Related Work ............................................................................................................... 2
  Our Contributions .......................................................................................................... 3
  Outline of the thesis ....................................................................................................... 4

**CHAPTER 2  MODEL** .............................................................................................................. 5
  Distributed System .................................................................................................... 5
  Program .............................................................................................................................. 6
  Self-Stabilization ............................................................................................................. 7

**CHAPTER 3  INTRA-DOMAIN ROUTING PROTOCOLS** ....................................................... 8
  Interior Gateway Protocol ......................................................................................... 9
    Distance Vector Routing and RIP ........................................................................... 9
    Link State Routing and OSPF ............................................................................... 10
  Specification of Intra-domain Routing Problem ................................................... 11

**CHAPTER 4  INTER-DOMAIN ROUTING PROTOCOLS** ....................................................... 12
  BGP Overview ............................................................................................................ 12
    IBGP ......................................................................................................................... 13
    EBGP ......................................................................................................................... 15
  BGP-4 ............................................................................................................................ 15
    BGP Message Types ............................................................................................ 16
    Routing Information Bases ................................................................................. 16
    BGP Route Selection ........................................................................................... 18
  Internet Routing Instability ...................................................................................... 20
  Specification of Inter-domain Routing Problem ................................................... 20

**CHAPTER 5  OSPF PROTOCOL** ............................................................................................ 22
  Global Topology Maintenance Using Counter Flushing ........................................... 22
    Specification ............................................................................................................ 22
    Reliable Flooding ................................................................................................... 23
    Algorithm $GTM$ .................................................................................................... 24
    Proofs ......................................................................................................................... 28

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Algorithm $\text{OSPF}$ ................................................................. 30
User Data Packet routing .................................................. 32
Complexity Analysis of Algorithm $\text{GTM}$ ...................... 33
Complexity Analysis of Algorithm $\text{OSPF}$ .................... 34

CHAPTER 6  BGP PROTOCOL ...................................................... 35
Algorithm $\text{BGP}$ ................................................................. 35
Data Structure ................................................................. 36
Macros and Procedures ................................................... 39
Procedures ................................................................. 40
Formal Statement ............................................................ 43
Informal Description ...................................................... 47
User Data Packet Routing .............................................. 48
Proof and Complexity Analysis ...................................... 48
Complexity Analysis ...................................................... 50

CHAPTER 7  CONCLUSIONS ..................................................... 53

BIBLIOGRAPHY ................................................................. 55

VITA ................................................................. 58
LIST OF FIGURES

4.1 IBGP and EBGP ................................................................. 13
4.2 Physical Topology ............................................................ 14
4.3 Full Mesh of IBGP Connections ....................................... 14
4.4 Loopback Interface .......................................................... 15
4.5 Network topologies of two ASs ........................................ 17
ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor Dr. Ajoy K. Datta for his direction throughout the duration of this work. His confidence in me and his belief in hard work are what motivated me to achieve this goal. Although he realized the frustration I was enduring while working on the thesis, he never opted an easy way out for me. I truly appreciate his discipline, and I am certain that I will always remember that. I also like to thank Dr. Kazem Taghva, Dr. John Minor, and Dr. John Wang for serving on my thesis committee. My special thanks to Dr. Sébastien Tixeuil (Université de Paris-Sud, France) whose great ideas helped me improve the organization and design of the algorithms.

I wish to thank all the faculty and staff of Computer Science Department, University of Nevada, Las Vegas for their sincere help and support during my graduate studies. I also would like to thank all my friends here who always supported me and cheered me up. This thesis is dedicated to my parents. I could not complete this degree without their love and encouragement.
CHAPTER 1

INTRODUCTION

1.1 Self-Stabilization

The concept of self-stabilization was first introduced by Edsger W. Dijkstra in 1974 [Dij74]. 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. In a non-self-stabilizing system, the system designer needs to enumerate the faults, such as link/node failures, that the system will face, and then must add the corresponding recovery mechanisms. They are usually independent and may cause conflicts. Also, some obscure errors like memory corruption may be difficult to enumerate. It makes sense that even if the error occurs rarely in the system, the networks should recover from those faults automatically [Var94]. In a large, distributed system, it is very hard to predict all the faults that may occur. Ideally, a system should continue its availability by correctly restoring the system state whenever the system exhibits incorrect behavior due to the occurrence of faults [AG93, Gou98]. Self-stabilizing technique provides a uniform mechanism to deal with not only arbitrary transient faults such as data, message, and location counter corruption [KP93], but also a variety of fault types like network congestion and software bugs [LAJ99]. The ability of the system to detect errors and correct itself without external intervention makes a self-stabilizing system more reliable and more powerful and more useful than a non-stabilizing system.
1.2 Routing Instability

In a brief number of years, the Internet has evolved from an academic-only network to an important component of the public telecommunication and business infrastructure. Internet backbone failures may now easily affect millions of end-users and businesses all over the world. Routing instability is one of the failures. The classic example is the crash of original ARPANET [Var94, Ros81, Per83]. The protocol was designed never to enter a state that contained three conflict updates. But, a malfunctioning router injected three such updates into the network and crashed. After this, the network cycled continuously between the three updates.

Routing instability 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 [LMJ97, VGE96]. Therefore, it is very important to have a self-stabilizing [Dij74] routing protocol which can recover from any arbitrary faults without external intervention.

1.3 Related Work

The Border Gateway Protocol (BGP) is an inter-domain routing protocol [RL95]. 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 [PD00, Ste98]. The availability and stability of the connection has a large effect on the stability of the Internet. [LMJ97, VGE96] conducted some experimental studies to investigate the Internet stability and the origins of failure in the Internet backbones, but no solution was given. [GW99b, GW99a, GSW99] presented a Simple Path Vector Protocol (SPVP) to eliminate the possible route oscillations in BGP. But, none of the above protocols [RL95, Ste98, PD00, GW99a] is self-stabilizing.

Network topology maintenance is an important component of Internet routing. A lot
of research has been done in this area [Gou98, Tan96, Hui95]. 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 [APSV94, Dol97, DH97, GS95, Mas95].

As we can see, 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 [Gou98, PDO00] routing protocol was 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. Algorithms for self-stabilizing shortest path tree construction is presented in [AKM+93, TH94]. The self-stabilizing spanning tree construction algorithms appeared in [AG94, AKM+93, DIM93, CYH91, HC92].

1.4 Our Contributions

None of the documents/papers on BGP or OSPF is self-stabilizing. In this thesis, we present the first self-stabilized Border Gateway Protocol. Since BGP's stability highly relies on the routing information in the autonomous system, we also propose a self-stabilizing Open Shortest Path First (OSPF) routing protocol, which computes the shortest path from one router to any other router in an autonomous system. The proposed protocols also dynamically allocate/deallocate storage for the routing information as the network size changes.

Once the BGP and OSPF routing tables are constructed, we present two routing protocols which deliver the user data packets: One depicts the packet traverse among the autonomous systems, and the other deals with the routers inside the autonomous system.

Both algorithms presented in this thesis require the same time $O(Idiam)$ to stabilize where $Idiam$ is the maximum diameter of an autonomous system.
1.5 Outline of the thesis

The remainder of the thesis is organized as follows: Chapter 2 discusses the model of the system used in this work, along with some important definitions. Chapter 3 gives an overview of intra-domain routing protocol, its various implementations, and the specification of the problem. Chapter 4 introduces inter-domain routing protocol, its example and the specification of the problem. Chapter 5 presents the Global Topology Maintenance algorithm, Open Shortest Path First algorithm, and user packet routing algorithm, along with the proofs of these algorithms. Chapter 6 presents the Border Gateway Protocol algorithm and user packet routing algorithm using BGP, along with the proofs of the BGP algorithm. Conclusions and some future research directions are discussed in Chapter 7.
CHAPTER 2

MODEL

2.1 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 thesis. A communication link \( (p, q) \) exists iff \( p \) and \( q \) are neighbors. Nodes communicate only by message passing [KP93]. The message delivery time is arbitrary, finite but unbounded. We assume FIFO channels.

We assume a weakly fair and distributed daemon. The weak fairness means that if a node \( p \) is continuously enabled, then \( p \) will be eventually chosen by the daemon to execute an action. The distributed daemon implies that during an execution step, if one or more processors are enabled, then the daemon chooses a nonempty subset of processors to execute an action.

The state of a process is defined by the value of its variables. The processes represent nodes or routers. The state of a system is a vector of \( n + 1 \) components where the first \( n \) represent the state of \( n \) processes, and the last component refers to the set of messages (denoted by a multi-set \( M \)) in transit in the links. In the following, we refer to the state of a process and system as a (local) state and configuration, respectively. Let a distributed protocol \( \mathcal{P} \) be a collection of binary transition relations denoted by \( \rightarrow \), 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 \( t \geq 0, \gamma_t \rightarrow \gamma_{t+1} \) (a single computation step), if \( \gamma_{t+1} \) exists, or \( \gamma_t \) is a terminal configuration. Maximality means
that the sequence is either infinite, or it is finite and no action of $P$ is enabled in the final configuration. All computations considered in this thesis are assumed to be maximal.

2.2 Program

During a computation step, one of the following actions (local steps) occurs on at least one process $p$: (1) $p$ receives a message; (2) $p$ executes some internal actions; (3) $p$ sends at least one message. The set of computations of a protocol $P$ in system $S$ starting with a particular configuration $\alpha \in C$ is denoted by $E_\alpha$. The set of all possible computations of $P$ in system $S$ is denoted as $E$.

The definition of a process consists of two parts. In the first part, messages, inputs, variables, parameters, macros and procedures of the process are declared. In the second part, actions of the process are defined [Gou98]. Messages defines the message type and message fields. Inputs can be read, but not written by the action of its process. Variables declared in a process can be read and written by the action of the process. Parameters of $p$ can be read but not written by the actions of process $p$. Macros defined in a process can be executed by the actions of the process. The details of the Macros are not given for various reasons (either they are not important to our work or they are simple). Procedures defined in a process can be invoked by the actions of the process.

Each action of a process is of the form:

```
< label > < guard > < statement >
```

The guard of an action in the program of a process $p$ is one of the following: a local guard of $p$ or a receiving guard of $p$. A local guard of $p$ is a boolean expression involving the variables of $p$. A receiving guard of $p$ is of the form:

Upon RECEIPT of $< message.type >$ from $< sending.process.name >$.

The statements of a process are of four types: assignment, sending, selecton, 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 or while ... endwhile.

The statement of an action of $p$ updates one or more variables of $p$. When $p$ executes a statement, we say that "$p$ moves" or "$p$ executes an action". An action can be executed only if its guard evaluates to true. We assume that the actions are atomically executed, meaning, the evaluation of a guard and the execution of the corresponding statement of an action, if executed, are done in one atomic step.

### 2.3 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 2.3.1 (Attractor)** Let $X$ and $Y$ be two predicates of a protocol $P$ defined on $C$ of system $S$. $Y$ is an attractor for $X$ if and only if the following condition is true:

$$\forall \alpha \vdash X : \forall e \in E_\alpha : e = (\gamma_0, \gamma_1, \ldots) :: \exists i \geq 0, \forall j \geq i, \gamma_j \vdash Y.$$  

We denote this relation as $X \triangleright Y$.

**Definition 2.3.2 (Self-stabilization)** The protocol $P$ is self-stabilizing for the specification $SP_P$ on $E$ if and only if there exists a predicate $L_P$ (called the legitimacy predicate) defined on $C$ such that the following conditions hold:

1. $\forall \alpha \vdash L_P : \forall e \in E_\alpha :: e \vdash SP_P$ (correctness).

2. $true \triangleright L_P$ (convergence).
CHAPTER 3

INTRA-DOMAIN ROUTING PROTOCOLS

Nowadays, Internet is becoming an integral part of our everyday’s life. About 20 years ago, Internet was just a simple network built around ARPANET. All routers shared routing information through one protocol, gateway-to-gateway protocol (GGP) [Hui95]. The routing table included entries for all the IP networks in the Internet. With the exponential growth of the Internet, this configuration started to cause problems. The size of the routing tables grew rapidly with the increase of the number of connected networks, while exchanging routing information became inefficient. The solution was to split the single network into a set of autonomous systems (ASs). Each AS is identified by a 16-bit “AS number”. It consists of a set of routers and networks adopting a single routing policy under the same administration. From the routing point of view, it makes perfect sense that all the routers within one AS must be interconnected and exchange routing information to maintain the connectivity. The routing protocols running among the routers within the AS are Intra-domain Routing Protocols, also called Interior Gateway Protocols (IGP). Examples of IGPs in use today are Routing Information Protocol (RIP) and Open Shortest Path First (OSPF). However, IGP routers can obtain information only about the internal networks within the AS. They must be able to discover the networks outside the AS to maintain the connectivity of the entire Internet. Border gateways, the routers (directly) connecting two adjacent autonomous systems, exchange reachability information by running Inter-domain Routing Protocol, also called Exterior Gateway Protocols (EGP) [Hui95, PD00]. Border Gateway Protocol (BGP) is an EGP protocol, and is the focus of this work.

This chapter includes an overview of IGPs, and its two major implementations: RIP
and OSPF.

3.1 Interior Gateway Protocol

An Interior Gateway Protocol (IGP) is used to distribute routing information among routers within an AS. The goal of IGP is to be able to find the best neighbor node on a shortest path from any node to all others. There are two major approaches to realize this goal: distance vector routing and link state routing.

3.1.1 Distance Vector Routing and RIP

In distance vector routing algorithms, each router maintains a vector containing the shortest distance (or the lowest cost) to all reachable networks, and the next hop on that path. Periodically, every router sends this list to its direct neighbors. The starting assumption for the distance vector routing is that each node knows the cost of the link to each of its directly connected neighbors. A down link is assigned an infinite cost. Upon receiving the vector, a router updates its local vector. One of the most widely used routing protocols based on the distance vector routing algorithm is the Routing Information Protocol (RIP).

RIP uses a very simple metric—the distance (or hop count)—to evaluate the routing information. It is the number of links a node has traversed to reach the destination. This distance has an integer value between 1 to 15. The value of 16 represents the infinity. RIP packets are carried over UDP and IP. Each router sends a packet every 30 seconds containing its personal list of distances to its directly connected neighbors. Upon receipt of the packet, a router validates the entries in the packet and makes changes to its own list of distances if necessary.

RIP is an ideal routing protocol for a small network. It has a low overhead of bandwidth and configuration, and its management is very easy. But, it also has some drawbacks:

- After a router or a link fails, the count-to-infinity problem [PD00, Tan96] may occur, and RIP takes a long time to stabilize. Some techniques, such as split horizon and triggered update, were proposed to reduce the slow convergence. But, none of them
may cure the problem [Tan96].

- RIP sets the maximum hop counter to 15. This indicates that RIP cannot be used in an autonomous system where the smallest number of hops between any two routers is 16 [Gou98]. This parameter limits the size of the network in which RIP can be used.

With the fast growing Internet, the demand for a more powerful routing protocol became urgent and necessary. Link state routing technology was developed to overcome the limitations of RIP.

### 3.1.2 Link State Routing and OSPF

In link state routing algorithm, we assume that each router knows the status (up or down) and cost of its adjacent links [PD00]. The principle behind the link state routing is simple: Instead of trying to compute the “shortest path” in a distributed fashion, every node maintains a copy of the network map that will be updated quickly after any changes in the topology. The updates are transmitted by a reliable flooding protocol. Each router performs the computation of the best route based on its local copy of the map using Dijkstra’s “shortest path first” algorithm. As all the routers have the same network map, the routes computed by them are consistent, and loops cannot occur [Hui95].

The reasons in favor of the link state routing over the distance vector routing are the following [Hui95]:

- Fast and Loop-less Convergence: The distance vector routing computes the shortest paths in a distributed fashion. The number of steps required for a system reaching the convergence is proportional to the number of nodes in the network. On the contrary, the link state routing uses a flooding protocol to transmit new information rapidly and calculates the best paths locally. Moreover, the link state routing does not have the routing looping or count-to-infinity problems.

- Multiple Metrics: It is possible for the link state routing protocol to support multiple metrics in parallel due to the local computation. Thus, the “best route” in link state
routing protocol can not only be measured by distance (hop counter), but also by link’s throughput, delay, or reliability.

- External Routes: The link state routing protocols add “gateway link state records” to the network database, which leads to a better performance due to a more precise metric and easier computations.

Open Shortest Path First (OSPF) is an example of the link state routing protocols. Its design followed the general principle of the link state routing (as presented above). There are four types of link state record contents: router links, network links, summary links (both IP networks and border routers), and external links. Among them, External Link State records support the external routes by advertising accesses to the external networks via the border routers using the external gateway protocol. Based on the LSPs from all other routers, a graph representing the network can be extracted. Local OSPF routers use the OSPF algorithm to compute the shortest path and equal paths (there might exist more than one path that have the same shortest distance) from itself to each destination. The next router towards the destination the data will be sent to can be derived from this computation [Hui95].

The goal of an IGP and an EGP are not the same. For an IGP, packets should be delivered as efficiently as possible from the source to the destination. EGPs have to take into account the politics. Since ASs belong to different owners, each owner may have different restrictions on packet delivery. Some ASs are not willing to carry transit packets originating in a particular AS, although it is on the shortest path between the sender and the destination [Tan96].

3.2 Specification of Intra-domain Routing Problem

**Specification 3.1** We consider a computation $e$ of the Intra-domain routing problem to satisfy the specification $SP_I$ when the routing tables (i) do not contain any information about the unreachable subnetworks in the autonomous system and (ii) contain the id of the next IGP router in the shortest path to a destination which belongs to a reachable subnetwork.
CHAPTER 4

INTER-DOMAIN ROUTING PROTOCOLS

In the previous chapter, we presented routing algorithms within an autonomous system. Inter-domain routing protocols are designed to exchange routes among the autonomous systems so that different ASs can share the reachability information. There have been two major exterior routing protocols in the history of the Internet: Exterior Gateway Protocol (EGP) (This EGP refers to a special protocol called EGP. This is a naming conflict with the general EGP.) and Border Gateway Protocol (BGP). EGP is the first generation inter-domain routing protocol. Although it is gradually replaced by BGP, many concepts in EGP are still in use today.

4.1 BGP Overview

Unlike many IGPs building reliability on datagram services, BGP uses TCP as its transport protocol. Operating this way provides reliable communication and hides all the details of the network being passed through. Two BGP routers establish a TCP connection between them, and exchange messages to open the connection, confirm the connection parameters, and report any route changes. Those two BGP routers are known as peers or neighbors. BGP allows each autonomous system to enforce its routing policies independently. Those policies are related to political, security, or economic consideration [RG95], and will affect the route selection and redistribution of the route information.

For BGP routers that are located in different ASs, they usually must 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 AS exchanging updates...
by running Internal BGP (IBGP) and BGP routers located in different ASs run External BGP (EBGP) to exchange the reachability information. Figure 4.1 shows an example of IBGP and EBGP.

Figure 4.1: IBGP and EBGP

4.1.1 IBGP

In Figure 4.1, Router C and Router D belong to the same autonomous system, AS300. 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 [Ste98]. 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. Figure 4.2 shows a physical topology of the BGP routers in an autonomous system, and Figure 4.3 shows the full-mesh connections among the IBGP routers [Ste98].

Loopback interfaces are often used by the IBGP peers. Loopback interfaces are not associated with a physical link or a hardware interface. They are virtual interfaces within the routers. Loopback interfaces are more flexible and reliable. By having the IBGP session run between the loopback interface, the IBGP session will stay up as long as there is a path.
between two routers. An IGP is used to route the addresses corresponding to the loopback addresses [Ste98]. Figure 4.4 gives an example of the loopback interface and IBGP session. The IBGP session is running between 139.27.128.1/30 and 139.27.128.5/30. Those addresses correspond to the loopback interfaces within Router A and Router B. Both A and B need IGP to figure out how to reach each other's interface.

It is obvious that the full-mesh connections among the IBGP routers scale very poorly. Two approaches are employed so that the IBGP connection become more scalable: router reflection and AS confederation [Ste98].
4.1.2 EBGP

Two BGP routers that are in different ASs must be 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 [RL95]. 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, such as OSPF, that run within the ASs. Therefore, when an AS carries traffics 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.

4.2 BGP-4

Since the introduction of BGP in the Internet late in 1980s, this protocol has undergone significant evolution. So far, the BGP specification has gone through several versions. The most recent version of the protocol is BGP-4. The specification of BGP-4, RFC 1771, was published in March 1995. The industrial involvement also played a great role in the development and improvement of BGP. Cisco, the world wide leader in networking for the Internet, and other router vendors worked together to make BGP a standard.
4.2.1 BGP Message Types

There are four message types in BGP: OPEN, UPDATE, NOTIFICATION, and KEEPALIVE. OPEN message is used to establish a TCP connection between two BGP routers. KEEPALIVE messages are exchanged periodically to make sure that the connection is still open. NOTIFICATION messages are sent when an error condition is detected. The BGP connection is closed immediately after sending it. UPDATE messages are used to transfer routing information between BGP peers. This message carries the actual routing information—a single feasible route to a peer, multiple withdrawn infeasible routes, or both.

4.2.2 Routing Information Bases

Routes are stored in Routing Information Bases (RIBs). An Adj_RIB_In places routes that received from other BGP peers. There are as many Adj_RIB_Inns as there are peers. An Adj_RIB_Out presents the routes that will be advertised to other BGP router to support the inter-AS multicast [RL95] (Our work does not show the support for multicasting). As with Adj_RIB_In, there is one Adj_RIB_Out for each peer. Routes that will be used by the local BGP router will be stored in LOC_RIB [Ste98].

When a BGP router receives update message from multiple ASs that describe different paths to the same destination, it must choose the best path for reaching that destination based on the values of the attributes the update message contains and the routing policies the AS enforces. Some of the attributes that BGP uses in the decision-making process [RL95] are described here. A running example is used to demonstrate the use of these attributes. Figure 4.5 shows some attributes of two ASs.

- ORIGIN: This is a mandatory attribute. It defines the origin of the path information. It can have one of the three values: IGP, EGP, or INCOMPLETE. Thus, from router A, the route for reaching 170.10.0.0 has an origin attribute of IGP if Router E received the attributed via its IGP.

- AS_PATH: This is a mandatory attribute. It is the sequence of AS numbers that an update message has traversed in order to reach a certain destination. When the
update is traversed within one AS, the value of AS_PATH remains the same. In Figure 4.5, Router E advertises a network 170.10.0.0 with an AS_PATH of 300. When router A receives the update, it appends its own AS number to it. Thus, the AS_PATH attribute for reaching network 170.10.0.0 from router A is 100, 300. If router B receives the update via IBGP session with Router A, the value of AS_PATH is still 100, 300. Loops are detected by checking if a router's own AS number is in AS_PATH received from the neighboring ASs.

- **NEXT-HOP**: This is a mandatory attribute. It is the IP address of the router that will be used as the next hop to reach the destination. In EBGP session, the value of NEXT-HOP is usually the IP address of the sending BGP router. NEXT-HOP will not be changed when the receiving BGP router distributes the route to its peers within the same AS. In Figure 4.5, Router E advertises network 170.10.0.0 to Router A with a NEXT-HOP of 170.10.20.2, and Router A advertises network 150.10.0.0 to Router E with a NEXT-HOP of 170.10.20.1. When router A redistributes 170.10.0.0 to its IBGP peer Router B, the NEXT-HOP remains as 170.10.10.1. According to router B, the next hop to reach 170.10.0.0 is 170.10.20.1 instead of 150.10.30.1. Therefore, Router B needs to learn how to reach 170.10.20.1 first via an IGP.
• **LOCAL_PREF**: This is a discretionary attribute. It is used by the BGP router to indicate the degree of preference of a route. LOCAL_PREF can only be propagated to other BGP routers within the same autonomous system. The value of LOCAL_PREF is computed locally for routes learned via EBGP session or for routes learned via an IGP. The route with the highest LOCAL_PREF is considered the best route, and will be selected. The function to calculate LOCAL_PREF can be any function that the AS administrator chooses.

• **MULTI_EXIT_DISC (MED)**: This is an optional attribute. It is used by the BGP router to distinguish among multiple entry or exit points to the same neighboring autonomous system. A lower MED value is preferred than a higher value. MED is exchanged between ASs, but the value of the MED only stays in the AS when it arrives at the AS. In other words, when an UPDATE message is received from a neighboring BGP router, the value of MED contained in the UPDATE will be used to decide which path to choose. When the receiving BGP router sends that UPDATE to another AS, the MED is set to 0 [Doc99].

4.2.3 BGP Route Selection

BGP selects only one path as the best route. When the path is selected, BGP adds it to the LOC.RIB, and propagates it to the neighboring routers.

**Metrics.** For routings between autonomous systems, the metrics assigned to each path is decided by the routing policies in each autonomous system. Since each autonomous system may run its own interior gateway protocol, and use any scheme (delay, cost, or throughput) to assign metrics to paths, it is impossible to calculate an optimal path cost for a path that crosses multiple ASs. As a result, the routes exchanged between ASs advertise only reachability, instead of an optimal path. Each route is evaluated by a BGP router using some local function, and an integer value is assigned to the route to indicate the degree of preference of the route. The route having the highest preference will be chosen.
Tie-breaking Rules for Route Selection. BGP uses some tie-breaking rules to decide which route to select if it has more than one route to the same destination with the same highest degree of preference. The details can be found in [RL95, Ste98]:

BGP Decision Process. The BGP Decision Process operates on routes contained in each Adj_RIB_In, and is responsible for the selection of routes to be advertised to BGP routers located in the local router's AS and in the neighboring ASs. The Decision Process takes place in three different phases [RL95]:

- Phase 1: Calculation of Degree of Preference.
  When the local BGP router receives an UPDATE message from a peer located in a neighboring AS, Phase 1 is invoked. For each newly received feasible route or replacement route, the local BGP shall determine the degree of preference of the route based on the policy information of the local system. The local BGP router then runs the Internal Update Process to select and advertise the preferred route.

- Phase 2: Route Selection.
  Phase 2 is invoked on the completion of Phase 1. Phase 2 process needs to consider all the routes in each Adj_RIB_In. The local BGP router then adds the selected route to the LOC_RIB, replacing any route to the same destination. The local BGP router then must determine the immediate next hop to the address presented by the NEXT_HOP attribute of selected route by performing a lookup in the IGP. Infeasible routes should be removed from LOC_RIB, and at the same time, they should be removed from Adj_RIB_Ins.

- Phase 3: Route Dissemination.
  All routes in the Loc.RIB shall be processed into a corresponding entry in the associated Adj.RIB.Out. When the update of the Adj.RIB.Outs is complete, the local BGP router shall invoke the External Update Process.

Internal Update Process. When a BGP router receives an UPDATE message from a peer located in its own AS, it should not re-advertise the routes contained in the UPDATE
message to any other BGP peers located in its own AS. When a BGP router receives an UPDATE message from a peer located in a neighboring AS, it should re-advertise the routes contained in the UPDATE message to all other BGP peers located in its own AS.

**External Update Process.** As part of Phase 3 process, all newly installed routes and all new infeasible routes for which there are no replacement routes, shall be advertised to BGP routers located in the neighboring ASs by using the UPDATE messages.

### 4.3 Internet Routing Instability

The behavior and dynamics of Internet routing stability have gone virtually without formal study, with the exception of [LMJ97, VGE96]. In 1997, [LMJ97] investigated Internet routing instability based on data collected from BGP routing messages generated by border routers at five of the Internet’s 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 increase network latency and slow convergence. At the extreme, high-level of 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. Our work of self-stabilizing BGP provides a way to detect and automatically recover from this kind of faults.

### 4.4 Specification of Inter-domain Routing Problem

Before we define the specification of the Inter-domain routing problem, we define NEXT_BR as follows:
Definition 4.4.1 (NEXT_BR) This attribute defines the IP address of the border router that should be used as the next closest neighbor to a specific destination.

Specification 4.1 We consider a computation e of the Inter-domain routing problem to satisfy the specification $S_P$ 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.
CHAPTER 5

OSPF PROTOCOL

In this chapter, we first present an outline of $\text{OSPF}$ as Algorithm 5.0.1. Next, we propose a self-stabilizing Global Topology Maintenance algorithm ($\text{GTM}$) using the counter flushing scheme [Var94]. The output of $\text{GTM}$ will be used as the input to $\text{OSPF}$. We first present the data structure used by the algorithm, and Algorithm $\text{GTM}$ is introduced in Algorithm 5.1.1 and Algorithm 5.1.2. A self-stabilizing Open Shortest Path First algorithm ($\text{OSPF}$) is introduced next, followed by the user packet routing algorithm. Finally, we present the complexity analysis for the above algorithms.

5.1 Global Topology Maintenance Using Counter Flushing

Nodes upon detecting any topology change exchange messages with other processes so that all processes in the network contain the updated topology information. Algorithm $\text{OSPF}$ uses the output produced by $\text{GTM}$. Thus, it is a very important layer for our routing protocol, and we argue that the stabilization of $\text{GTM}$ is the first step of stabilizing $\text{OSPF}$.

5.1.1 Specification

When the network topology is stabilized, every node should contain the up-to-date information of the entire network.

Specification 5.1 We consider a computation $e$ of $\text{GTM}$ to satisfy the specification, $\text{SP}_e$, when every node contains the up-to-date global topology information, and no information about the unreachable nodes is present.
### Algorithm 5.0.1 (Abstract OSPF) Abstract OSPF

1.01 if ready to broadcast a message
1.02 then
1.03 Update the list of link costs of any two active nodes using its own link cost list;
1.04 endif
1.05 if a message is received from a neighboring node \( q \)
1.06 then
1.07 if a new message
1.08 then
1.09 Set \( q \) as the parent node;
1.10 Remove any node that is not in the network from the local variables;
1.11 Update the list of link costs of any tow active nodes using the message received;
1.12 endif
1.13 if only one neighboring node
1.14 then
1.15 SEND an acknowledgment to \( q \);
1.16 else
1.17 SEND the received message to all the neighboring nodes except the parent;
1.18 endif
1.19 endif
1.20 if an acknowledgment is received from a neighboring node \( q \)
1.21 then
1.22 Record the acknowledgment;
1.23 Build the shortest path tree rooted at self;
1.24 endif

#### 5.1.2 Reliable Flooding

We used the counter flushing scheme of [Var94] to implement the reliable flooding. If a node \( p \) needs to advertise some information, it periodically sends a message with a counter value \( \text{Counter} \) to all its neighbors. Assume that \( \text{MAX} \) is the maximum number of counters that can be stored in the network and \( L_{\text{max}} \) be the maximum number of packets on each link. Then \( \text{MAX} = |E|L_{\text{max}} + |V| \) [Var94]. When a neighbor \( q \) receives the message, it can distinguish between the old and new messages by comparing its own counter value with the counter value received. If they are the same, then \( q \) realizes that it has received the same message before. It may simply discard it or send an acknowledgment back to the sender. Otherwise, \( q \) accepts the message, changes its counter value as \( \text{Counter} = c \), and processes the message. Every node \( p \) in the network can initiate a broadcast by incrementing its own counter value as \( \text{Counter} = \text{Counter} + 1 \) for its own cycle of broadcasting. With this scheme, the messages sent by one node are flooded in the network reliably.
5.1.3 Algorithm $\mathcal{GTM}$

The algorithm runs in parallel at every node. When a node $p$ wants to send some information to all other nodes in the system, it sends the message to all its directly connected neighbors, and they forward the message to their neighbors, and so on. Finally, a spanning tree rooted at $p$ is constructed, and the message reaches at every node in the network. Every node $p$ also plays the role of an intermediate node in the broadcast initiated by other nodes in the network. Therefore, every node should maintain information about different broadcast cycles initiated by all nodes including itself. A node can start a new cycle of broadcast only when its variable $Reported_p$ is true. It sends a message containing its id, a new counter value it generated, and the list of the link costs to its directly connected neighbors. When the initiating node receives acknowledgments from all its neighbors, the broadcast finishes successfully. When a node receives a message, it needs to update its variables based on the information received.

Assumption 5.1 We assume that there exists a self-stabilizing Local Topology Maintenance ($\mathcal{LTM}$) protocol running under Algorithm $\mathcal{GTM}$.

For each node $p$, $\mathcal{LTM}$ produces a dynamic link cost list, $MyCost_p$, of $p$'s directly connected neighbors. It becomes the input for Algorithm $\mathcal{GTM}$. The list is dynamic, meaning that any unreachable node would be removed from the list immediately and any new node would be added to the list. So, two nodes are directly connected, and the link between them are up iff there exists a corresponding tuple in $MyCost_p$. A node always consults its neighbor's link status before it sends a message.

The COST message has the following fields: $c$, $r$, and $Ct_r$. $c$ is the counter value in the current cycle of message passing. $r$ is the id of the node that initiates the advertisement of the message. $Ct_r$ contains the link cost of the directly connected neighbors of $r$.

The ACK message is used to enforce the reliable flooding of the COST message. After a node receives all the ACKs from its current neighboring router, it knows that the message it sent has been received by all its directly connected neighbors and their child nodes successfully. An ACK message has the following fields: $c$ and $r$. They have the same
meaning as in the COST message.

Every router $p$ maintains the following data structure, which is presented in Algorithm 5.1.1:

- $Reported_p$: Boolean variable to keep track of the status of a COST message initiated by $p$. When $Reported_p$ is true, the message is flooded to all the nodes in the network successfully, or $p$ can start a new cycle of broadcast if necessary.

- $Counter_p$: List of tuples $(id, counter)$ where $id$ is the node identifier and $counter$ is an integer in the range $0 \ldots$ MAX. It maintains the counter values in trees rooted at different nodes. $Counter_p(i, 5)$ denotes that $p$'s current counter value for the broadcast cycle in a spanning tree rooted at node $i$ is 5. If a node failed and became active again after some time, an arbitrary value would be used to start a new cycle of message passing.

- $Parent_p$: List of tuples $(id, Parent.id)$ where $id$ and $Parent.id$ are node identifiers. It maintains the pointers to the node's parent in trees rooted at different nodes. $Parent_p(i, q)$ represents that the parent of $p$ in the spanning tree rooted at node $i$ is $q$.

- $Weight_p$: List of integers with a tuple $(id, id)$ as the index. It maintains the cost of the link between any two active nodes. $Weight_p(p, q) = 5$ indicates that the link cost between $p$ and $q$ is 5.

- $Acked_p$: List of tuples $(id, Acked.ids)$ where $id$ is the node identifier and $Acked.ids$ is a set of node identifiers from which $p$ received acknowledgments. It maintains the node ids from which the ACKs have been received by node $p$ in a particular spanning tree. $Acked_p(i, \{a, b, c\})$ represents $p$ has received ACKs from nodes $a$, $b$, and $c$ for the message broadcasting initiated by root $i$.

A node $p$ can start a new cycle of broadcast only when its variable $Reported_p$ is true (Line 1.01). That is, a new cycle of broadcast can be started only when the previous cycle
Algorithm 5.1.1 (GTM) Global Topology Maintenance for Process $p$ (Part I)

Messages:
- **COST**: $c$: counter
  - $r$: the id of the node that initiates the broadcast
  - $C_{tr}$: the list of link costs of $r$'s directly connected neighbors
- **ACK**: $c$: counter
  - $r$: the id of the node that initiates the broadcast

Inputs:
- $N$: the upper bound of the number of OSPF routers in the autonomous system
- $M_{Cost}_{ip}$: a list of tuples $(id, cost)$
- $MAX$: the maximum number of distinct counter values

Variables:
- $Reported_{ip}$: boolean
- $Counter_{ip}$: a list of tuples $(id, Counter)$
- $Parent_{ip}$: a list of tuples $(id, Parent.id)$
- $Weight_{ip}$: a list, with a tuple $(id, id)$ as the index, of integers
- $Acked_{ip}$: a list of tuples $(id, Acked.ids)$

Parameters: $p, q, n, k, r$: $N$

/*@ START A NEW BROADCAST CYCLE */
1.01 if $Reported_{ip}$
1.02 then
   /* $Counter_{ip}$ indicates the counter value in tuple $(p, Counter)$ in the list $Counter_{ip}$ */
   /* maintained by $p$ */
1.03 $Counter_{ip} := (Counter_{ip} + 1) \mod MAX$;
   /* $Parent_{ip}$ indicates the id of the parent node in tuple $(p, Parent.id)$ in the list */
   /* $Parent_{ip}$ maintained by $p$ */
1.04 $Parent_{ip} := NULL$;
   /* $MyCost_{ip}$ indicates the set of nodes that are directly connected to $p$. */
1.05 for each $q \in MyCost_{ip}$ do
1.06   $Weight_{ip}(p, q) := MyCost_{ip}q$;
1.07 endfor
1.08 endif

is finished successfully. $p$ then increments its counter value and updates the variables based on the inputs (Lines 1.03 – 1.07).

When a node $p$ receives a COST message, it compares the counter value in the message with its own value. If they are different, the message received is valid, and it sets the node from which it received the message as its parent node (Lines 2.04 – 2.05). $p$ first removes any node that is not in the network from $Weight_{ip}$ and $Counter_{ip}$ based on the information received (Lines 2.06 – 2.07). For example, suppose node $i$ has three directly connected neighbors: $u$, $v$, and $w$. After some time, neighbor $u$ fails. This change is captured by $\mathcal{LTM}$, and now, $i$ only has two active neighbors left. $i$ floods this new link cost list in the network. Node $p$ receives this list. It realizes that this time node $u$ is not in the list. It clears the counter value maintained for the spanning tree rooted at $u$ (Line 2.06).
Algorithm 5.1.2 (GTM) Global Topology Maintenance for Process p (Part II)

/* RECEIVE A LINK COST LIST FROM A NEIGHBOR */
2.01 Upon RECEIPT of COST(c, r, Ct_r) from q
2.02 if Counterpr ≠ c
2.03 then /* Receive a new message for the first time */
2.04 Counterpr := c;
2.05 Parentpr := q; /* Set the parent */
2.06 Counterp := { }; /* Remove any node that is not in the network */
2.07 Weightp(r) := { }; /* Remove any node that is not in the network */
2.08 for each (k, Ct,k) € Ct_r do
2.09 if Weightp(r, k) ≠ Ct_r ∨ (k, Ct_r) ∉ Weightp
2.10 then
2.11 Weightp(r, k) := Ct_r; /* Update Weightp list */
2.12 endif
2.13 endfor
2.14 endif
2.15 if |MyCostp| = 1 /* Only has one neighbor */
2.16 then
2.17 SEND ACK(Counterpr, r) to q;
2.18 else /* Forward the message to all neighbors except the parent node. */
2.19 for each n € MyCostp ∩ MyCostp ∩ Parentpr do
2.20 SEND COST(c, r, Ct_r) to n;
2.21 endfor
2.22 endif

/* RECEIVE AN ACKNOWLEDGMENT FROM A NEIGHBOR */
2.23 Upon RECEIPT of ACK(c, r) from q
2.24 if Counterpr = c ∧ q € MyCostp ∧ p = r /* If the acknowledgment is for me. */
2.25 then
2.26 /* Ackedr represents the set of node ids in tuple (r, Acked_ids) in the list Acked_p */
2.27 Ackedpr := Ackedpr ∪ {q};
2.28 if Ackedpr = MyCostp /* Receive ACKs from all of the current active neighbors */
2.29 then
2.30 Reportedp := true; /* Convergecast Done */
2.31 Ackedpr := NULL;
2.32 endif
2.33 elseif Counterpr = c ∧ q € MyCostp ∧ p ≠ r /* If the acknowledgment is not for me. */
2.34 Ackedpr := Ackedpr ∪ {q};
2.35 if Ackedpr = MyCostp \ Parentpr
2.36 then
2.37 SEND ACK(c, r) to Parentprr;
2.38 else
2.39 endif
2.40 endfor

/* SENDS ITS COST LIST TO ITS DIRECTLY CONNECTED NEIGHBORS PERIODICALLY */
2.41 for each n € MyCostp do
2.42 c := Counterpr;
2.43 SEND COST(c, p, Ct_p) to n;
2.44 endfor

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
also removes the entry $Weight_p(i, u)$ from the list $Weight_p$ (Line 2.07). At this time, the invalid node $u$ is deleted from the data structure. $p$ then updates $Weight_p$ using the list of link costs it received (Lines 2.08 – 2.13). If $p$ only has one directly connected neighbor and this neighbor is marked as $p$'s parent node in a particular spanning tree, $p$ simply sends an acknowledgment back to its parent node. Otherwise, $p$ forwards the message it received to all its directly connected neighboring nodes except its parent node.

When a node $p$ receives an ACK message, it compares the counter value in the message with its own counter value. If they are identical, the acknowledgment is valid. When $p$ is the node that initiated a broadcast cycle receives acknowledgments from all its neighbors, it sets $Reported_p$ to true for the next broadcast cycle (Line 2.29). Otherwise, it sends an ACK message corresponding to a particular root back to the parent node it has for that root (Line 2.36).

Lines 2.40 – 2.43 indicates that node $p$ who initiates the broadcast sends its own link cost list periodically to its directly connected neighbors to ensure the reliable flooding of the message. When its neighboring nodes or the remote nodes receive the message, they simply forward the message to their directly connected neighbors except to their parent nodes (Lines 2.19 – 2.21).

Algorithm $GTM$ produces $Weight_p$, a list containing the cost of any link between two active and directly connected nodes. The list is constructed dynamically (Line 2.10). Therefore, any inactive node would not be in the list. This implies that a pair of nodes are directly connected and are active iff the tuple is in the list.

5.1.4 Proofs

Algorithm $GTM$ is implemented on top of a self-stabilizing Local Topology Maintenance Protocol ($LTM$) [APSV94, Dol97, DH97, GS95, Mas95]. $LTM$ produces a set of active neighboring nodes and their link costs which are fed as input to Algorithm $GTM$. We assume that the legitimacy predicate for the $LTM$ is denoted by $L_{LTM}$. We define the legitimacy predicates $L_g$ for $GTM$ as follows:

$L_g \equiv L_{LTM} \land SP_g$
Lemma 5.1 (Correctness) Starting from a configuration \( \gamma \) which satisfies \( \mathcal{L}_g \), if the local topology of a node remains the same, then all configurations reachable from \( \gamma \) in any possible executions of Algorithm \( \text{GTM} \) satisfy \( \mathcal{L}_g \).

**Proof.** Assume that in the configuration \( \gamma \), the local topology of a node is stabilized. This implies that each node has the correct and up-to-date local topology information. Thus, the input \( \{\text{MyCost}_p\} \) of a node \( p \) remain unchanged. Each node periodically sends a COST message with \( \text{MyCost}_p \) to keep the connection alive (Lines 2.40 – 2.43). Lines 2.08 - 2.13 in \( \text{GTM} \) indicates that same link cost value will be ignored. Thus the value of \( \text{Weight}_p \) will not be changed. \( \text{Weight}_p \) is the output of Algorithm \( \text{GTM} \). If it remains the same, the global topology of the network will remain unchanged. \( \square \)

Lemma 5.2 (Convergence) Starting from an arbitrary state, Algorithm \( \text{GTM} \) will eventually compute the correct global network topology as per Specification 5.1(i.e., true \( \triangleright \mathcal{L}_g \)).

**Proof.** Starting from an arbitrary state, the underlying \( \text{CTM} \) executes so that eventually the local topology of a node is stabilized as per Assumption 5.1. At this time, the input \( \{\text{MyCost}_p\} \) of a node \( p \) contains the up-to-date information. It is obvious now that the sending statement (Line 2.42) of all the nodes in the system can be executed and eventually, the receiving guards in the algorithm (Line 2.01) will be enabled. When the process executes the block of codes (Lines 2.06 - 2.13) corresponding to the receiving (a COST message) guard, some nodes and their related information are dynamically removed from or added to \( p \)'s local variables. Eventually, every node in the autonomous system will get the updated information and the value of \( \text{Weight}_p \) will be changed in every node. It is clear that based on the new information, every node in the system can construct a new network topology. \( \square \)

Our final result follows from Lemmas 5.1 and 5.2:

**Theorem 5.1.1** Algorithm \( \text{GTM} \) is self-stabilizing.
5.2 Algorithm OSPF

Open Shortest Path First (OSPF) is a widely employed intra-domain routing protocol. OSPF routers accumulate the link-state information by exchanging the Link State Packets (LSPs). LSPs are used to build the topology of a network with respect to the routers. Every router maintains a complete network map. Routers employ SPF by calculating the shortest path to each node with itself as the root. This calculation is done locally. Since each node maintains a complete network topology, the SPF constructed at each node will be identical. Unlike the distance-vector protocols, a router only broadcasts the cost list of its directly connected neighbors. An OSPF router uses the output of $\mathcal{GTM}$ to calculate the shortest path from itself to all other routers. Thus, $Weight_p$ is the input to Algorithm OSPF.

Every OSPF router $p$ has the following local variables:

- $Best_Nbr_p$: A list of tuples $(id, id)$. It maintains the best neighbor to a specific destination in a rooted tree. $Best_Nbr_p(i, q)$ denotes that the best neighbor to reach node $i$ from node $p$ is node $q$.

- $Cost_p$: A list of tuples $(id, Cost)$. It maintains the cost of the path from the root to all other active routers. $Cost_p(q, 15)$ indicates that the link cost from node $p$ to node $q$ is 15.

- $M_p$: A set of routers that is incorporated by the root so far.

- $MinNode_p$: A variable to store the id of the node that has the lowest cost to reach from the root.

Algorithm 5.2.1 constructs a shortest path tree at a particular node. Starting from itself, a node $p$ looks for a node $q$ that is reachable at the lowest $Cost_{pq}$ and adds it to $M_p$. Then, it updates list $Cost_p$ by considering the cost to reach the nodes through $q$. To reach a node $n$ that is directly connected to $q$, if the cost going from the source $p$ to $q$ and then following the link from $q$ to $n$ is less that the old route it had to reach $n$, then this new
route is chosen, and q is set as the Best.Nbr to reach node n. This operation is repeated until all the active nodes are incorporated.

Algorithm 5.2.1 (OSPF) Open Shortest Path First Algorithm for Process p

Inputs:  
N: the upper bound of the number of OSPF routers in the system  
Weightp: a list of integers with a tuple (id, id) as the index

Variables:  
Best.Nbrp: a list of tuples (id, id), used by BGP’s IGP_LOOKUP macro  
Costp: a list of tuples (id, Cost)  
Mp: setq: q ∈ N  
MinNodep: N

Parameters: n: N

Macros:  
NOT_IN: check a tuple to see if it is contained in a list  
input: a tuple of (id, id); output: true or false  
FIND_MIN_NODE: look for a node that is reachable from p at the lowest cost  
output: the id of the node that has the lowest cost to reach from p

/* CONSTRUCT THE SHORTEST PATH TREE ROOTED AT p */
1.01 Mp := p;  
/* Costp indicates that the link cost in tuple (p, Cost) in the list Costp maintained by p */
1.02 Costp := 0;
1.03 Best.Nbrp := p;
1.04 for each (p, n) ∈ Weightp do
1.05 Costpn := Weightp(p,n);
1.06 endfor
1.07 /* While there are nodes that have not been incorporated */
1.08 while n ∈ Costp ∧ n ∉ Mp do
1.09 MinNodep := FIND_MIN_NODE();
1.10 Mp := Mp ∪ {MinNodep}; /* Include the node to M */
1.11 for each (MinNodep, n) ∈ Weightp ∧ n ∉ Mp do /* Update the shortest distance */
1.12 if n ∈ Costp  
1.13 then
1.14 if CostpMinNodep + Weightp(MinNodep, n) ≥ Costpn
1.15 then
1.16 go to 1.10;
1.17 endif
1.18 endif
1.19 Costpn := CostpMinNodep + Weightp(MinNodep, n); /* n is one hop away from the node that is directly connected to root p. */
1.20 if (p, MinNodep) ∈ Weightp  
1.21 then
1.22 Best.Nbrp := MinNodep; /* n is more than one hop away from the node that is directly connected to root p. */
1.23 else
1.25 endif
1.26 endfor
1.27 endwhile

The shortest path tree is constructed locally at each router in Algorithm OSPF. Thus,
it is obvious that by Theorem 5.1.1, we can conclude the following:

**Theorem 5.2.1** Algorithm OSPF is self-stabilizing.

5.3 User Data Packet routing

In previous section, we presented a self-stabilizing algorithm which managed the control messages exchanged in the autonomous system and produced the complete routing information. In this section, we present a simple algorithm which manages the actual user packets exchanged in the system.

According to the OSI model [DZ83], when a process has some data it wants to send to another process, it first delivers the data to the application layer. The application layer is on the top of the transport layer, which uses two protocols: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP is a reliable connection-oriented protocol that allows a byte stream originating on one machine to be delivered without error on any other machine on the Internet. In this work, we assume that the user data packets are transmitted using TCP. Therefore, when a message is sent, it will eventually arrive at the destination. This also implies the correctness and convergence of the algorithm.

In Algorithm 5.3.1, when an OSPF router receives a user packet, it first verifies if it can reach the destination directly, or if the destination is in the same autonomous system. If it is, the router checks Best_Nbr list constructed by OSPF to find out the best neighbor it can forward the packet to towards the destination, and sends the packet to it. If the packet is designated a different autonomous system, the OSPF router first determines the best or the closest border gateway towards the destination. Then, it consults the Best_Nbr list to get the best neighboring router to which it can forward the packet to reach the border gateway. The information concerning the border gateway as the exit point of the autonomous system can be configured statically, or a BGP router can inject this information to the OSPF routing table with a flag indicating that this information is from BGP. The actual implementation is out of the scope of this work.
Algorithm 5.3.1 (OSPF ROUTING) Routing User Packet Using OSPF

Messages: Data: c: counter
          sd: the sender's IP address
          rv: the receiver's IP address
          data: user's data packet

Inputs: N: the upper bound of the number of OSPF routers in the autonomous system
        MyCost_p: the list of link cost of p's directly connected neighbors

Parameters: q: MyCost_p
            sd, rv: N

Macros: IN_AS: check if the IP address belongs to any subnetworks in the autonomous system
        input: IP address; output: true or false
EXIT_BGPROUTER: for the IP address that does not belong to any subnetworks in the
                autonomous system, returns the default BGP router
                input: IP address; output: IP address of the BGP router
BEST_NBR: return the IP address of the best neighboring router to a specific destination
           input: IP address of the destination
           output: IP address of the best neighboring router

/* RECEIVE A DATA PACKET */
1.01 Upon RECEIPT of DATA(sd, rv, data) from q
1.02 if IN_AS(rv) /* The destination is in this AS */
1.03 then
1.04 SEND DATA(sd, rv, data) to BEST_NBR(rv);
1.05 else /* The data packet is for some destination outside the AS */
1.06 SEND DATA(sd, rv, data) to BEST_NBR(EXIT_BGPROUTER(rv));
1.07 endif

5.4 Complexity Analysis of Algorithm GTM

Space Complexity. N is the upper bound of the number of OSPF routers in the autonomous system. Every node maintains its set of neighbors, denoted as MyCost. It is maintained by an underlying protocol. The degree of p is denoted by Δ_p and is equal to |MyCost|. Every node p in the system holds the following variables: Reported_p, Counter_p, Acked_p, Weight_p, and Parent_p. The id of a node requires log N, the counter value requires log MAX, and an id which belongs to MyCost_p requires log Δ_p. Reported_p contains a boolean value, which only requires 1 bit. Counter_p is a list of tuples (id, Counter) which requires N(log N+log MAX). Acked_p maintains the set of neighboring router identifiers. Thus, the memory needed is N(log NΔ_p log Δ_p). Weight_p maintains an integer value and 2 node identifier, thus the memory required is 2N log N. Parent_p maintains one id and one neighboring node id. So, the memory required is N(log N+ log Δ_p). Thus, the total

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
memory needed by OSPF is $O(N \log N + N \log \Delta_p + N \log \text{MAX})$.

**Proposition 5.1** Algorithm GTM requires $O(N \log N + N \log \Delta_p + N \log \text{MAX})$ for each router in the autonomous system.

**Time Complexity.** The time complexity can be determined by the longest delay (the time needed to send a message from one node to the farthest node in the system). We assume $IDiam$ is the diameter of the autonomous system with the largest diameter. Algorithm GTM requires $O(IDiam)$ to stabilize.

**Proposition 5.2** Algorithm GTM requires $O(IDiam)$ time to stabilize.

### 5.5 Complexity Analysis of Algorithm OSPF

**Space Complexity.** Every OSPF router holds the following variables: $\text{Best.Nbr}_p$, $\text{Cost}_p$, $M_p$, $\text{MinDist}_p$, and $\text{Next}_p$. $\text{Best.Nbr}_p$ maintains a list of tuples $(id, id)$, which requires $N(log N + \log \Delta_p)$. $\text{Cost}_p$ maintains a list of tuples $(id, \text{Cost})$, which requires $N \log N$. $M_p$ requires $n \log n$ and $\text{Next}_p$ requires $\log N$. Adding the memory needed for Algorithm GTM, the totally memory required for Algorithm OSPF is $O(N \log N + N \log \Delta_p + N \log \text{MAX})$.

**Proposition 5.3** Algorithm OSPF uses $O(N \log N + N \log \Delta_p + N \log \text{MAX})$ for each router in the autonomous system.

**Time Complexity.** The time needed for an autonomous system to stabilize is $O(IDiam)$. Since the shortest path computation is a local operation at every node, the time needed is $O(IDiam)$.

**Proposition 5.4** Algorithm OSPF requires $O(IDiam)$ time to stabilize.
CHAPTER 6

BGP PROTOCOL

In Chapter 4, we mentioned the possibility of routing instability in the Internet, and the importance of a self-stabilizing routing protocol. In this chapter, we present a self-stabilizing Algorithm BQP. Our algorithm starts in an arbitrary state without any initialization. A node in BQP periodically sends its own routing table to its neighboring peers, instead of only sending the update messages. Therefore, if a node lost all the routing information due to memory corruption, it would be able to reconstruct the table by receiving the complete routing tables from its neighboring peers. In the original algorithm, the system would crash because if only the update messages were exchanged after the initiation, a node might never regain the complete routing information. Algorithm BQP also includes SPVP proposed by [GW99b, GW99a, GSW99] to eliminate the possibility of the routing oscillation.

We organize this section as follows. We first present an outline of Algorithm BQP. Then we introduce the data structure of the algorithm, followed by the actual algorithm. We also present some procedures used by the algorithm. A simple algorithm for user packet routing is also proposed. Finally, we provide the proof for Algorithm BQP and its complexity analysis.

6.1 Algorithm BQP

We first present an abstract version of the algorithm (Algorithm 6.1.1) to help get a quick overview of the algorithm. Before we formally introduce BQP, we explain inputs, local variables, macros, and procedures used in the algorithm. The pseudo-code of BQP is introduced in Section 6.3. Procedures used by BQP are presented in details in Section 6.2.
Algorithm 6.1.1 (BGP)  Abstract BGP

1.01 if an UPDATE message is received from neighbor q
1.02 then
1.03 if the my counter value and the received counter value are different
1.04 then
1.05 Save the message in the waiting queue;
1.06 Send an acknowledgment back to q;
1.07 if the previous broadcasting cycle is done
1.08 then
1.09 Update Adj.rib. In.ep;
1.10 Decision Process;
1.11 endif
1.12 else /* the counter values are the same */
1.13 Send an acknowledgment back to q;
1.14 endif
1.15 endif
1.16 if an acknowledgment is received from an external peer q
1.17 then
1.18 Record the acknowledgment;
1.19 endif
1.20 if an IP packet is received from an internal peer q
1.21 then
1.22 if I am the destination
1.23 then
1.24 Decapsulate the IP packet;
1.25 if the message contained in the IP packet is an UPDATE message
1.26 then
1.27 if the message is new
1.28 then
1.29 Update Adj.rib. In.ep;
1.30 SEND encapsulated acknowledgment to q;
1.31 endif
1.32 endif
1.33 if the message contained in the IP packet is an acknowledgment
1.34 then
1.35 Record the acknowledgment;
1.36 endif
1.37 else /* I am not the receiver */
1.38 Forward the IP packet to the best neighbor towards the destination using
1.39 OSPF routing scheme;
1.40 endif
1.41 endif

6.1.1 Data Structure

A BGP router p may have two different types of neighbors: internal and external. An internal neighbor q may or may not be directly connected to p, but is in a same autonomous system as p. We call it an Internal_Peer of p. An external neighbor q is directly connected to p, but resides in the different autonomous system as p, and we call it External_Peer of p.
Whenever a BGP router wants to send a message to its internal peers or external peers, it checks its *Internal_Peers* or *External_Peers* set to make sure that the corresponding peer router is active.

We assume that Algorithm $GTM$ is running under BGP. As mentioned before, by Assumption 5.1, $CTM$ produces a set of routers that are directly connect to a BGP router, and we call it set $L$. We assume that BGP routers can identify their neighboring routers as BGP routers or OSPF routers. Therefore, a BGP router can filter out all the OSPF routers from the set $L$. We call this new set of BGP routers $E$. $E$ is then used as the input to Algorithm $GTM$. The output of above computation is the network topology containing only BGP routers and OSPF routers in the autonomous system, and we call this set of routers $G$. It is obvious that a BGP router $p$ can compute its *External_Peers* as $L \setminus (L \cap G)$. A BGP router can get its *Internal_Peers* set as $E \setminus \text{External_Peers}$.

Within each autonomous system, $OSPF$ provides the shortest path routing. It runs in every BGP router in parallel with Algorithm $BGP$ so that all internal BGP peers know how to reach each other. There are three kinds of messages exchanged among the BGP routers: the UPDATE message, the ACK message, and the encapsulated IP packet. A BGP router may receive different types of messages at the same time. The reason we use the IP packet is the following. BGP peers inside one AS may not be directly connected with each other. In order to prevent the looping routing announcement within the AS, when a BGP router receives a route from another BGP peer located in the same AS, it cannot re-advertise the route to other BGP peers in the same AS. Thus, a full-mesh connection is needed among all BGP routers in the same AS. A BGP-level message may traverse among non-BGP routers before it reaches the final destination. The encapsulation hides the details of the actual message, and to the non-BGP routers, they are just like ordinary IP packets.

For simplicity, we argue that the UPDATE or ACK message can be used to exchange information among the external peers, and the encapsulated IP packet can be used to exchange information among the internal peers.

The UPDATE message contains the following fields: $c$ and $RIB$. $c$ contains the value of the current cycle of broadcasting. $RIB$ consists of the routes being broadcast. The
UPDATE message piggybacks the RIB to reduce the traffic in the system.

The ACK message contains one field: \( c \), the value of the current cycle of broadcasting initiated by the sender.

The IP packet is an encapsulated message. It has two fields: \( sd \) and \( rv \). \( sd \) is the id of the sender, and \( rv \) denotes the id of the destination. The BGP router encapsulates the UPDATE or the ACK message into ordinary IP packet so that non-BGP routers will be able to understand it. Since the final destination of a message or a packet is always a BGP router, the (intermediate) non-BGP routers do not have to interpret the message at all. They only need to forward the IP packet to their best neighbor towards the specific destination.

Algorithm BGP is built on top of several other algorithms. Thus, it has a complex data structure. Every BGP router \( p \) has the following local variables:

- \( \text{Counter}_p \): List of tuples \((id, Counter)\). \( id \) is the node identifier and \( Counter \) is an integer in the range \( 0...MAX \). It keeps track of the counter values used in different message-sending cycle initiated by a specific node. \( \text{Counter}_p(i, 5) \) denotes that the current message-sending cycle initiated by node \( i \) has the counter value 5.

- \( \text{EAcked}_p \): Set of external BGP routers from which an acknowledgment was received at node \( p \) in the cycle of broadcasting started by \( p \).

- \( \text{IAcked}_p \): Set of internal BGP routers from which an acknowledgment was received at node \( p \) in the cycle of broadcasting started by \( p \).

- \( \text{Finished}_p \): Boolean variable. \( \text{Finished}_p = \text{true} \) indicates that the cycle of broadcasting started by \( p \) is finished successfully, and a new cycle of broadcasting at \( p \) can start if necessary.

- \( \text{MsgQueue}_p \): Queue to store the new UPDATE messages from the external BGP peers. The queue is implemented in a FIFO fashion to ensure the fairness.

- \( \text{BestRoutes}_p \): 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*$_{pq}$: Set of routes received from peer $q$.

• *Adj.RIB.Out*$_{pq}$: Set of routes needed to be broadcast by to peer $q$.

• *LOC.RIB*$_p$: Set of routes selected by the local BGP router's Decision Process.

• $P$: Route in RIBs.

• $P.NLRI$: IP address of the destination.

6.1.2 Macros and Procedures

Every BGP router $p$ uses the following macros locally. The details of the macros are not presented because they are simple or they are not critical to our work.

• ENCAP: This macro encapsulates a BGP level UPDATE message or an ACK message into an ordinary IP packet. The BGP level UPDATE message or ACK message is its input, and it returns the IP packet.

• DECAP: This macro decapsulates an IP packet. It takes an IP packet as its input and returns a BGP level UPDATE message or an ACK.

• IGP.LOOKUP: This macro returns the best neighboring router in the autonomous system to reach 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.

In addition to the macros, every BGP router also uses some procedures:

• UPDRIB: Updates the Routing Information Base using the routing information contained in the received UPDATE message from both internal or external peers.
- **INTERNAL\_UPDATE**: Selects the best route towards a distinct destination using all the routes received from both internal and external peers.

- **LOCRIB\_UPDATE**: Uses the routes chosen in **INTERNAL\_UPDATE** to update the LOC\_RIB buffer, and also, installs the routes that need to be advertised in the corresponding *Adj\_RIBs\_Out*.

- **OUTMSG**: Formats *RIBs* in the outgoing message.

- **SPV**: Checks if a particular route contains a cycle due to the change of the routing policies. The output of the procedure is a new route without any cycle to a certain destination.

### 6.2 Procedures

Before we present the pseudo-code of Algorithm $BG\bar{P}$, we introduce some procedures used by the algorithm.

**INTERNAL\_UPDATE.** This procedure (Algorithm 6.2.1) will be invoked only when a new route is received from an external peer. After the *Adj\_RIB\_In* buffer is updated, 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 BGP routers inside the autonomous system. If several routes to a certain destination have the same value of *LOC\_PREF*, some tie-breaking rules mentioned in Section 4.2.3 will be used.

**UPDRIB.** This procedure (Algorithm 6.2.2) updates the *Adj\_RIB\_In* buffer accordingly. When a BGP 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 (Line 1.04). $p$ also updates the attributes of the accepted routes (Lines 1.07 – 1.08). If the route update message is from an internal peer, $p$ only updates the *Adj\_RIB\_In* if necessary without changing the attributes of the routes.
Algorithm 6.2.1 (INTERNAL UPDATE) Advertise Selected Routes Within the AS

Variables: \( b_{\text{r}} \): temporary buffer to store the selected route

Procedures: SPV: detect the routing oscillations due to the routing policy conflicts

Macros: BEST: select the best route for a certain destination from all the routes received from the external peers
CAL.PREF: assign the degree of preference based on the routing policies

1.01 \( \text{Adj}.RIB_{\text{In}}.P.\text{LOC}.PREF := \text{CAL}.PREF(\text{Adj}.RIB_{\text{In}}.P.\text{NLRI}); \)
/* For each distinct destination, the preferred route has the highest degree of preference, */
/* or is selected as the result of applying tie-breaking rules. */
1.02 \( b_{\text{r}} := \text{BEST}(); \)
1.03 \( b_{\text{r}} := \text{SPV}(b_{\text{r}}); \)

Algorithm 6.2.2 (UPDRIB) Update Adj.RIB.In Buffer

Variables: \( P.\text{NEXT}.BR \): IP address of the best neighboring route to the destination
\( P.\text{AS}.\text{PATH} \): a sequence of integers to represent the ASs the route has traversed through

Macros: APPEND : append own AS number to the attribute \( \text{AS}.\text{PATH} \)

1.01 if \( q \in \text{External}.\text{Peers}_p \)
1.02 then
1.03 for each \( M sg.RIB_{\text{In}}.P.\text{NLRI} \) do
/* For a certain destination, update the buffer only when the route is not in the buffer */
/* or the route in the buffer is different from the one in the message. */
1.04 if \( (M sg.RIB_{\text{In}}.P \neq \text{Adj}.RIB_{\text{In}}.P) \lor (M sg.RIB_{\text{In}}.P.\text{NLRI} \notin \text{Adj}.RIB_{\text{In}}.P) \)
1.05 then
1.06 \( \text{Adj}.RIB_{\text{In}}.P := M sg.RIB_{\text{In}}.P; \)
1.07 \( \text{Adj}.RIB_{\text{In}}.P.\text{NEXT}.BR := p; \)
1.08 \( \text{Adj}.RIB_{\text{In}}.P.\text{AS}.\text{PATH} := \text{APPEND}(\text{MyAS}_p, \text{Adj}.RIB_{\text{In}}.P.\text{AS}.\text{PATH}); \)
1.09 endif
1.10 endfor
1.11 else if \( q \in \text{Internal}.\text{Peers}_p \)
1.12 then
1.13 for each \( M sg.RIB_{\text{In}}.P.\text{NLRI} \) do
1.14 if \( (M sg.RIB_{\text{In}}.P \neq \text{Adj}.RIB_{\text{In}}.P) \lor (M sg.RIB_{\text{In}}.P.\text{NLRI} \notin \text{Adj}.RIB_{\text{In}}.P) \)
1.15 then
1.16 \( \text{Adj}.RIB_{\text{In}}.P := M sg.RIB_{\text{In}}.P; \)
1.17 endif
1.18 endfor
1.19 endif

LOCRIB.UPDATE. This procedure (Algorithm 6.2.3) takes all the routes that are presented in the \( \text{Adj}.RIBs.In \) including those received from BGP routers in the autonomous systems and those received from the neighboring autonomous system, and selects the best route for each distinct destination. These selected routes then are installed in \( \text{LOC}.RIB \), replacing any route to the same destination that is currently used.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Algorithm 6.2.3 (LOCrib update) Update LOC.RIB Buffer

Variables: $Route_p$: temporary storage for a route

Macros: SELECTION: select 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

1.01 for each $P.NLRI$ do /* For each route with a distinct destination */
1.02 $Route_p := \text{SELECTION}();$
1.03 /* Detect the cycle. */
1.04 if $Route_p \notin \text{LOC.RIB}_p$
1.05 then
1.06 /* Add the selected route to LOC.RIB buffer. */
1.07 $\text{LOC.RIB}_p := \text{LOC.RIB}_p \cup \{Route_p\};$
1.08 endif
1.09 /* Add the selected route to the corresponding Adj.RIB.OUT buffer. */
1.10 $\text{Adj.RIB.Out}_{pq} := \text{Adj.RIB.Out}_{pq} \cup \{Route_p\}$;
1.11 endfor

OUTMSG. This procedure (Algorithm 6.2.4) formats the outgoing messages. When a message is sent to the BGP peers inside the autonomous system, the routes can be simply copied into the messages without any alternation. When a message is sent to the BGP peers outside the autonomous system, some of the attributes of the routes, such as $\text{LOC.PREF}$ and $\text{MED}$, need to be changed.

Algorithm 6.2.4 (OUTMSG) Format Outgoing Messages

Inputs: $\text{RIB}$: set of routes that need to be advertised

Variables: $P.\text{LOC.PREF}$: integer, the degree of the preference of the route
$P.\text{MED}$: integer, the preference of an entry/exit point of an AS

1.01 if $q \in \text{External.Peers}_p$ /* The receiving node is $p$'s external peer */
1.02 then
1.03 for each $RIB.P$ do
1.04 $\text{Msg.c} := \text{Counter}_p$;
1.05 $\text{Msg.RIB}_{pq}.P.\text{LOC.PREF} := dlp$;
1.06 $\text{Msg.RIB}_{pq}.P.\text{MED} := 0$;
1.07 endfor
1.08 else if $q \in \text{Internal.Peers}_p$ /* the receiving node is $p$'s internal peer */
1.09 for each $RIB.P$ do
1.10 $\text{Msg.c} := \text{Counter}_p$;
1.11 $\text{Msg.RIB}_{pq}.P := RIB.P$;
1.12 endfor
1.13 endif

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
This procedure (Algorithm 6.2.5) provides a scheme to identify the route oscillations caused by the routing policy conflicts. According to [GW99a, GW99b, GSW99], the conflicts due to the routing policies might introduce extreme oscillations into the global routing system. A Simple Path Vector Protocol (SPVP) was developed to capture and suppress those routes that contain cycles due to the routing policy conflicts. Details of this protocol can be found in [GW99a, GW99b, GSW99].

Algorithm 6.2.5 (SPV)  Simple Path Vector

Inputs:  
\( b_r : \) the route that is chosen as the best route to a certain destination

Variables:  
\( Old_r : \) the previous route to a certain destination  
\( New_r : \) the new route that is chosen as the best route to a certain destination  
\( History_{New_r} : \) 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: compute a new path history  
CYCLE: detect whether a certain history list contains a cycle or not  
BEST: compute the best route to a certain destination

1.01  if \( br \in LOC\_RIB \land br \neq LOC\_RIB_p.P \)  
1.02  then
1.03    \( Old_r := LOC\_RIB_p.P; \)
1.04    \( New_r := br; \)
1.05    \( History_{New_r} := HIST(LOC\_RIB_p.P); \)
1.06    if CYCLE(History_{New_r}) = true /* A cycle is detected. */  
1.07    then
1.08      \( Bad\_Path := Bad\_Path \cup \{ New_r \}; \)
1.09      \( New_r := BEST(); \)
1.10      if \( New_r \neq Old_r \)  
1.11      then
1.12        \( History_{New_r} := (\cdot, Old_r); \)
1.13      endif
1.14 endif
1.15  endif
1.16  if \( New_r \neq Old_r \)  
1.17      \( Route_p := (New_r, History_{New_r}); \)
1.18      \( LOC\_RIB_p := LOC\_RIB_p \setminus \{ Old_r \}; \)
1.19      endif
1.20 endif

6.3 Formal Statement

In this section, we present the pseudo-code of the algorithm. Data structure, macros, and procedures used in the algorithm are listed in Algorithm 6.3.1. The pseudo-code of
BGP is presented in Algorithm 6.3.2 and Algorithm 6.3.3.

**Algorithm 6.3.1 (BGP)  Border Gateway Protocol for Process p (Part I)**

**Messages:**
- UPDATE: c: counter
  - \( RIB_{q} \): the routing information base from neighboring router \( q \)
- ACK: c: counter
- IP: \( sd \): the id of the sender
  - \( ru \): the id of the destination

**Inputs:**
- MAX: the upper-bound of the data link capacity
- \( External\_Peers_{p} \): set \( \{ n: \) a neighbor of \( p \), directly connected to \( p \), but not in the same autonomous system as \( p \} \)
- \( Internal\_Peers_{p} \): set \( \{ s: \) a sibling of \( p \), may not be directly connected to \( p \), but in the same autonomous system as \( p \} \)
- \( NBR \): the upper bound of the number of routers
- \( MyAs \): the number of the autonomous system

**Variables:**
- \( Adj\_RIB\_In_{q} \): set of routes received from peer \( q \)
- \( Adj\_RIB\_Out_{p} \): set of routes advertised to \( q \) by \( p \)
- \( LOC\_RIB_{p} \): set of routes selected by the local BGP router’s Decision Process
- \( P \): a route in RIBs
- \( P.NLRI \): IP address of the destination
- \( E\_Acked_{p} \): set of external BGP routers an acknowledgment has been received
- \( I\_Acked_{p} \): set of internal BGP routers an acknowledgment has been received
- \( Counter_{p} \): a list of tuple \( (id, Counter) \)
- \( Finished_{p} \): boolean
- \( Best\_Routes_{p} \): set of routes chosen as the preferred routes for every destination
- \( Routes\_Selected_{p} \): set of preferred routes;
- \( Msg\_Queue_{p} \): a queue maintaining the unprocessed messages from the external peers

**Parameters:**
- \( p, q, k, sd, ru: NBR \)
- \( s: Internal\_Peers_{p} \)
- \( n: External\_Peers_{p} \)

**Procedures:**
- UPDRIB: update the routing information base
- INTERNAL\_UPDATE: update RIB in local AS
- LOCRI\_UPDATE: update LOC\_RIB buffer
- OUTMSG: format RIBs in the outgoing messages
- SPV: eliminate the possible routing oscillations

**Macros:**
- ENCAP: encapsulate an UPDATE or an ACK message into an IP packet
- DECAP: decapsulate an IP packet into an UPDATE or an ACK message
- IGP\_LOOKUP: return the best neighboring router from OSPF’s Best\_Nbr to reach a specific destination
- ENQUEUE: add a new UPDATE message to the tail of the message queue
- DEQUEUE: remove an UPDATE message from the head of the message queue
Algorithm 6.3.2 (BGP)  Border Gateway Protocol for Process p (Part II)

/* RECEIVE AN UPDATE MESSAGE FROM AN EXTERNAL PEER */
/* Piggybacks the RIB_{pq} in the UPDATE message. */
1.01 Upon RECEIPT of UPDATE(c, RIB_{qp}) from q
  /* The new message is from an external BGP peer. */
  if q ∈ External_Peers_p ∧ Counter_{pq} ≠ c
    then
      ENQUEUE(UPDATE(c, RIB_{qp})); /* Save the new message in the waiting queue. */
      SEND ACK(Counter_{pq}) to q;
      if Finished_p = true /* The previous broadcast cycle is done. */
        then
          DEQUEUE(); /* Assume that this message is from external peer k */
          Counter_{k} := c;
          UPDRIB(k);

        /* Decision Process Phase 1 */
        for each Adj.RIB.In_{k,p}.P.NLRI do /* Check external peers' RIBs. */
          Best_Routes_p := INTERNAL.UPDATE() ∪ Best_Routes_p;
        endfor

        /* Decision Process Phase 2 */
        for all Adj.RIB.In_{k,p} do /* For both internal and external peers' RIBs. */
          Routes_Selected := LOCRIB.UPDATE() ∪ Routes_Selected;
        endfor

        /* Decision Process Phase 3 */
        Finished_p := false;
        Flag_{p} := Internal;
        Counter_{p} := (Counter_{p} + 1) mod MAX;
        endif
      else /* The same message was received before. Discard. */
        SEND ACK(Counter_{pq}) to q;
      endif

      /* RECEIVE AN ACKNOWLEDGMENT FROM AN EXTERNAL PEER */
      Upon RECEIPT of ACK(c) from q
      if Counter_{p} = c /* The acknowledgment is arrived at the destination. */
        then
          EAcked_{p} := EAcked_{p} ∪ {q}; /* Received the ACKs from all the current active external peers. */
          if EAcked_{p} = External_Peers_p
            then
              Finished_{p} := true;
              EAcked_{p} := NULL;
              Flag_{p} := Internal;
            endif
        endif
      endif
/* RECEIVE AN IP PACKET FROM EITHER AN INTERNAL PEER */
/* OR A NON-BGP ROUTER */

Upon RECEIPT of IP(sd, ru) from q

if p = ru /* The packet arrived at the destination */
then
  /* Decapsulate the IP packet to get the original UPDATE or ACK message. */
  if DECAP(IP) = UPDATE /* It is an UPDATE message. */
  then
    if Counterp = c /* New message */
    then
      Counterp := c;
      UPDRIB;
      IP := ENCAP(ACK(Counterp, sd));
      SEND IP(p, sd) to IGP_LOOKUP(sd);
    end if
  end if

end if

if DECAP(IP) = ACK /* It is an acknowledgment. */
then
  if Counterp = c then
    IAcknowledged := IAcknowledged U {sd};
    /* Received ACKs from all the current active internal peers */
    if IAcknowledged := Internal Peers
    then
      IAcknowledged := NULL;
      Flagp := External;
    end if
  end if
else /* The packet is in transit. */
  SEND IP(sd, ru) to IGP_LOOKUP(ru);
end endif

/* PERIODICALLY SEND AN UPDATE MESSAGE CONTAINING THE ROUTING TABLE */

if Flagp = Internal
then
  c := Counterp;
  for each s ∈ Internal_Peersp do
    OUTMSG(s, Best_Routesp);
    IP := ENCAP(UPDATE(c, RIB));
    SEND IP(p, s) to IGP_LOOKUP(s);
  endfor
end if

if Flagp = External
then
  c := Counterp;
  for each n ∈ External_Peersp do
    OUTMSG(n, Routesp);
    SEND UPDATE(c, RIB) to n;
  endfor
end if
6.4 Informal Description

In the following, we refer to Algorithm 6.3.2 and 6.3.3. A BGP router will receive either an UPDATE message from its peers located in different ASs, or an IP packet from its peers located in the same AS. When a router $p$ receives an UPDATE message from a router $q$, it realizes that $q$ is a BGP peer located in a different AS. It compares the counter value $c$ in the UPDATE message with its own counter value $\text{Counter}_p q$ corresponding the broadcast cycle started by $q$. If they are the same, $p$ simply sends an acknowledgment back to $q$ (Lines 1.26-1.27). If not, $p$ saves the message in the message waiting queue first. If the value of the variable $\text{Finished}_p$ is true (Line 1.05), the previous broadcasting cycle initiated by $p$ is finished and a new broadcasting cycle can start. Thus, $p$ 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 $\text{Counter}_p k = c$, sends an acknowledgment to the sender of the UPDATE message and updates the $\text{Adj.RIB.In}_{kp}$ based on the RIB contained in the UPDATE message (Lines 1.08 - 1.10). $p$ then invokes the Decision Process for the route selection. In Phase 1 (Lines 1.11 - 1.12), $p$ 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 BGP peers in the same AS. In Phase 2 (Lines 1.14 - 1.15), $p$ selects the best route out of all the routes presented in the $\text{Adj.RIBs.In}$, including the routes received from BGP peers located in its own AS and in the neighboring ASs, and installs them into $\text{LOC.RIB}_p$. In Phase 3, the new installed routes in $\text{LOC.RIB}_p$ are processed into corresponding entries in the associated $\text{Adj.RIBs.Out}$, and advertised to each peer located in a neighboring AS. $p$ sets the variable $\text{Flag}_p$ to Internal so that the new changes will be sent to all its internal peers first (Line 1.17).

When a router $p$ receives an acknowledgment from its external peer, 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, and checks if it has received acknowledgments from all its current active external peers. If so, $p$ sets $\text{Finished}_p$ to true so that a new cycle of broadcast can start, and set $\text{Flag}_p$ to Internal (Lines 1.25 - 1.30).
When a node \( p \) receives an IP packet from node \( q \), it recognizes that the packet is from either a BGP peer or a non-BGP 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 1.59 – 1.60). If \( p \) itself is the actual receiver of the packet, \( p \) can interpret the packet by decapsulating it. The message containing in the IP packet may be an UPDATE message or an acknowledgment. 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 1.39 – 1.44). If it is an acknowledgment message, \( p \) records the actual sender and checks to see if it has received all the acknowledgments (Lines 1.49 – 1.55). Variable \( Flag_p \) is set to External (Line 1.55) so that \( p \) can broadcast the new changes to its external peers.

6.5 User Data Packet Routing

In this section, we present an algorithm for routing the users' data packet. As mentioned in Section 5.3, the user packets are transmitted using TCP, a reliable transport protocol. Therefore, we can safely assume that a message will eventually arrive at its destination.

When a BGP 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 BGP peer. If it is true, the BGP router consults the routing table maintained by Algorithm \( OSPF \) to get the IP address of the best neighbor from which the destination can be reached. Otherwise, it checks its own forwarding table and forwards the packet to the BGP next hop towards the destination.

6.6 Proof and Complexity Analysis

The sets \( External.Peer_p \) and \( Internal.Peer_p \) of BGP Router \( p \) are computed by Algorithm \( GTM \). By Theorem 5.1.1, we conclude that those two sets will eventually
Algorithm 6.5.1 (BGP ROUTING) Routing User Packet Using BGP

Messages: Data: \( c \): counter
- \( sd \): the sender's IP address
- \( rv \): the receiver's IP address
- \( data \): user's data packet

Parameters: \( p, q \): \( NBR \cup N \)

Macros:
- \( \text{IN\_AS}: \text{check if the IP address belongs to any subnetworks in the autonomous system} \)
  input: IP address; output: true or false
- \( \text{REACHABLE}: \text{check if the IP address can be reached via any one of the directly} \)
  connected external neighbors
  input: IP address; output: true or false
- \( \text{OSPF\_BEST\_NBR}: \text{get the IP address of the best neighbor for a certain destination} \)
  using OSPF routing table
- \( \text{BGP\_NEXT\_HOP}: \text{get the IP address of the next hop for a certain destination} \)
  using BGP routing table

/* RECEIVE A DATA PACKET */

1.01 Upon RECEIPT of DATA(\( c, sd, rv, data \)) from \( q \)
1.02 if \( \text{IN\_AS}(rv) \) or \( \text{REACHABLE}(rv) \)
   /* The destination is in this AS or it can be reached via */
   /* one of the directly connected external neighbors */
1.03 then
1.04 SEND DATA(\( c, sd, rv, data \)) to OSPF\_BEST\_NBR(\( rv \));
1.05 else /* The destination is in another AS. */
1.06 SEND DATA(\( c, sd, rv, data \)) to BGP\_NEXT\_HOP(\( rv \));
1.07 endif

contain the up-to-date information. Also, in order to carry the transit traffic from one AS to another, BGP must rely on the routing information computed by Algorithm OSPF. The correctness of the OSPF routing information has a direct impact on the BGP routing. By Theorem 5.2.1, we can assume that the routing information within the autonomous system is always correct. We define the legitimacy predicate \( L_B \) for Algorithm BGP as follows:

\[
L_B \equiv L_{TM} \land L_{0} \land SPO
\]

Lemma 6.1 (Correctness) Starting from a configuration \( \gamma \) which satisfies \( L_B \), if the network topology in the autonomous systems and among the autonomous systems remain the same, then all configurations reachable from \( \gamma \) in any possible executions of Algorithm BGP satisfy \( L_B \).

Proof. Assume that starting from a configuration \( \gamma \), no network topology changes (and the routing policies do not change). This implies that \( L_{TM} \) is stabilized and the
inputs (External_Peers\textsubscript{p} and Internal_Peers\textsubscript{p}) contain up-to-date information. Procedure \textit{UPDRIB} in Algorithm \textit{BGP} guarantees that if a received route to a certain destination already exists in the \textit{Adj.RIB.In}, the route will be discarded. Thus, no changes will be made to \textit{Adj.RIB.In}. Obviously, \textit{LOC.RIB} and \textit{Adj.RIB.Out} will remain the same. □

**Lemma 6.2 (Convergence)** Starting from an arbitrary state, Algorithm \textit{BGP} will eventually compute the correct routing information as per Specification 4.1(i.e.,true > L\textsubscript{B}).

**Proof.** Starting from an arbitrary state, Algorithm \textit{GTM} executes so that eventually the network topology is stabilized as per Theorem 5.1.1. At this time, the inputs (External_Peers\textsubscript{p} and Internal_Peers\textsubscript{p}) of a BGP router \textsubscript{p} contain the up-to-date information. By Theorem 5.2.1, we can conclude that the OSPF routing tables will also be stabilized eventually.

It is obvious that in Algorithm 6.3.3, either the condition in Line 1.62 or the condition in Line 1.69 in all the nodes is true. Thus, the sending statements (Line 1.67 or Line 1.73) will be enabled. As a result, the receiving guard (Line 1.01 in Algorithm 6.3.2 or Line 1.33 in Algorithm 6.3.3) becomes active. Line 1.22 in Algorithm 6.3.2 will also be enabled so that variable \textit{Finished}\textsubscript{p} will be set to true eventually. In this case, routes reflecting topology changes can be processed (Lines 1.10 - 1.15). Variable \textit{Finished}\textsubscript{p} is set to false and \textit{Flag}\textsubscript{p} to Internal guarantees that in this configuration, no more new messages can be processed. The sending statement (Line 1.67 in Algorithm 6.3.3) is forced to be active so that the result of the processing can be advertised to every BGP router in the autonomous system. It is clear that eventually the new routes will be distributed to BGP routers in different ASs, and the routing table in every router will be changed accordingly. □

Our final result follows from Lemmas 6.1 and 6.2:

**Theorem 6.6.1** Algorithm \textit{BGP} is self-stabilizing.

6.6.1 Complexity Analysis

**Time Complexity.** We assume that \textit{IDiam} is the maximum diameter of an autonomous system. Starting from an arbitrary state, the Local Topology Maintenace Protocol takes
$O(IDiam)$ time to stabilize. BGP 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 a BGP 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 $O(IDiam)$.

**Proposition 6.1** The Algorithm BGP requires $O(IDiam)$ time to stabilize.

**Space Complexity.** The space complexity is measured by the total number of memory bits used to implement the algorithm. We assume that $NBR$ is the upper bound of the number of BGP routers in the system. 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). BGP 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 id of the sending node and the receiving node. Therefore, the message requires $2\log NBR$. 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, denoted as $Ex$ and $In$ respectively. They are maintained by Algorithm $GTM$. The degree of $p$ is the number of neighbors of $p$. The degree of $p$ of its $External\_peers$ is denoted by $\Delta_{ep}$ and is equal to $|Ex|$. The degree of $p$ of its $Internal\_peers$ is denoted by $\Delta_{ip}$ and is equal to $|In|$. Each BGP router has the following variables: $Counter_p$, $EAcked_p$, $IAcked_p$, $Finished_p$, $Msg\_Queue_p$, $Best\_Routes_p$, $Adj\_RIB\_In_{qp}$, $Adj\_RIB\_Out_{pq}$, and $LOC\_RIB$. $EAcked_p$ contains a set of ids of external peers and requires $\Delta_{ep} \log \Delta_{ep}$. $IAcked_p$ contains a set of ids of internal peers and uses $\Delta_{ip} \log \Delta_{ip}$. $Counter_p$ 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 $4MAX_b \log N$. $Msg\_Queue_p$ contains three types of messages and requires $MAX_q(\log MAX + (\log MAX + MAX_b \log NBR) + 2 \log NBR)$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Therefore, Algorithm $BGP$ requires $O(N_{BR} \log N_{BR} + N \log MAX + \Delta_{ep} \log \Delta_{ep} + \Delta_{ip} \log \Delta_{ip})$ at each BGP router.

**Proposition 6.2** Algorithm $BGP$ requires $O(N_{BR} \log N_{BR} + N \log MAX + \Delta_{ep} \log \Delta_{ep} + \Delta_{ip} \log \Delta_{ip})$ at each BGP router.
CHAPTER 7

CONCLUSIONS

In this thesis, we presented two self-stabilizing algorithms, $\textit{GTM}$ and $\textit{BGP}$. Both algorithms are based upon some practical (Internet) protocols. 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. $\textit{BGP}$ uses the outputs of $\textit{GTM}$ as inputs to guarantee the correctness. Its stability also relies on the stability of $\textit{OSPF}$ routing information. $\textit{BGP}$ takes $O(IDiam)$ to stabilize after $\textit{GTM}$ and $\textit{OSPF}$ are stabilized.

The idea behind both algorithms is to use the counter flushing scheme [Var94] to ensure the reliable delivery of the control messages. Also, both algorithms use dynamic data structures so that the invalid nodes would be removed from the network immediately. 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 $\textit{BGP}$ and $\textit{GTM}$, and at the same time, are robust enough to deal with different transient faults. In order to respect the original construction of $\textit{OSPF}$ protocol, Algorithm $\textit{OSPF}$ constructs a shortest path tree locally at each node after the global topology is stabilized. It is worthwhile to use a distributed algorithm to construct the shortest path instead. After all, the real life protocols are running in a distributed fashion. BGP is a very young routing protocol, and it is changing for the better everyday. 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.
BIBLIOGRAPHY


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.


VITA

Graduate College
University of Nevada, Las Vegas

Yu Chen

Local Address:
1600 E. University Ave. #226
Las Vegas, NV 89119

Home Address:
# 402, No. 30, 120 Lane, KangJian Road
ShangHai, 200233, China

Degrees:
Bachelor of Arts, English Literature, 1992
JiangXi University, NanChang, China

Master of Education, Curriculum and Instruction, 1996
University of Nevada, Las Vegas

Thesis Title: Self-stabilizing Border Gateway Protocol

Thesis Examination Committee:
Chairperson, Dr. Ajoy Kumar Datta, Ph.D.
Committee Member, Dr. John Minor, Ph.D.
Committee Member, Dr. Kazem Taghva, Ph.D.
Graduate Faculty Representative, Dr. John Wang, Ph.D.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.