1-1-2013

Improving BGP Convergence And Reachability Through Stable Path Aggregation (SPAGG)

Amro A. Sabbagh
Ryerson University

IMPROVING BGP CONVERGENCE AND REACHABILITY THROUGH STABLE PATH AGGREGATION (SPAGG)

By

Amro Azzam Sabbagh

B.Sc in Computer and Information Science
Prince Sultan University, Riyadh, Saudi Arabia, 2007

A thesis
presented to Ryerson University
in partial fulfillment of the
requirements for the degree of
Masters of Applied Science
in the Program of
Computer Networks

Toronto, Ontario, Canada, 2013

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

IMPROVING BGP CONVERGENCE AND REACHABILITY THROUGH STABLE PATH AGGREGATION (SPAGG)

Master of Applied Science, 2013, Amro Sabbagh, Computer Networks, Ryerson University

ABSTRACT

BGP is the standard inter-domain routing protocol of the internet. It has proven to be scalable enough to accommodate the exceptional growth of the Internet. However, because of the sheer size of the Internet and the complexity of its topology, the behaviour of BGP can be unpredictable sometimes. Researchers have been proposing various changes and enhancements in the past 10 to 15 years to improve the security, stability and convergence of BGP. Some of the solutions have been adopted, but BGP is still suffering from possible deficiencies when it comes to convergence time and stability at specific situations and scenarios. In this thesis, we focus on providing a reasonable solution for the problem of BGP instability but without causing long convergence, which leads eventually into minimizing BGP churn and path exploration. We, first, analyse the current BGP standard protocol and previous proposed solutions. Then, we study current problems associated with a recently proposed improvement, suggest a new algorithm that avoids path selection problem at the aggregator and the path shortening problem. We also describe its implementation in OPNET. Finally, we show the results from our simulation and compare them to the results of previous work suggested. Our results show a great improvement of the convergence of BGP while preserving reachability and optimality all the time.

Acknowledgement

I thank Allah Almighty the most graceful and most merciful for all his blessings which enabled me to complete my research.

I thank my supervisor Dr. Muhammad Jaseemuddin who invested much appreciated time and effort in my research. His guidance and support were essential to the completion of this thesis. I also thank Dr. Khalid Hafeez who provided me with priceless recommendations and comments during my work. Thanks are extended to Dr. Nkog-Wa Mah and Arseny Taranenko who were very supportive during my studies at Ryerson.

I deeply thank my wife Mais for her understanding, compassion, support, and love during the hard times we have been through during my studies. None of this could have been possible without her. She took care of me during the best and worst times and encouraged me to stand up again each time I fell. I thank my son Faisal who was born during my first year at Ryerson. He is the reason in this world that makes me want to achieve more and succeed.

I also appreciate all the emotional support from my dad Azzam and mom Neda for their prayers and wishes. They gave me certainty when I was skeptical and were always there for me when I needed them. Finally, I thank Alalem family for standing by my side on every step of the way.

Table of Contents

ABSTRACT iii

Acknowledgement iv

Table of Contents v

Chapter One: Introduction 1

1.1 Introduction 1

1.2 Thesis Overview 5

Chapter Two: Background and Literature Review 6

2.1 BGP Overview 6

2.2 BGP Route Selection Process 8

2.3 Route Flap Damping 9

2.4 Minimum Route Advertisement Interval 10

2.5 Path Exploration Damping 11

2.6 Churn Aggregation Approach 11

Chapter Three: Test Network Design and Protocol Analysis 15

3.1 OPNET Modeler Overview 15

3.1.1 BGP Implementation in OPNET Modeler 17

3.2 Simulation Network Design 19

3.3 Standard BGP Simulation with No Flaps or Path Exploration (Simulation 1) 20

3.3.1 AS 7 Convergence statistic (Simulation 1) 20

3.3.2 AS 9 Convergence statistics (Simulation 1) 23

3.3.3 Application Statistics (Simulation 1) 24

3.4 Standard BGP Simulation with Flaps and Path Exploration (Simulation 2) 26

3.4.1 AS 7 Convergence statistic (Simulation 2) 27

3.4.2 AS 9 Convergence statistics (Simulation 2) 29
3.4.3 Application Statistics (Simulation 2) 30
Chapter Four: Problem Description and Simulation 34
4.1 AS Path Selection Problem 34
4.1.1 AS 7 Convergence statistic (Simulation 3) 35
4.1.2 AS 9 Convergence statistic (Simulation 3) 38
4.1.3 Application Statistics (Simulation 3) 39
4.2 AS Path Shortening Problem 42
4.2.1 AS Path Shortening Problem – Scenario 1 44
4.2.2 AS Path Shortening Problem – Scenario 2 45
Chapter Five: Stable Path Aggregation (SPAGG) 48
5.1 Stable Path Aggregation (SPAGG) Algorithm 49
5.1.1 Stable Path Selection 49
5.1.2 AS path prepending 53
5.2 SPAGG Implementation 56
5.3 SPAGG Results 63
5.3.1 Results of Simulation 5 63
5.3.2 Results of Simulation 6 67
Chapter Six: Conclusion and Future Work 70
References 72
Appendix A: Complete Simulation Results 76
A.1 Simulation 1 Results 76
A.1.1 FTP 76
A.1.2 HTTP 77
A.1.3 Links Utilization 78

A.2 Simulation 2 Results 81
A.2.1 FTP 81
A.2.2 HTTP 83
A.2.3 Links Utilization 84
A.3 Simulation 3 Results 87
A.3.1 FTP 87
A.3.2 HTTP 88
A.3.3 Link Utilizations 89
A.4 Simulation 5 Results 92
A.4.1 FTP 92
A.4.2 HTTP 93
A.4.3 Link Utilizations 95
Appendix B: C Code Implementation 97
B.1 Modified bgp_conn_update_message_handle()code 97
B.2 bgp_update_message_State_enter_exicutives code and Modified Selection Code 109

Table of Figures

Figure 1 Comparison between BGP and aBGP [9] convergence activity 2

Figure 2 Comparison between aBGP [9] traffic delay and SPAGG traffic delay 3

Figure 3 BGP network with a flapping link 12

Figure 4 CAGG Algorithm for constructing aggregated AS path in [9] 13

Figure 5 Example of a process with multiple forced and unforced states 16

Figure 6 Interrupt Operations in OPNET 17

Figure 7 bgp process model 18

Figure 8 bgp_conn process model 18

Figure 9 Test Network Topology 19

Figure 10 AS7 Convergence Activity (Simulation 1) 21

Figure 11 Routing Convergence Report (Simulation 1) 21

Figure 12 BGP Traffic AS7 (Simulation 1) 22

Figure 13 AS9 Convergence Activity (Simulation 1) 23

Figure 14 BGP Traffic AS9 (Simulation 1) 23

Figure 15 Email Download/Upload Response Time (Simulation 1) 24

Figure 16 Email Traffic Sent/Received (Simulation 1) 25

Figure 17 Ping Replies Received and Response Time (Simulation 1) 25

Figure 18 IP Packet Drops (Simulation 1) 26

Figure 19 AS7 Convergence Activity (Simulation 2) 28

Figure 20 BGP Traffic AS7 (Simulation 2) 28

Figure 21 AS9 Convergence Activity (Simulation 2) 29

Figure 22 BGP Traffic AS9 (Simulation 2) 29

Figure 23 Email Download/Upload Response Time (Simulation 2) 30
Figure 24 Email Traffic Sent/Received (Simulation 2) 31
Figure 25 Ping Replies Received and Response Time (Simulation 2) 32
Figure 26 IP Packet Drops (Simulation 2) 33
Figure 27 Test Topology from [9] 34
Figure 28 AS7 Convergence Activity (Simulation 3) 36
Figure 29 BGP Traffic AS7 (Simulation 3) 36
Figure 30 Network Convergence Duration from Simulation 2 above Compared to Simulation 3 37
Figure 31 AS9 Convergence Activity (Simulation 3) 38
Figure 32 BGP Traffic AS9 (Simulation 3) 39
Figure 33 Email Download/Upload Response Time (Simulation 3) 40
Figure 34 Email Traffic Sent/Received (Simulation 3) 40
Figure 35 Ping Replies Received and Response Time (Simulation 3) 41
Figure 36 IP Packet Drops (Simulation 3) 42
Figure 37 Modified Test Network for Simulation 4 43
Figure 38 AS9-AS7 Link Utilization - AS Path Shortening Scenario 1 (Simulation 4) 44
Figure 39 AS9-AS10 Link Utilization - AS Path Shortening Scenario 1 (Simulation 4) 45
Figure 40 AS9-AS10 Link Utilization - AS Path Shortening Scenario 2 (Simulation 4) 46
Figure 41 AS9-AS7 Link Utilization - AS Path Shortening Scenario 2 (Simulation 4) 46
Figure 42 SPAGG - Stable Path Selection Algorithm 50
Figure 43 SPAGG - Stable Path Selection Workflow 51
Figure 44 SPAGG - AS Path Prepending Algorithm 53
Figure 45 Stable Path Aggregation (SPAGG) Workflow 54
Figure 46 New BGP Prefix Structure 57
Figure 47 New BGP Route Structure 57

Figure 48 Modified bgp_support_rte_entry_copy function 58
Figure 49 bgp_process Established State Exit Executives 59
Figure 50 Modified bgp_conn_update_message_handle() function code workflow 60
Figure 51 bgp_update_message_State_enter_exicutives code workflow 61
Figure 52 bgp_support_route_select_modified() Code Workflow 62
Figure 53 Email Download/Upload Response Time (Simulation 5) 64
Figure 54 Email Traffic Sent/Received (Simulation 5) 64
Figure 55 Ping Replies Received and Response Time (Simulation 5) 65
Figure 56 IP Packet Drops (Simulation 5) 66
Figure 57 Modified Test Network for Simulation 6 67
Figure 58 AS7-AS9 Utilization for Simulation 6 68
Figure 59 AS9-AS10 Utilization for Simulation 6 68
Figure 60 AS9-AS10 Utiliztion for Simulation 6 69
Figure 61 FTP Download/Upload Response Time (Simulation 1) 76
Figure 62 FTP Traffic Sent/Received (Simulation 1) 77
Figure 63 HTTP Object Response/Page Response Time (Simulation 1) 77
Figure 64 HTTP Traffic Sent/Received (Simulation 1) 78
Figure 65 Link AS7-AS9 Utilization (Simulation 1) 79
Figure 66 Link AS7-AS4 Utilization (Simulation 1) 79
Figure 67 Link AS7-AS8 Utilization (Simulation 1) 80
Figure 68 FTP Download/Upload Response Time (Simulation 2) 81
Figure 69 FTP Traffic Sent/Received (Simulation 2) 82
Figure 70 HTTP Object Response/Page Response Time (Simulation 2) 83
Figure 71 HTTP Traffic Sent/Received (Simulation 2) 83
Figure 72 Link AS7-AS9 Utilization (Simulation 2) 84

Figure 73 Link AS7-AS4 Utilization (Simulation 2) 85
Figure 74 Link AS7-AS8 Utilization (Simulation 2) 86
Figure 75 FTP Download/Upload Response Time (Simulation 3) 87
Figure 76 FTP Traffic Sent/Received (Simulation 3) 88
Figure 77 HTTP Object Response/Page Response Time (Simulation 3) 88
Figure 78 HTTP Traffic Sent/Received (Simulation 3) 89
Figure 79 Link AS7-AS9 Utilization (Simulation 3) 90
Figure 80 Link AS7-AS4 Utilization (Simulation 3) 90
Figure 81 Link AS7-AS8 Utilization (Simulation 3) 91
Figure 82 FTP Download/Upload Response Time (Simulation 5) 92
Figure 83 FTP Traffic Sent/Received (Simulation 5) 93
Figure 84 HTTP Object Response/Page Response Time (Simulation 5) 93
Figure 85 HTTP Traffic Sent/Received (Simulation 5) 94
Figure 86 Link AS7-AS9 Utilization (Simulation 5) 95
Figure 87 Link AS7-AS4 Utilization (Simulation 5) 96
Figure 88 Link AS7-AS8 Utilization (Simulation 5) 96

List of Abbreviation

AS	Autonomous Systems
BGP	Border Gateway Protocol
EBGP	External Border Gateway Protocol
FSM	Finite State Machine
IBGP	Internal Border Gateway Protocol
ICMP	Internet Control Message Protocol
IP	Internet Protocol
ISP	Internet Service Provider
MED	Multi Exit Discriminator
MRAI	Minimum Route Advertisement Interval
NLRI	Network Layer Reachability Information
RIB	Routing Information Base
TCP	Transport Control Protocol
IGP	Interior Gateway Protocol
PA	Path Attribute
RFD	Route Flap Damping
PED	Path Exploration Dampening
MRAI	Minimum Route Advertisement Interval
CAGG	Churn Aggregation
aBGP	Aggregation Boarder Gateway Protocol
RFC	Request For Comments
STD	State Transition Diagram

Chapter One

Introduction

1.1 Introduction

It is very important to understand the significance that Border Gateway Protocol (BGP) has in modern day communication systems [1, 2, 3]. It is the only routing protocol that is used to achieve the global connectivity of the Internet; as a result, many systems depend on the stability and robustness of this protocol [1]. Furthermore, BGP was designed and built to achieve the highest level of stability and scalability. BGP uses path-vector protocol where it exchanges topology information in order for each router to select the best path to the destination prefix. Unlike other Interior Gateway Protocols (IGPs), BGP doesn't only use a single metric to select the best path. It depends on a collection of Path Attributes (PAs) to determine the best route to install in the BGP table and routing table. We describe the process of path selection in details later in the thesis. This structure gives a high level of flexibility and control to ISPs in order to control traffic paths under local policies, which lead to a stable and predictable network. Unfortunately, this is not always the case. With the expansion of the Internet and the existence of incremental numbers of users and traffic streams, many problems surfaced and affected the operations of Autonomous Systems.

One of the problems faced by routing protocols is link flapping and its side-effects. In BGP, when a flapping link exists in the network, it has a negative effect on the stability of the protocol that might lead to lengthening convergence times. This was observed early in the life of BGP and one solution was standardized and adopted by vendors, which is Route Flap Damping (RFD) [4]. However, because of the mechanism of this solution, it was no longer recommended to implement RFD on the internet [5]. RFD can actually lead to a loss of reachability for an unacceptable time even long after a flap has occurred in the network. Another technique was Minimum Route Advertisement Interval (MRAI) [6,7] which delays consequent updates received within a predetermined interval. This approach lowers the number of updates sent to other neighbors, but it extends the convergence. The default MRAI configuration is not recommended any more [6,7].

In 2010, a new solution similar to MRAI called Path Exploration Damping (PED) was proposed [8]. The PED delays update messages which would announce a route with a same-length or longer AS Path than the previously announced route for the same prefix. This method was successful to lower the number of updates but sometimes prolonged the total convergence duration [9]. Furthermore, the PED didn't take into consideration the history of the prefix weather it has been stable or not.

In 2011, Wang et al [9] focused on the AS-PATH Attribute through aggregating AS-Path attributes for highly active prefixes. This method called Churn Aggregation (CAGG) focuses on the observation that a small number of AS PATHs are explored by each highly active prefix. In [9], CAGG proved to give better results than RFD, MRAI, and PED. Simply, CAGG functions in a very similar way to RFD, where each prefix has a penalty and a threshold. When the threshold is reached, CAGG stops sending advertisements and suppress later advertisements as long as the penalty is over a configured threshold. Furthermore, CAGG avoids reachability problems associated with RFD by sending a single advertisement with an aggregated AS path that contains most of the AS paths usually used to reach the destination. Also, CAGG doesn't suppress withdrawals in order to achieve a quick convergence. During our research, we simulate the normal behavior of BGP during instability and compare it to the behavior of aggregation BGP (aBGP) [9]. Figure 1 shows how aBGP [9] is successful to cut the convergence activity to only the first 800 seconds at one of the routers. Each convergence activity is represented by a 1 in the graph.

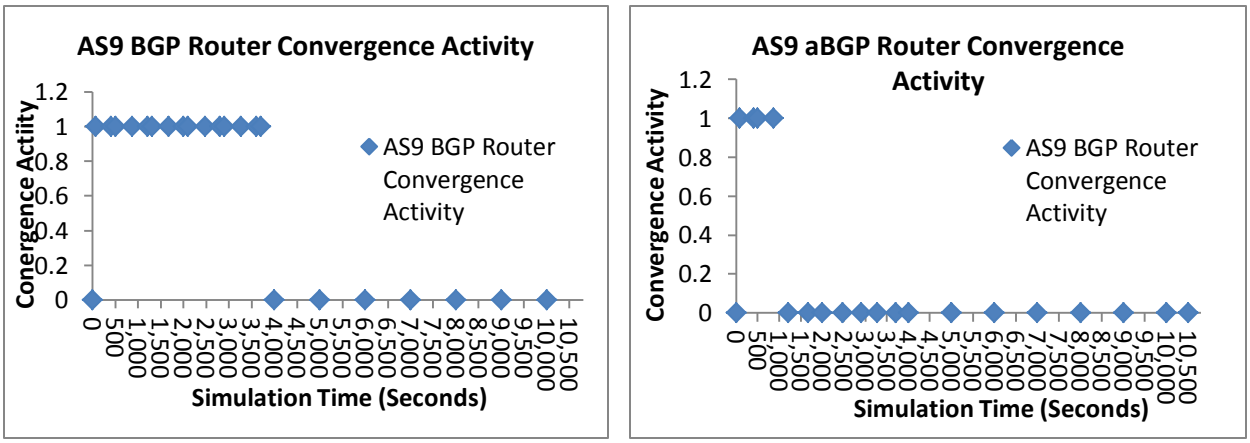


Figure 1 Comparison between BGP and aBGP [9] convergence activity

In addition, CAGG is successful to reduce the number of BGP updates by over 50% of the original churn. However, there are two main problems currently existing with the original CAGG [9] algorithm that represent the core of our research. The first problem is the path selection problem at the aggregator, where we study the behavior of data traffic to understand the effect of BGP instability. The aggregator is the

router that notices the instability in the network and creates the new advertisement with an aggregated AS path. This process results in a significant reduction of BGP updates and convergence time. However, Wang et al in [9, 26] didn't specify the behavior of the aggregator when it receives traffic. We set up our simulation to mimic the operations of aBGP [9] and inject multiple traffic streams into the network. Applications like Email, FTP, HTTP, and ICMP are running constantly throughout the simulation. Although we find that Churn Aggregation is successful in improving BGP stability, but we don't see any improvement in applications' delay time or traffic drops. We try to fix this problem by introducing our algorithm Stable Path Aggregation (SPAGG), which contains a new selection algorithm that is triggered immediately after the process of path aggregation. Although there might be some similarities with the work of [9] in 2011, however our work differs noticeably due to the changes incorporated into the original algorithm. For example, based on some of the ideas in [10], we are adding the functionality of stable path selection (SPS) to the CAGG enabled router which is triggered whenever AS-PATH aggregation is activated. Stable Path Selection aims to select most stable route among all available routes with implicit consideration to penalty. Original SRS try to maximize route stability while minimizing route deviation from best path. Figure 2 shows the improvement in Email download delay running SPAGG compared to the original Churn Aggregation algorithm.

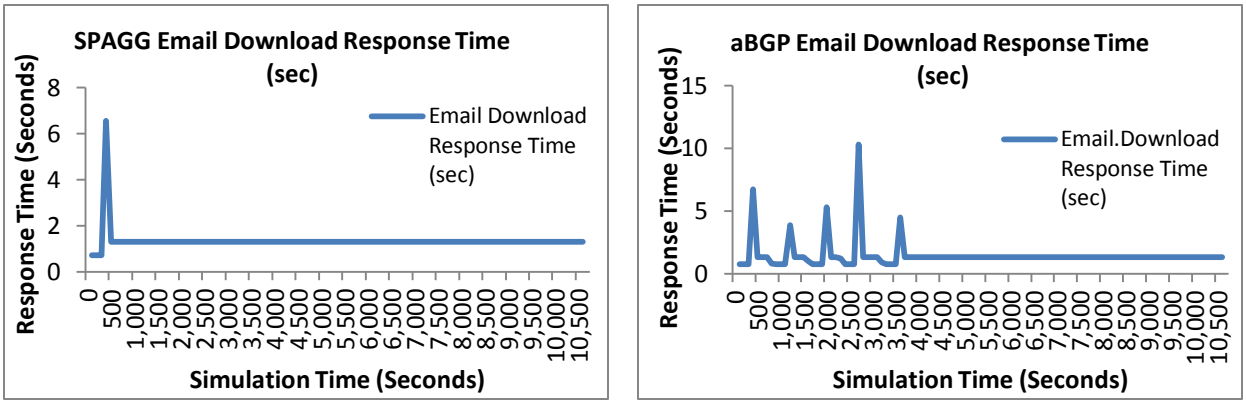


Figure 2 Comparison between aBGP [9] traffic delay and SPAGG traffic delay

The second problem under study in our research is the AS path shortening problem. The process of AS path aggregation results in a shorter path than any of the original paths used to calculate it. Hence, this new aggregated AS path is usually more preferable during best path selection. During our research, we simulate different scenarios where traffic is forced to use the aggregated AS path advertised by the aggregator. The result is a shift in traffic towards an instable path and an increase in traffic delay and IP drops. Our algorithm, SPAGG, aims to solve this problem by lengthening the newly created AS path. SPAGG keeps track of the length of the best path that triggered the algorithm. It compares the length of

the best path with the length of the new aggregated AS path. If the new path is shorter, then it is prepended to make it of equal length. After that, the new aggregated path is advertised. This process protects the integrity of BGP decision process at upstream routers. With SPAGG, traffic is no longer shifted due to the shortening of AS paths.

OPNET Modeler [11] is used as a platform of simulation during the course of our work. In order to understand the implementation and simulation of scenario, it is important to understand how OPNET simulates and implements the original BGP protocol. We deliver an analysis of the BGP dynamics in OPNET to make it easier for the reader to grasp our concepts. Changes to the original code are implemented using C language and Proto C which is used by OPNET itself. We also go through STD diagram to visualize the processes under concern. Debugging of processes was achieved through linking the OPNET process to Microsoft Visual Studio 2003. The network topology used is similar to the topology used in [9] which represents a common structure in research papers. However, we made some changes to the topology to complicate the scenario in certain cases. Also, each simulation is run multiple times with different parameters each time. For example, we run multiple simulations with different link failure times. Final results introduced in this document are the averages of various runs.

Throughout this document, multiple simulation results are introduced. We gradually start by running a flawless network with zero faults or flaps. This establishes the best case scenario where results are used as reference. Then, we study another simulation where link flaps are incorporated to see the effect on BGP convergence, stability duration, and applications behavior. After that, we run another simulation with an altered version of BGP to achieve the behavior of aBGP invented in [9]. Finally, we run a simulation that represents our algorithm, SPAGG, and inspect results related to data traffic behavior. We show that SPAGG achieves the same convergence results as CAGG, but produces lower data traffic delay and better data throughput. Also, SPAGG conserves traffic paths of upstream routers by lengthening the produced aggregated path.

1.2 Thesis Overview

This thesis is organized as follows. In Chapter 2, we first provide a background of BGP and then summarize the related work in this area to point out the difference from our work. This chapter includes various areas of concern based on methodologies used: (a) Current BGP standard as described in RFCs including BGP attributes and selection process (b) A study of RFD implementation and problems (c) A study of MRAI implementation and problems (d) Previous work created based on PED and a study of problems associated (e) Description of work in [9] and CAGG behavior.

In chapter 3, we introduce our test network design and analysis through simulating multiple BGP scenarios. The analysis is structured as follows: (a) OPNET Modeler Overview and BGP implementation (b) Our test network design (c) BGP simulation showing convergence behavior in OPNET with no faults or problems (d) BGP simulation and analysis of behavior when link flaps and path exploration exist in the network.

In chapter 4, we define the problem produced by aBGP and present in-depth analysis of aBGP reaction to link flaps and path exploration. We show simulation results of aBGP and highlight problems like traffic delay and drop induced by CAGG. Furthermore, we show another simulation to prove how CAGG represents an attraction point of traffic in an instable network.

Chapter 5 presents our proposed algorithm to the problems described in chapter 4 and implementation of the solution. This chapter describes algorithm used and code design. Also, it shows the end results of our solution simulation showing enhancements to BGP convergence activity, convergence duration, traffic delay, and link utilization. Finally, chapter 6 summarizes our work and provides information about future work expected.

Chapter Two

Background and Literature Review

2.1 BGP Overview

Since its beginning in 1969, the Internet has grown from few host computer systems to tens of millions. Every computer connected to the Internet is part of this vast network. A very important building block of the Internet Architecture is Autonomous systems (ASes). An Autonomous System (AS) is a connected segment of a network topology that consists of a collection of subnetworks (with hosts attached) interconnected by a set of routers [13, 14]. The Border Gateway Protocol (BGP) handles the task of establishing routes between the Autonomous Systems (ASes) that make up the Internet. It enables ASes to make local decisions based on private routing policies, and forms global routes from these local decisions. At a very high level, the execution of BGP requires ASes to continuously make independent route selections, based on their local preferences over routes, and announce these choices to all neighboring ASes [13, 14, 15].

BGP exchanges topology information in order for routers to eventually learn the best routes to a set of IP prefixes. Unlike IGPs, BGP does not use a metric to select the best route among alternate routes to the same destination. Instead, BGP uses several BGP path attributes (PAs) and an involved decision process when choosing between multiple possible routes to the same subnet [16]. BGP neighbors form a TCP connection with each neighbor, sending BGP messages over the connections. Each router explicitly configures its neighbors' IP addresses, using these definitions to tell a router with which IP addresses to attempt a TCP connection. Also, if a router receives a TCP connection request (to BGP port 179) from a source IP address that is not configured as a BGP neighbor, the router rejects the request. After the TCP connection is established, BGP begins with BGP Open messages. Once a pair of BGP Open messages has been exchanged, the neighbors have reached the established state, which is the stable state of two working BGP peers. At this point, BGP Update messages can be exchanged [16].

There are two types of BGP connections: eBGP and iBGP. eBGP connections represent a relationship between two routers in different ASes. On the other hand, iBGP connections represent relationship between two routers in the same AS. The two types of neighbors differ only slightly in regard to forming neighbor relationships, with more significant differences in how the type of neighbor (iBGP or eBGP)

impacts the BGP update process and the addition of routes to the routing tables [16]. Both BGP connection categories exchange the same BGP message type. However, we focus in our implementation on changing the content of the update messages that are defined in [16] and [17] as follows:

- UPDATE: Used to transfer routing information between BGP peers. The information in the update message can be used to construct a graph that describes the relationships of the various Autonomous Systems. By applying rules, routing information loops and some other anomalies may be detected and removed from inter-AS routing. An UPDATE message is used to advertise feasible routes that share common path attributes to a peer, or to withdraw multiple unfeasible routes from service. An UPDATE message may simultaneously advertise a feasible route and withdraw multiple unfeasible routes from service. The UPDATE message always includes the fixed-size BGP header, and also includes the other fields [16, 17].

Also, BGP go through six states that lead to an established state and a successful exchange of routes. During each of the states, the peers must send and receive messages, process message data, and initialize resources before proceeding to the next state. If the process fails at any point, the session is torn down and the peers both transition back to an Idle state and begin the process again. Each time a session is torn down, all routes from the peer who is not up will be removed from the tables, which causes downtime [16, 17, 18]. We see in chapter 3 how all states of BGP are represented in OPNET. All states of the connection are described in [16] and [17] as follows:

- Idle: In this state, BGP FSM refuses all incoming BGP connections for this peer. No resources are allocated to the peer.
- Connect: In this state, BGP FSM is waiting for the TCP connection to be completed.
- Active: In this state, BGP FSM is trying to acquire a peer by listening for, and accepting, a TCP connection.
- Open Sent: In this state, BGP FSM waits for an OPEN message from its peer.
- Open Confirmed: In this state, BGP waits for a KEEPALIVE or NOTIFICATION message.
- Established: In the Established state, the BGP FSM can exchange UPDATE, NOTIFICATION, and KEEPALIVE messages with its peer

After the established state is reached, routes learned via BGP have attributes that are used to determine the best route to a destination when multiple paths exist to a particular destination. Such attributes are of great importance to us because they provide the much needed flexibility in controlling traffic paths

eventually. We explain some of the common attributes that are used in the selection process and in the course of our work:

- Multi-Exit Discriminator: The multi-exit discriminator (MED) acts as a metric to let neighbouring ASes know about the preference of the advertising AS. It can affect which route is used to send traffic to the advertising AS.
- AS_path: This attribute is used to announce the path of the route. Each time the route is advertised from an AS, this AS adds its number to the path sequence. Also, this attribute acts as a loop prevention mechanism. AS path attribute is the main focus of the thesis.
- Next-Hop: The next-hop attribute is used to include the IP address of the advertising router. The value of the next-hop IP address depends on the type of the BGP connection.

2.2 BGP Route Selection Process

In order for a router to select the best path possible, it has to go through a filtration process first. Not all routes that are received from a neighbour going to be considered for the selection process. Routers ignore paths in some circumstances as mentioned in [19]. After the filtration process is conducted, another process is invoked to choose the best path. We quote from [19] some of the related steps in this decision process:

1. Prefer the path with the highest LOCAL_PREF.
2. Prefer the path that was locally originated.
3. Prefer the path with the shortest AS_PATH.
4. Prefer the path with the lowest origin type.
5. Prefer the path with the lowest multi-exit discriminator (MED).
6. Prefer eBGP over iBGP paths
7. Prefer the path with the lowest IGP metric to the BGP next hop.
8. When both paths are external, prefer the path that was received first (the oldest one). This step minimizes route-flap because a newer path does not displace an older one.
9. Prefer the route that comes from the BGP router with the lowest router ID.
10. Prefer the path that comes from the lowest neighbor address.

The above information is vital for grasping the concepts introduced in this thesis. We see in chapter 4 how this selection process plays an important role in affecting data traffic paths and the dynamics of BGP's fault reaction.

2.3 Route Flap Damping

To maintain scalability of a routed internet, it is necessary to reduce the amount of change in routing state propagated by BGP in order to limit processing requirements. The primary contributors of processing load resulting from BGP updates are the BGP decision process and adding and removing forwarding entries [20]. Since the early development of BGP, problems in convergence were discovered and solved. However, with the growth of ISPs and expansion of the Internet topology, those solutions were not recommended any more. One of those solutions was Route Flap Damping.

RFD is based on timers that monitor the behavior of prefixes in the network. Each prefix has a penalty value that will be increased whenever this prefix experiences a change. The penalty value should decrease certain value when a specified amount of time is passed with no changes to the prefix. For example, if a prefix is advertised to a neighbour, then this neighbour increases the penalty value by 1000. After that, if this same prefix experiences a withdrawal, the neighbor increases the penalty value by 500. The process goes on until the threshold is reached e.g. 2000. Typically, this behavior indicates instability in the network and more specifically a flapping link. To prevent more updates and unnecessary convergence activity, this neighbor stops sending updates about this prefix that has reached the threshold. As time passes by, the penalty is decreased based on the configuration of the router. When the penalty reaches a certain value i.e. half-life, the prefix will be active again and allowed to be propagated throughout the network. Following are the important timers related to the RFD:

- Suppress limit: A route is suppressed when its penalty exceeds this limit.
- Half-life: Once the route has been assigned a penalty, the penalty is decreased by half after the half-life period.
- Reuse limit: As the penalty for a flapping route decreases and falls below this reuse limit, the route is unsuppressed. That is, the route is added back to the BGP table and once again used for forwarding.
- Maximum suppress limit: This value is the maximum amount of time a route can be suppressed.

Surprisingly, route flap damping can significantly exacerbate the convergence times of relatively stable Routes [21] sometime for up to an hour. As mentioned in [22], C. Labovitz proved that a single route withdrawal can result in other routers exploring a sequence of alternate paths before deciding that the destinations are unreachable. This is commonly known as path exploration. This process can trigger secondary flaps that can lead the RFD process to reach the suppression limit. This prevents the

advertisement of subsequent updates that contribute to the stability of the network resulting in delayed convergence [21]. Furthermore, Routing Working Group in [5] proposes that with the current implementations of BGP flap damping, the application of flap damping in ISP networks is NOT recommended. If flap damping is implemented, the ISP operating that network will cause side-effects to their customers and the Internet users of their customers' content and services. These side-effects would quite likely be worse than the impact caused by simply not running flap damping at all.

2.4 Minimum Route Advertisement Interval

As the effort continues to stabilize BGP convergence and increase its reliability, minimum route advertisement interval (MRAI) was introduced. It was observed that by minimizing the number of updates triggered and sent by BGP, we contribute positively to the stability of BGP [23, 24]. MRAI limits the minimum time interval between two consecutive update messages sent for the same destination. When MRAI timer is set, the update couldn't be sent out immediately unless the timer is timed out. There are two implementation methods for MRAI timer. One is per-peer MRAI timer, the other is per destination MRAI timer [24]. The BGP convergence time is affected by the duration of MRAI and the implementation of MRAI timers. The default MRAI value (30 s) is used in the majority of today's routers [25].

MRAI allows a router to privately explore its alternative choices for best route without exposing its neighbors to the intermediate step, thus reduces the number of updates during path exploration [26]. Hence, MRAI contributes to the reduction of BGP updates by sacrificing the rapid convergence of the network. On the other hand, the previous recommended value for MRAI is not proven as a standard for all ISP networks. The suggested default values for the MRAI given [26] are deprecated. The appropriate choice of default values is left to the discretion of implementers. Implementations should provide a means to allow operators to choose values appropriate to their requirements, on a per-peer and per-prefix basis. Implementations may exempt withdrawals from the MRAI timer [26]. Hence, the need for a new mechanism that would reduce BGP updates and preserve convergence speed is needed.

2.5 Path Exploration Damping

In 2010, a new solution similar to MRAI called Path Exploration Damping (PED) was proposed. The idea took into consideration the observation that a small number of prefixes cause the largest number of updates [22]. However, the idea of the suppression mechanism would depend heavily on the AS path attribute. The change in the AS path attribute was a good indicator of path exploration in the network. PED delays update messages which would announce a route with a same-length or longer AS Path than the previously announced route for the same prefix [8]. Also, all other updates and withdrawals are sent out without any delay. In addition, the suppression technique has a timer per prefix per neighbour. Whenever the timer is active, a temporary outbound queue is used to hold the updates that are of the same prefix but with longer AS path, same AS path with different attributes, same AS path, or with no change at all in the path or attributes. PED would aspire to achieve reachability as quickly as possible in the case of path exploration, with optimality of the path used being a secondary goal [8].

The authors of [8] claim that experimental analysis of actual BGP announcements, updates and withdrawals captured from the Internet quantify this approach and measure a reduction in the update load of up to 32% while Path Exploration events are reduced by 77%. This method was successful to lower the number of updates but sometimes prolonged the total convergence duration [9]. There is no correlation between the reduction of BGP updates and its total event duration. Furthermore, PED didn't take into consideration the history of the prefix weather it has been stable or not. On the other hand, all the problems associated with PED were solved in CAGG as we see in the next section.

2.6 Churn Aggregation Approach

In 2011, Wang et al focused on the AS-PATH Attribute through aggregating AS-Path attributes for highly active prefixes. This method called Churn Aggregation (CAGG) focuses on the observation that a small number of AS PATHs are explored by each highly active prefix [26]. In [9], CAGG proved to give better results than RFD, MRAI, and PED. Wang et al experiments with real BGP data show that CAGG can reduce as much as 50% of BGP updates, and 60% of BGP path exploration duration in its best case, while on average 28.1% and 32% respectively across 36 RouteViews monitors [9].

CAGG implementation lead to a slightly modified version of the BGP standard named aBGP (Aggregation BGP). aBGP is similar to RDF in the sense that per prefix per neighbour there is a history list that keeps track of all updates and withdrawals. Each time a new update or withdrawal is received, the

penalty counter for that prefix would increase. When the penalty reaches a cut off threshold, the CAGG algorithm would collect all the AS paths of the different updates and unify them into one normalized or aggregated path. All subsequent updates would be suppressed as long as the penalty is over the cut off threshold. Like RFD, there is a $T_{inactive}$ threshold where the aggregated AS path is removed from the history set if the penalty goes beyond this limit. Of course, the penalty would decrease based on a specific timely criteria just like RFD and this would be a configurable attribute. There are also some additions like path frequency which increases each time a certain path is used. Whenever the frequency falls below a certain configurable limit, the path is removed from the history list that is used to create the aggregated AS path. This leads to a more accurate aggregated path as a result. In Churn Aggregation, function $r_{I,d}$ is defined as the set of changing routes r_i to destination d which are disseminated to an upstream BGP neighbour N, at variable times t_i. Hence, information propagated to this neighbor is denoted as $r_{I,d} = \{(r1, t1), (r2, t2)...(r_i, t_i)\}$ for $i \geq 1$. Churn Aggregation (CAGG) tries to compute a new route $r'_{I,d}$, which represents the new set of routes with aggregated AS paths [26].

In order to understand how CAGG works, figure 3 shows a typical BGP network where each of routers represents an ISP. At this point, we do not go into the details of the ISP and iBGP relationships. In this network, AS7 prefers the path through AS4, followed by AS5 and finally AS8 to reach the destination d originated by AS1. In this set up we assume there is a slight delay of updates reaching AS7 from AS5 and the link between AS1-AS2 is flapping. Standing on AS9, it sees several updates converging finally to $d_{7,4,2}$ or $d_{7,8,6,3}$. This process of sending updates and withdrawals is reduced by implementing CAGG at

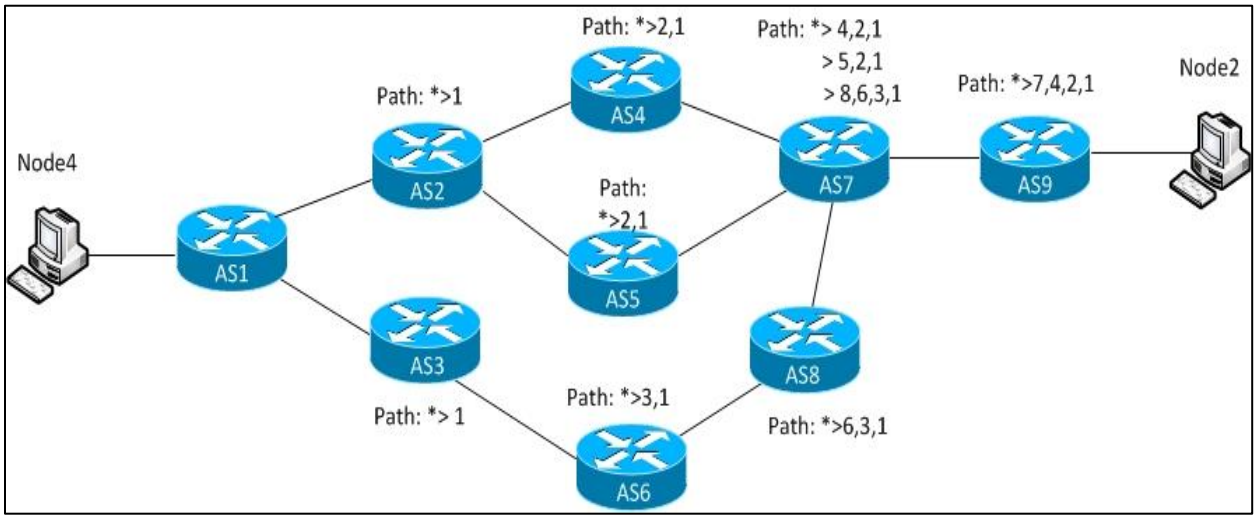


Figure 3 BGP network with a flapping link

AS7 which suppresses all subsequent updates sent to AS9 after the threshold of the penalty is reached. When the cut off threshold is reached, AS7 only sends one advertisement r'_i of prefix d with an AS path attribute of 7,{4,5,8,2,6,3},1. AS9 continues forwarding traffic to AS7 without going through convergence. In addition, if the aggregated path at time i is equal to the path sent out in i-1, then this new update is also suppressed. On the other hand, withdrawals are sent immediately which achieves full convergence as soon as possible. The following is the algorithm that takes r_i, which is the best route to a prefix at a given time and produces r'_i, which is a route to the same prefix with an aggregated AS path. Also, identity is the identifier of the CAGG Aggregator composed of the AS number and router ID [9, 26]:

```
Input: ri-1, ri, r'i-1, History List of used paths
Output: r'i

if ri = ri-1 then
        r'i = r'i-1
else
        r'i = ri
if ri ≠ Withdrawal then
        if ri.AS_Path is represented by r'i-1.AS_Path then
                r'i.AS_Path = r'i-1.AS_Path
        else
                r'i.AS_Path = Aggregate(ri.AS_Path , Histroy list)
        if r'i.AS_Path ≠ ri.AS_Path then
                r'i.Aggregator = identity
return r'i
```

Figure 4 CAGG Algorithm for constructing aggregated AS path in [9]

CAGG does not choose all the AS paths in the history list. Instead, it chooses some of them based on a configurable variable like the top 5 frequent paths, which are the most used paths for a specific destination. Also, the Aggregate function is used to produce the aggregated AS path which can be a shorter path that is part of r_i.AS_Path or can be r_i.AS_Path itself when the aggregation fails. The notation r_i.AS_Path represents the sequence of AS paths used by route r_i to reach a destination. Unlike regular BGP aggregation, all other attributes like MED, NEXT_HOP, COMMUNITY...etc. stay preserved in the aggregated path except for the Aggregator attribute that is changed to reflect the process of CAGG taking

place at a certain router. This will be the same algorithm used to produce the aggregated path in our work. However, the workflow of the CAGG algorithm itself is changed to reflect our enhancements and avoid possible deficiencies as we see in the next chapters. For more details about the workflow of aBGP, refer to [9] and [26].

As mentioned before, the results presented in [9, 26] clearly show the improvement achieved in reducing convergence times while reducing the amount of BGP churn resulting from path explorations or any event that causes unnecessary updates. However, after careful study and analysis of the algorithm suggested in [9], we came to the conclusion that data traffic is still affected by the convergence activity happening downstream. Wang et al [9,26] suggested a successful workflow for handling BGP updates and reducing convergence but did not mention anything about how the aggregator itself is going to handle the flaps downstream. If we run the same scenario in figure 3, then AS7 drops the traffic for the brief time it is going through path exploration even though the network upstream is converged and the aggregated path has stabilized the network. Furthermore, when the CAGG algorithm sends out an aggregated AS path, this path becomes an attraction point to all traffic due to path shortening. Unlike route aggregation where a shorter mask is less desirable, shorter AS paths are preferred over longer paths if no other attribute affects the selection process. In [9, 26], these problems are not mentioned or taken into consideration leaving an important enhancement that could lead to a more stable solution feasible to be applied in real life. In the rest of this document, we formalize the problems mentioned and prove their existence through simulation while modifying existing algorithms to solve areas of dispute and prove the efficiency of our solution through OPNET simulation as well.

Chapter Three

Test Network Design and Protocol Analysis

In this chapter we focus on our analysis and simulation tool which is OPNET Modeler 15 [27]. We start with a brief introduction to OPNET and its simulation architecture. We also take a look on how OPNET implements and simulates BGP. Understanding how OPNET conducts simulation events for BGP is important for understanding how we programmed our changes and achieved our results. After that, we describe the design of our simulation network and assumptions during simulations. Finally, we conduct two simulations to understand the behavior of BGP with and without failures. The first simulation runs a smooth network with zero faults or flaps. The second simulation introduces link flaps and path exploration to the network. The results of the two simulations show clearly the essential problem existing today with the BGP implementation.

3.1 OPNET Modeler Overview

OPNET Modeler is the De facto Industry Standard for Network Modeling and Simulation [27, 28, 29, 30]. It is an event based simulator where time only advances when an event happens. It provides the end user with the ability to design, model, implement, and analyse network protocols, devices and applications through a graphical user interface. OPNET uses its own model library which is an extensive library of standard-based and vendor models. However, OPNET Modeler gives the ability to create new models or modify existing models. Furthermore, OPNET has a wide pre-programmed set of statistics that can be selected at the time of the simulation to help assess and analyse the network. The components in OPNET are programmed using an object-oriented approach using C and C++ languages. In addition, it uses its own Proto-C which reference the code augmented by OPNET specific functions. It is difficult to cover all the aspects of OPNET in this thesis. However, we focus on two important concepts necessary to grasp how we implemented our code and simulation: Process models and Event interrupts. Parts of the descriptions of the concepts in the following paragraphs are referenced from the help documentation of OPNET Modeler [28].

Process Models are in the heart of any implementation in OPNET Modeler. Process models are specifications for the dynamic behavior of processes. To be complete, a process model must describe the actions that a process will implement under all possible circumstances. In many cases, processes that are

executing within discrete event simulations represent a real-world equivalent protocol or algorithm. Individual processes or groups of processes are defined to implement a particular task when placed within a queue. Each process can invoke other processes that perform specific code and return the handle back to the parent process. In this case, the invoked process is called a child process. OPNET Modeler supports one language for developing models of processes. This language, called Proto-C, is supported by the Process Editor which is integrated into the OPNET Modeler application. Proto-C provides a powerful and efficient method for describing the behavior of discrete event systems. One important feature of Proto-C is the representation of processes through State Transition Diagrams (STD). Processes are represented by States which are generally used to represent the top-level modes that a process can enter. Transitions specify the changes in state that are possible for the process. STDs are graphically depicted in the Process Editor. Proto-C defines two types of states, called forced and unforced, that differ in execution-timing. In Proto-C diagrams, forced states are graphically represented as green circles, and unforced states are drawn as red circles. Unforced states allow a pause between the enter executives and exit executives, and thus can model true states of a system. After a process has completed the enter executives of an unforced state, it blocks and returns control to the previous context that invoked it. Forced states are so called because they do not allow the process to wait. They therefore cannot be used to represent modes of the system that persist for any duration. In other words, the exit executives of a forced state are executed by a process immediately upon completion of the enter executives. Therefore the exit executives of a forced state are generally left blank, because they are equivalent to the same statements placed at the end of the enter executives. [28]

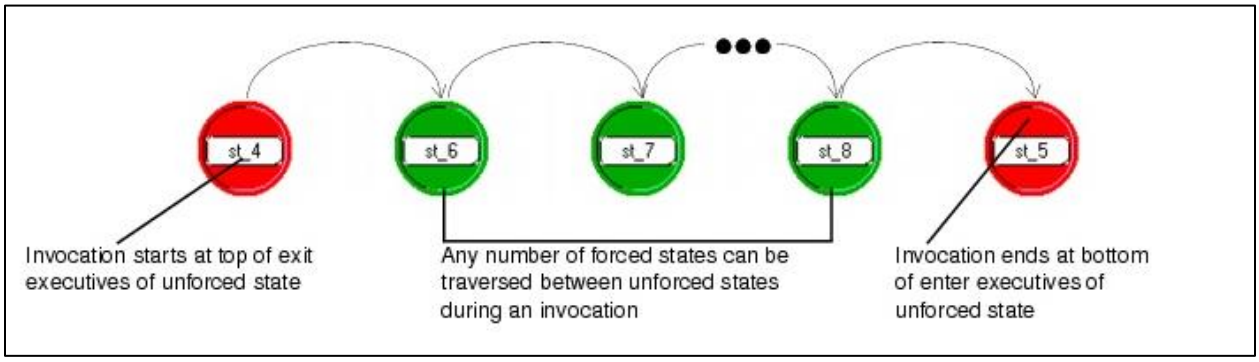


Figure 5 Example of a process with multiple forced and unforced states

Most of our changes to the BGP process are programmed under the forced and unforced states of the BGP model. Also, other changes were conducted under the BGP C language [31, 32] headers and libraries. In the following sections, we see more details about the BGP process itself and how it is represented in OPNET Modeler.

Event interrupts are fundamental for the operations of OPNET. Like all other subsystems in OPNET Modeler models, processes are driven by events. When an event is actually delivered to a process, it is termed an interrupt. The process is said to be interrupted which means that it is invoked to allow it to take some action in response to the interrupt. A process follows an alternating cycle of invocation and rest periods. Invocations may occur on an arbitrary basis depending on the timing of externally and internally generated events. A process always begins the simulation in a resting mode, waiting to be invoked; a process that is waiting in this condition is said to be blocked. Invocation allows a process to resume execution and to do new actions. After these actions are completed, the process must again block, returning control to the Simulation Kernel so that other events in the system may be executed. Because these subsequent events may be scheduled for arbitrarily near future times, and even for the same time as the current event itself, the invocation must occur without allowing any time to elapse.

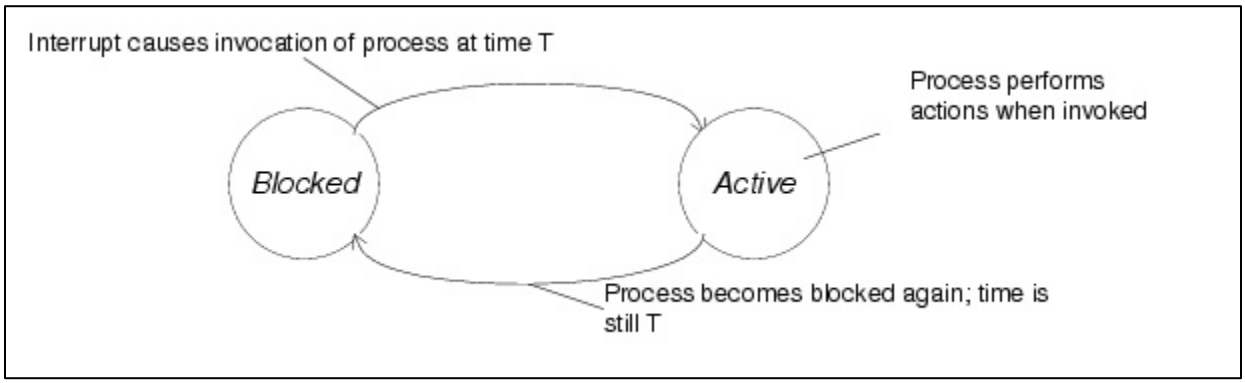


Figure 6 Interrupt Operations in OPNET

3.1.1 BGP Implementation in OPNET Modeler

In OPNET Modeler, the BGP process is implemented in two models. All router nodes that support BGP have a BGP module that spawns BGP-related processes in the router. The two processes that model BGP are: bgp and bgp_conn. The bgp process model is the root process model. It is responsible for handling all incoming BGP packets. It also reads the entire BGP configuration during initialization and creates one instance of the bgp_conn process model for each BGP neighbor configured on the node. We can see in figure 7 all the states that BGP go through to handle updates. The "Update Message" state is an important state that is entered when bgp gets an update packet from a neighbor [28]. Most of our changes to the protocol were incorporated in this state. Based on the neighbor from whom the packet has arrived, the appropriate bgp_conn process is invoked.

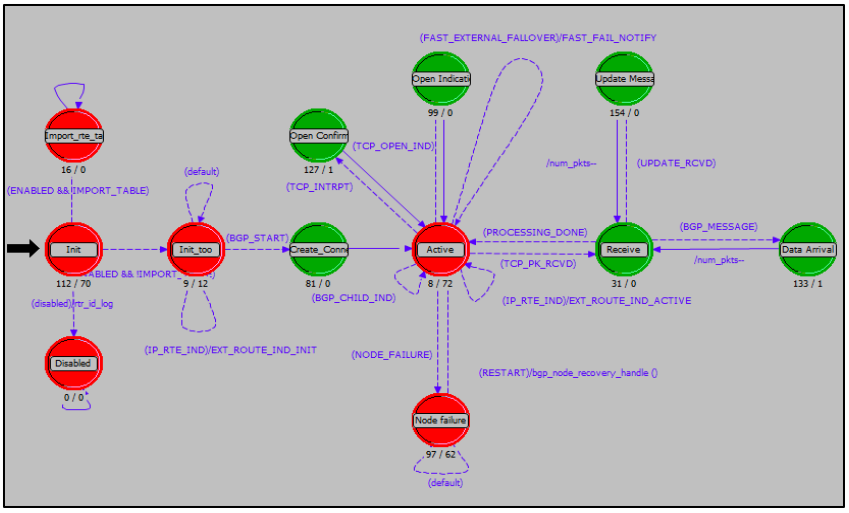


Figure 7 bgp process model

The bgp_conn process then handles all the reachability and unreachability information present in the incoming packet by updating the RIB-IN of the connection and handing the control back to the parent *bgp* process. The parent *bgp* process then processes all new information. It removes the unfeasible routes from the LOCAL-RIB, updates the LOCAL-RIB with new routes after running the BGP decision algorithm, and updates the IP forwarding table, if necessary. Once all of the routes are processed, the parent process invokes all of the other bgp_conn child processes, so this new information is propagated to other neighbors [28]. We see in figure 8 all the states that BGP goes through to achieve the established state. This process is invoked per neighbor.

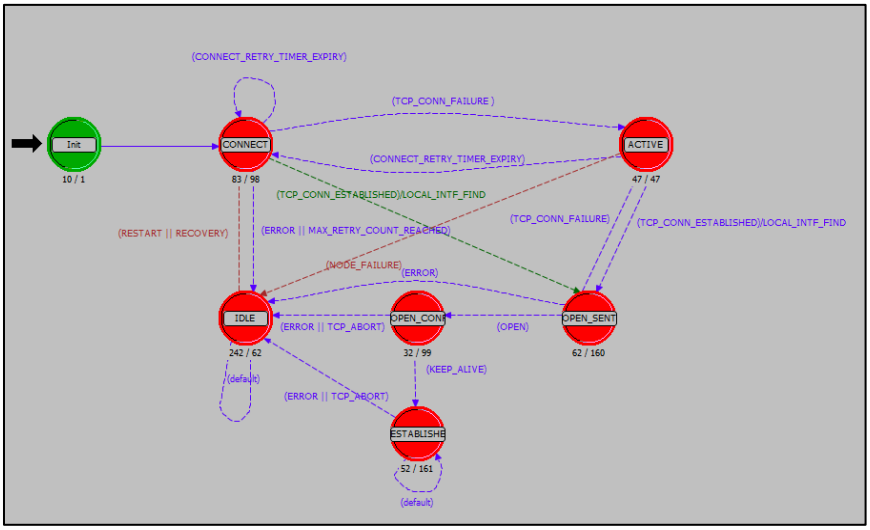


Figure 8 bgp_conn process model

3.2 Simulation Network Design

Based on many research papers in the BGP area [9, 26, 35, 36, 37, 38] we constructed our test network topology to be similar to the topology used in many of such papers. Our network topology resembles the one used in [9] with some modifications. This resemblance makes our results as comparable to the results in [9] as possible. Our network topology used in our simulation is shown in the figure 9. This topology shows multiple paths between the source and the destination. Each AS used represents a service provider. In each AS, we don't focus on what happens inside the provider. Hence, all BGP connections used are EBGP. During normal behavior, the source Node2 takes the path AS9-AS7-AS4-AS2-AS1-Node4. The path through AS8 is always the least preferable due to its longer AS-Path.

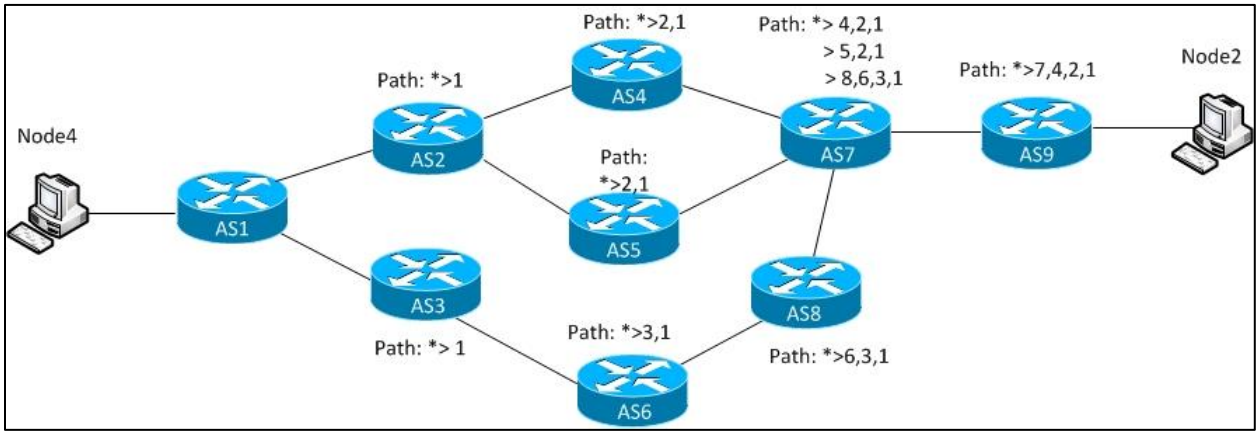


Figure 9 Test Network Topology

We also established a server-client communication in order to visualize network convergence effect on traffic. There are three applications running: Email, FTP, and HTTP. All mentioned applications are configured to start simultaneously. Also, ICMP ping traffic is initiated from Node2 to Node4 and vice versa. Ping is sensitive to outages and can be interrupted immediately. Each ping packet is 1024 bytes long and the inter-arrival time between pings is 1 ms.

All routers are configured with BGP using OPNET GUI interface. Also, applications are configured using the Application Profile and Application Definitions nodes. All traffic is bidirectional between Node 2 and Node 4. We also used a "Failure Node" that controls link and node failures during the simulation. We describe the node failure in later chapters.

We also chose a set of pre-programmed statistic handles to collect different results that give us an insight to the operation of the protocol. The main focus of our analysis is on BGP Convergence Activity, Convergence Duration, BGP Traffic Sent and Received, Applications' Behavior, Traffic Drops, and Link

Utilization. Simulations are usually run to an equivalent of three hours of real network activity. This network topology is used in the majority of our simulation with the exception of two simulations mentioned later where the network is expanded to achieve proper results.

3.3 Standard BGP Simulation with No Flaps or Path Exploration (Simulation 1)

This simulation is performed to demonstrate the normal behavior of BGP without any problems such as link flaps and path exploration. The results of the simulation represent the best case scenario that can be used as a reference when comparing BGP behavior in later simulations. The other purpose of this simulation was to understand how OPNET Modeler implements BGP in the form of processes and interrupts. OPNET was linked to Visual Studio 2006 [33, 34] Debugger while this simulation was running and a step by step debug was conducted to see how update packets move and the values of the fields inside those packets. In this scenario, BGP is running on all routers and the path through AS4 is selected as best path. In this section, we are presenting the various results collected after running the topology for 3 hours.

3.3.1 AS 7 Convergence statistic (Simulation 1)

We focus on AS7 which is a critical AS that all traffic has to go through and which is the aggregator of AS paths in later chapters. We study the convergence of AS7 along with AS9 to understand the normal BGP convergence activity and traffic load. We can see that AS7 is experiencing a single convergence activity at the beginning of the simulation. BGP is configured to start at constant 70 seconds. Hence, we can see this activity around that time. Figure 10 shows the convergence activity, where convergence activity is reflected as a 1.

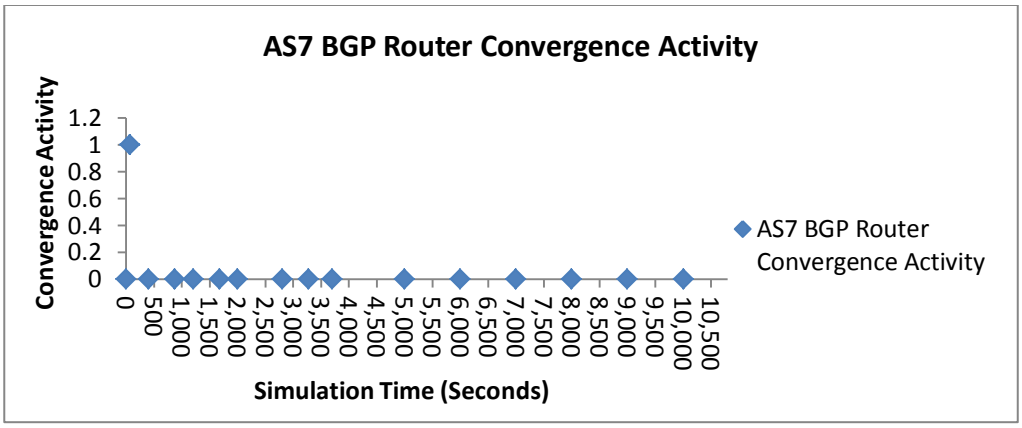
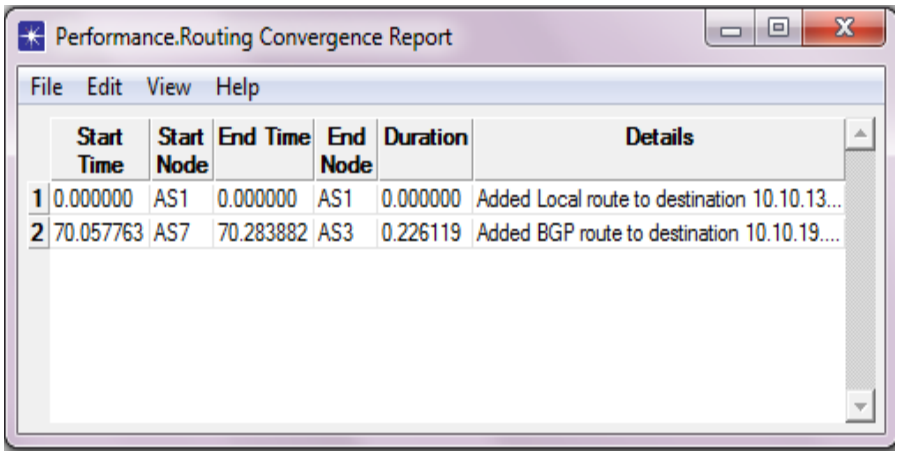


Figure 10 AS7 Convergence Activity (Simulation 1)

The following table shows a convergence report for the whole network. The only activity present is at second 70 where source and destination subnets are added to the various ASes. There are no further activity because BGP only sends updates when a change is discovered. In this simulation, network is stable and no changes are happening.



	Start Time	Start Node	End Time	End Node	Duration	Details
1	0.000000	AS1	0.000000	AS1	0.000000	Added Local route to destination 10.10.13...
2	70.057763	AS7	70.283882	AS3	0.226119	Added BGP route to destination 10.10.19....

Figure 11 Routing Convergence Report (Simulation 1)

Furthermore, all above results reflect on the traffic sent/received statistics. The figure below shows how only 160 packets where received and 220 sent in the beginning of the simulation to establish BGP connections with the neighbours of AS7.

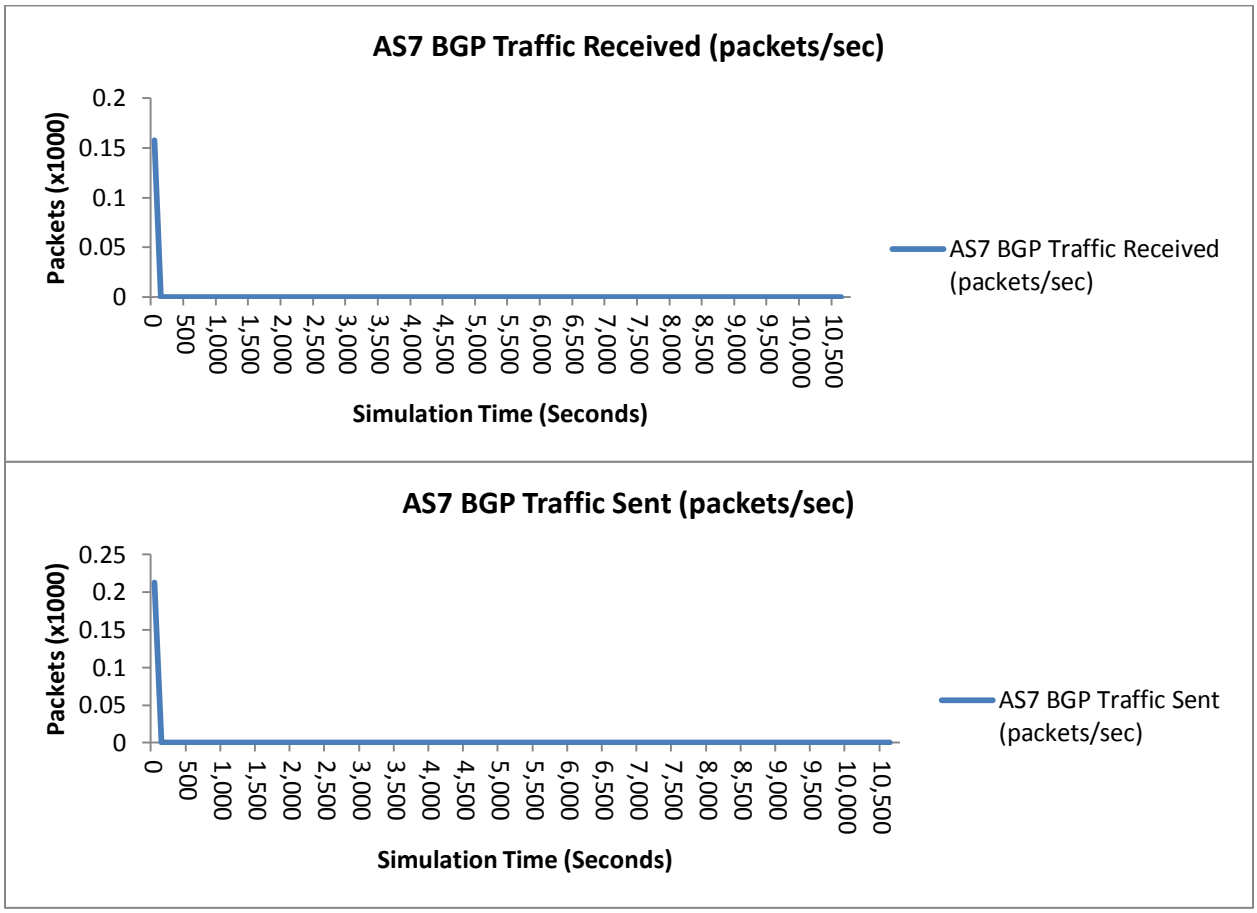


Figure 12 BGP Traffic AS7 (Simulation 1)

3.3.2 AS 9 Convergence statistics (Simulation 1)

Because AS9 is also not experiencing any changes or updates during the simulation, it reflects identical results to AS7. However, the BGP traffic sent/received is less because of the lower number of neighbors.

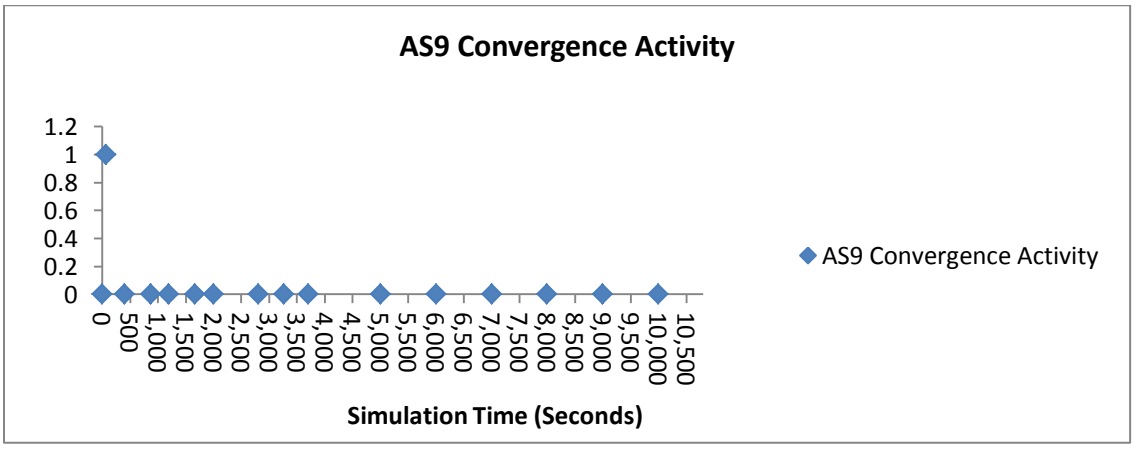


Figure 13 AS9 Convergence Activity (Simulation 1)

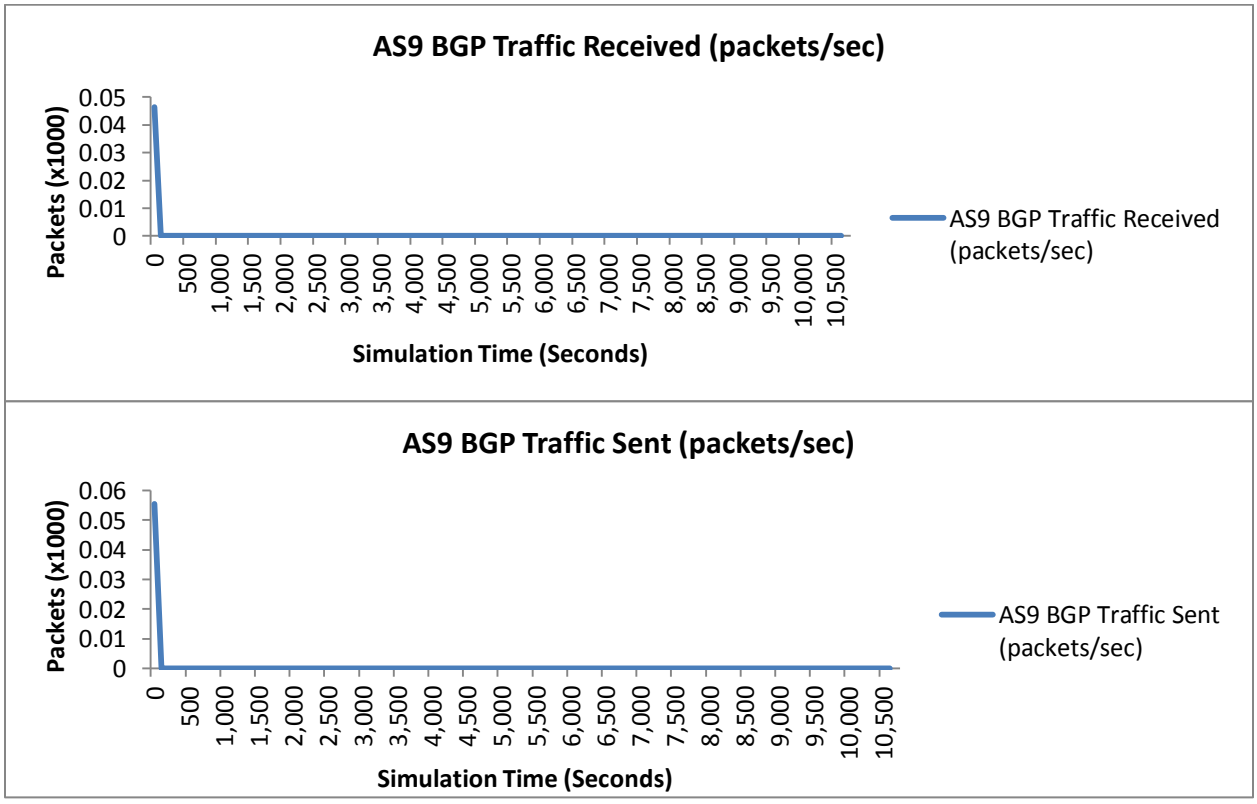


Figure 14 BGP Traffic AS9 (Simulation 1)

3.3.3 Application Statistics (Simulation 1)

We can clearly see how convergence and network activity affect traffic by measuring the delay and packet loss in applications. We chose three applications to measure the effects of network instability: Email, FTP, and HTTP. Also, ICMP traffic's delay is measured. In this document we always depict Email traffic statistics, IP Traffic Drops, and ICMP. All other related statistics are shown in Appendix A. In this simulation, network is stable and nothing affects the delay or reachability. Both Email Download and Upload Response are constant on 780 ms. This value can be considered as a reference for future comparisons with other scenarios. Also, traffic sent/received is averaged to almost 2600 packets/second.

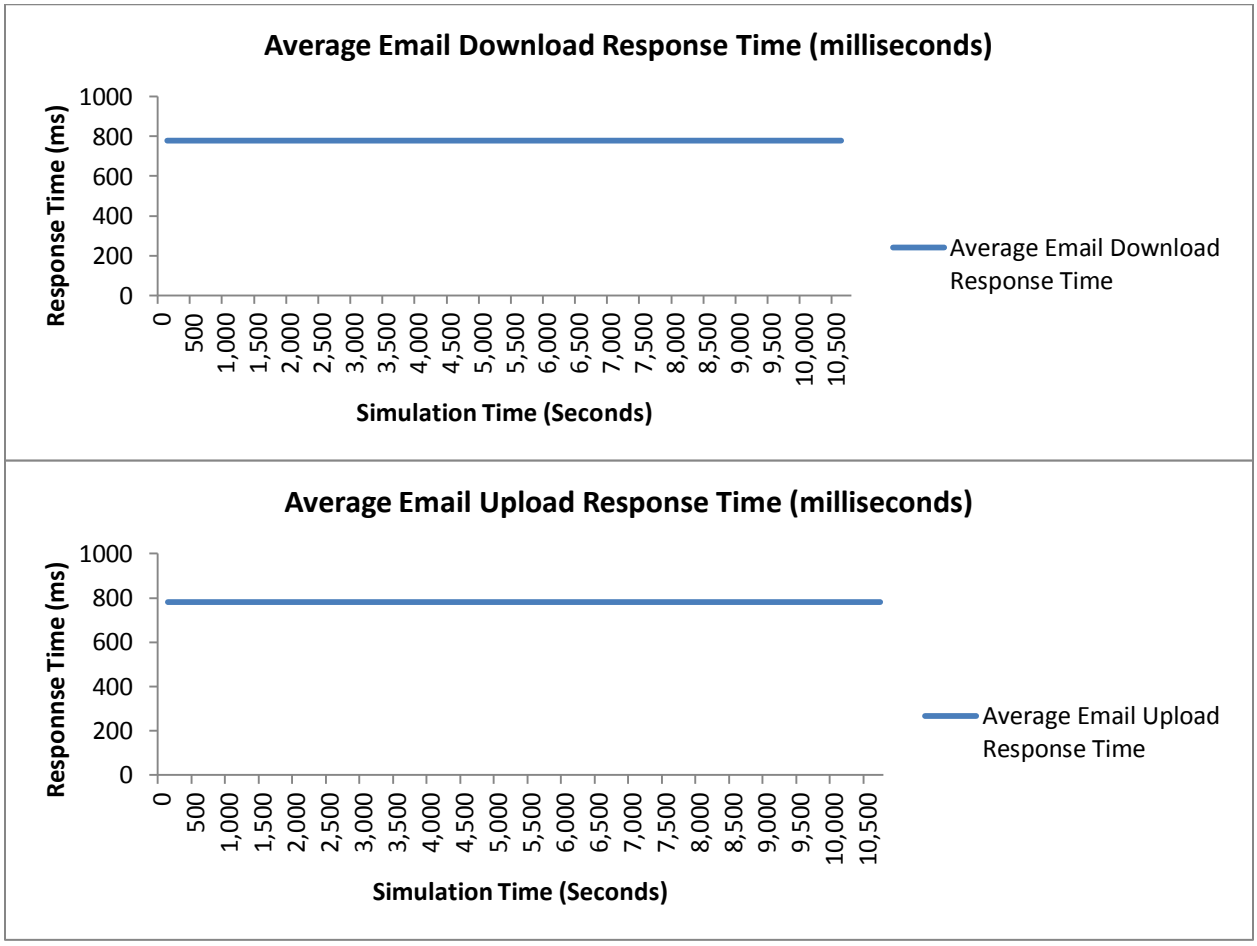


Figure 15 Email Download/Upload Response Time (Simulation 1)

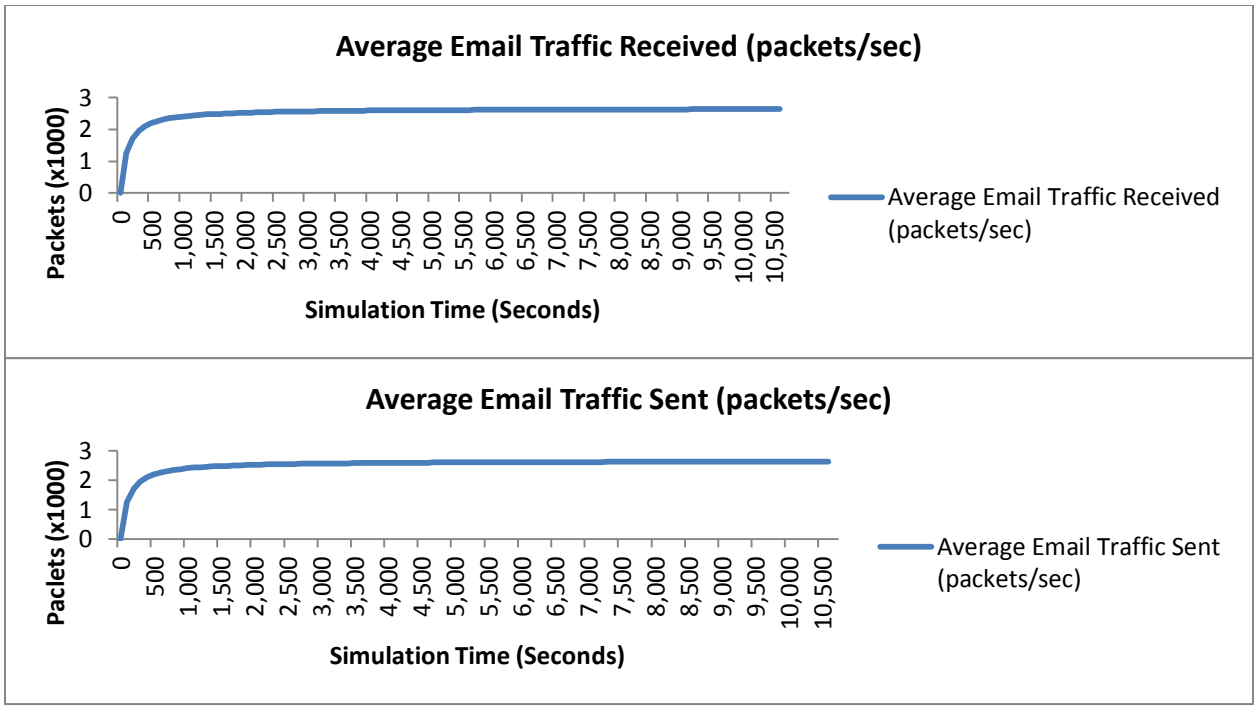


Figure 16 Email Traffic Sent/Received (Simulation 1)

Due to the stability of the network, ping replies received by Node2 are averaged to 11,000. Also, ping response time is almost 180 ms which is a reasonable number.

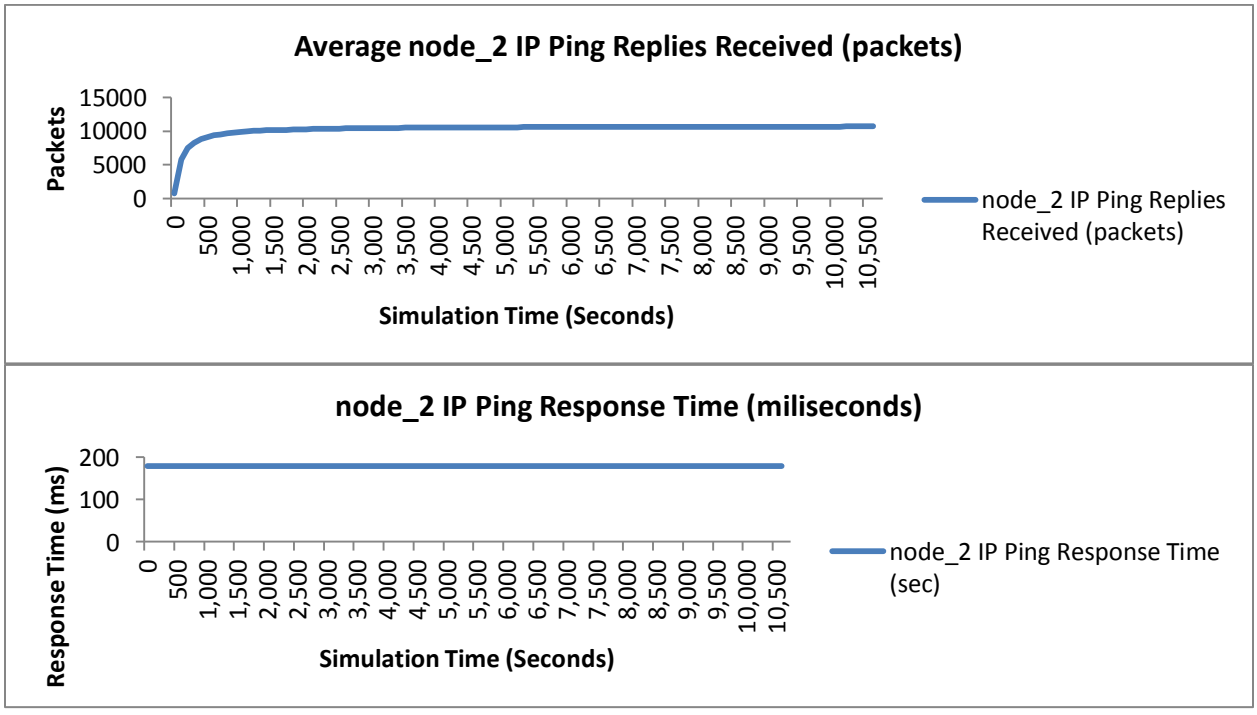


Figure 17 Ping Replies Received and Response Time (Simulation 1)

Also, IP Packet Drops is an important measure where we can truly see the effect of network convergence on traffic. In this case, we can't see a single packet drop. However, this will change in the following simulations.

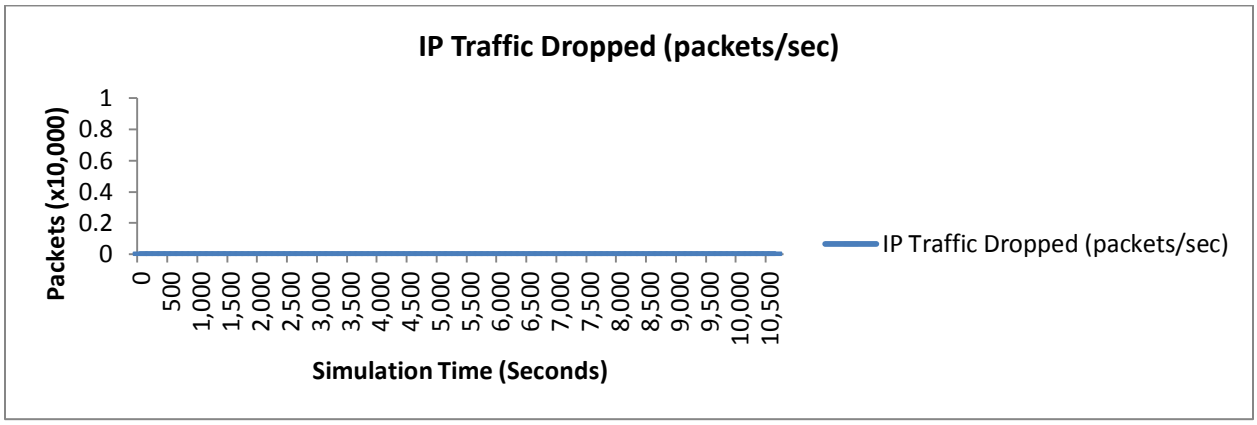


Figure 18 IP Packet Drops (Simulation 1)

3.4 Standard BGP Simulation with Flaps and Path Exploration (Simulation 2)

In this simulation, we create a failure scenario where the link between AS 1 and AS 2 is flapping. However, updates are reaching AS7 through AS4 faster than updates from AS5. This creates a process of path exploration at AS7. When a withdraw is received from AS4, AS7 immediately chooses the second best route which is through AS5. When a withdraw is received from AS5, AS7 routes through AS8. However during this period, all traffic routed to AS7 is being dropped at AS5 due to the lack of connectivity. Also, when the link AS1-AS2 comes back up again, AS7 chooses the route through AS4 and the process happens again on the next withdraw. This is a typical case of path exploration caused by link flaps and this simulation shows the original problem at hand that we try to solve.

The following is a summarization of the arrangement of link failure for this scenario achieved by using the Failure Node in OPNET. Note that link flaps start at 400 seconds and ends at 3700 seconds by a complete failure and traffic being routed through AS8.

- Link AS4-AS7 is transmitting updates faster than AS5-AS7.
- Link AS5-AS7 has a delay of around 100 seconds for updates sent from AS5.
- AS7 is going through path exploration by choosing AS5 as next hop, and after the update is received choosing AS8 as next hop. When the link comes up again, it goes back to AS4.
- The link failure is happening in the following intervals:
    - At time 400 second link is going down.
    - At time 800 second link is going up.
    - At time 1200 second link is going down.
    - At time 1600 second link is going up.
    - At time 2000 second link is going down.
    - At time 2400 second link is going up.
    - At time 2800 second link is going down.
    - At time 3200 second link is going up.
    - At time 3600 second link is going down.

Simulation is run over 3 hours. However, the network stabilizes after second 3700 on using the link AS7-AS8.

3.4.1 AS 7 Convergence statistic (Simulation 2)

The following convergence statistics shows an expected result of the failures designed. The stat-handle in OPNET marks a 1 whenever signs of convergence are present in the network. We can clearly see that the first failure starts at second 400 and flapping continues until second 3700. On the other hand, the convergence activity for the whole network is now extended due to the continuous updates being received. We see in the next chapter that between second 800 to 900, it is taking around 7.2 seconds for the network to converge. This is the average of the collective convergence duration the entire network.

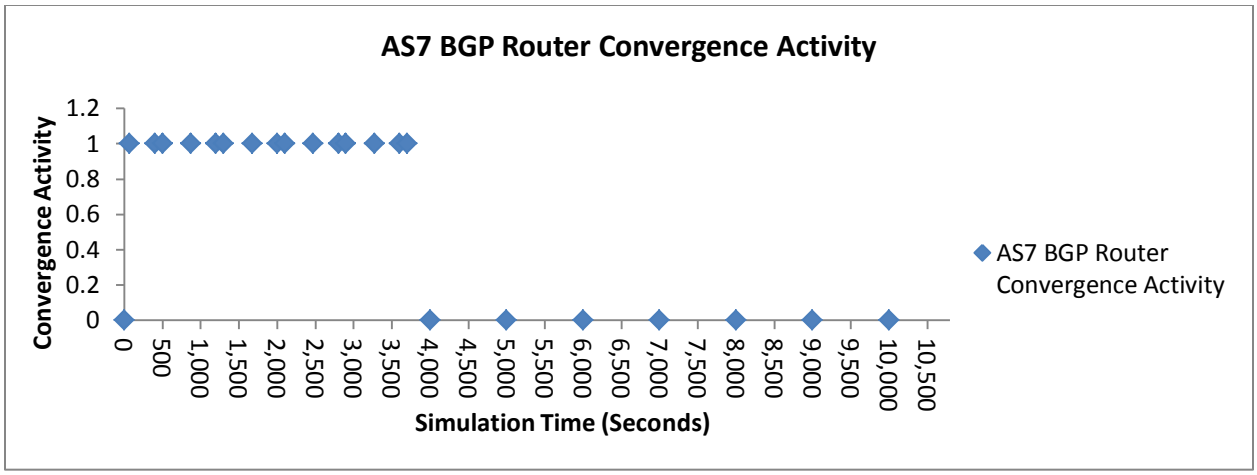


Figure 19 AS7 Convergence Activity (Simulation 2)

The BGP traffic sent/received also reflects the process of path exploration. We can see that the traffic sent by AS7 during convergence can reach a maximum of 37 packets. This amount of traffic can be avoided as we see in the next simulation.

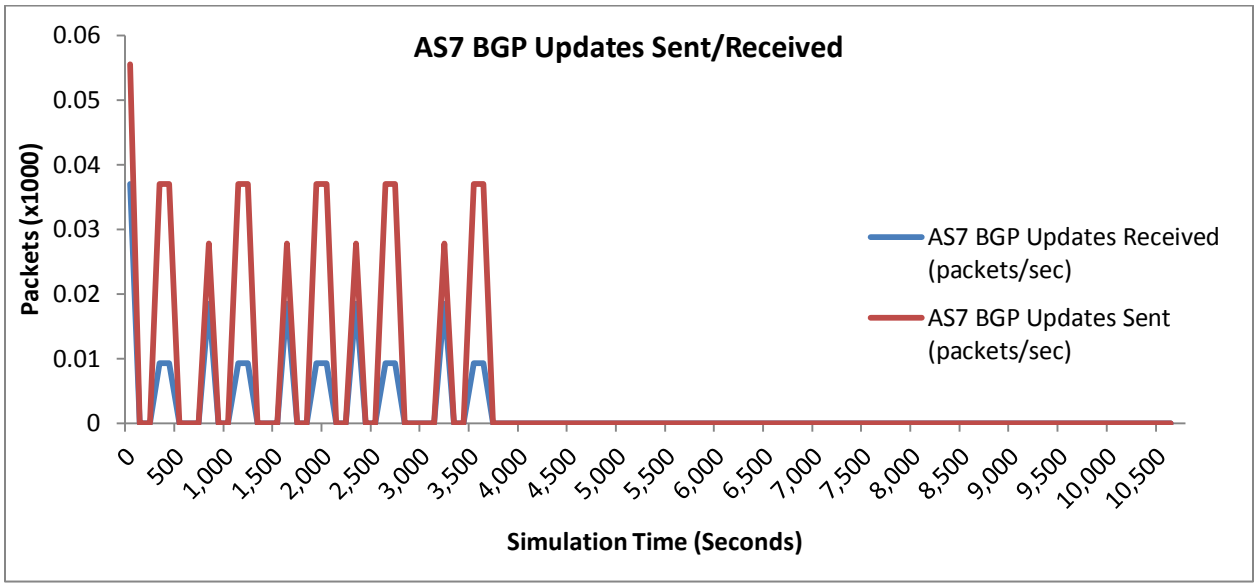


Figure 20 BGP Traffic AS7 (Simulation 2)

3.4.2 AS 9 Convergence statistics (Simulation 2)

By studying the convergence of AS9, we can see that all changes happening on AS7 are being reflected on AS9 as well. This is creating a BGP churn that is unnecessary in our case of flapping. We can see in the below figure that convergence activity is spanned over the entire instability period that AS7 is experiencing. Also, AS9 is receiving 9 update packets in times of instability. Also, at times of flapping, AS9 is receiving constant updates of extended period like between Second 400 and 500 where the link between AS7-AS5 is still up but would propagate a late update at 500 seconds.

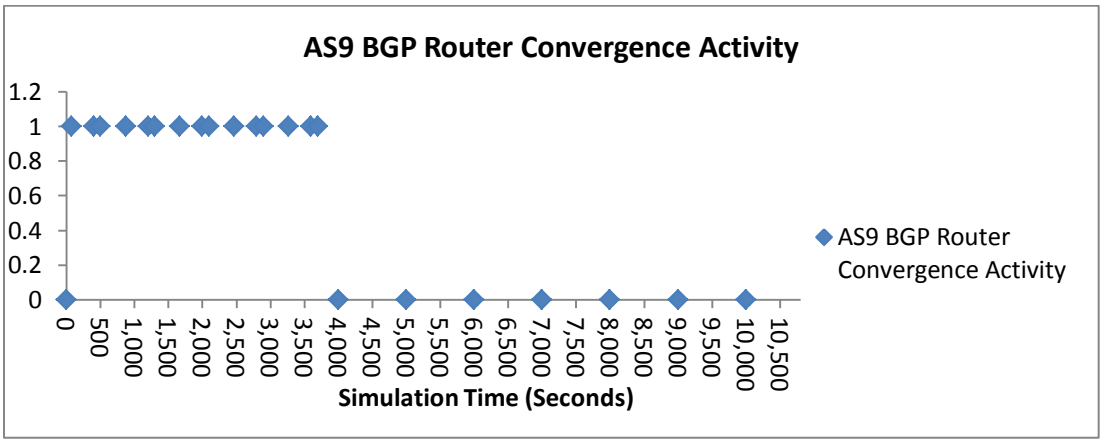


Figure 21 AS9 Convergence Activity (Simulation 2)

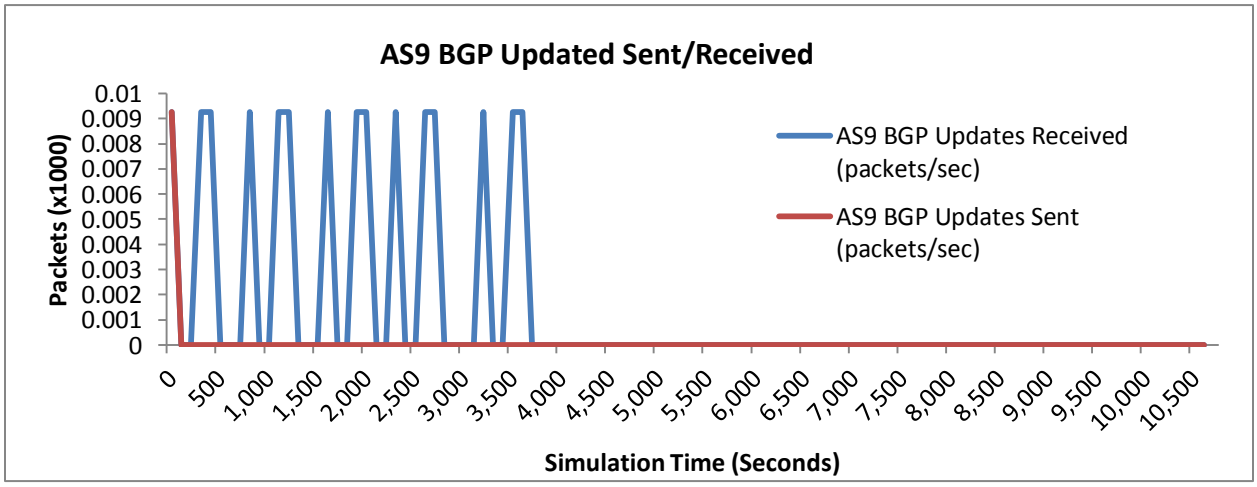


Figure 22 BGP Traffic AS9 (Simulation 2)

3.4.3 Application Statistics (Simulation 2)

In this simulation, it is expected to see periods of packet loss and high rates of delay and response. When flapping is happening, packets are being forwarded for an about 100 seconds or less towards AS5 which has lost reachability with the destination. In the following figure, we can see how Email traffic response is delayed up to 10.4 seconds compared with 0.72 (average) seconds in normal scenario. Also, notice that delay starts to decrease and stabilize when the network reaches stability at 3700 seconds. Traffic received/sent also reflects the effect of network convergence and path exploration.

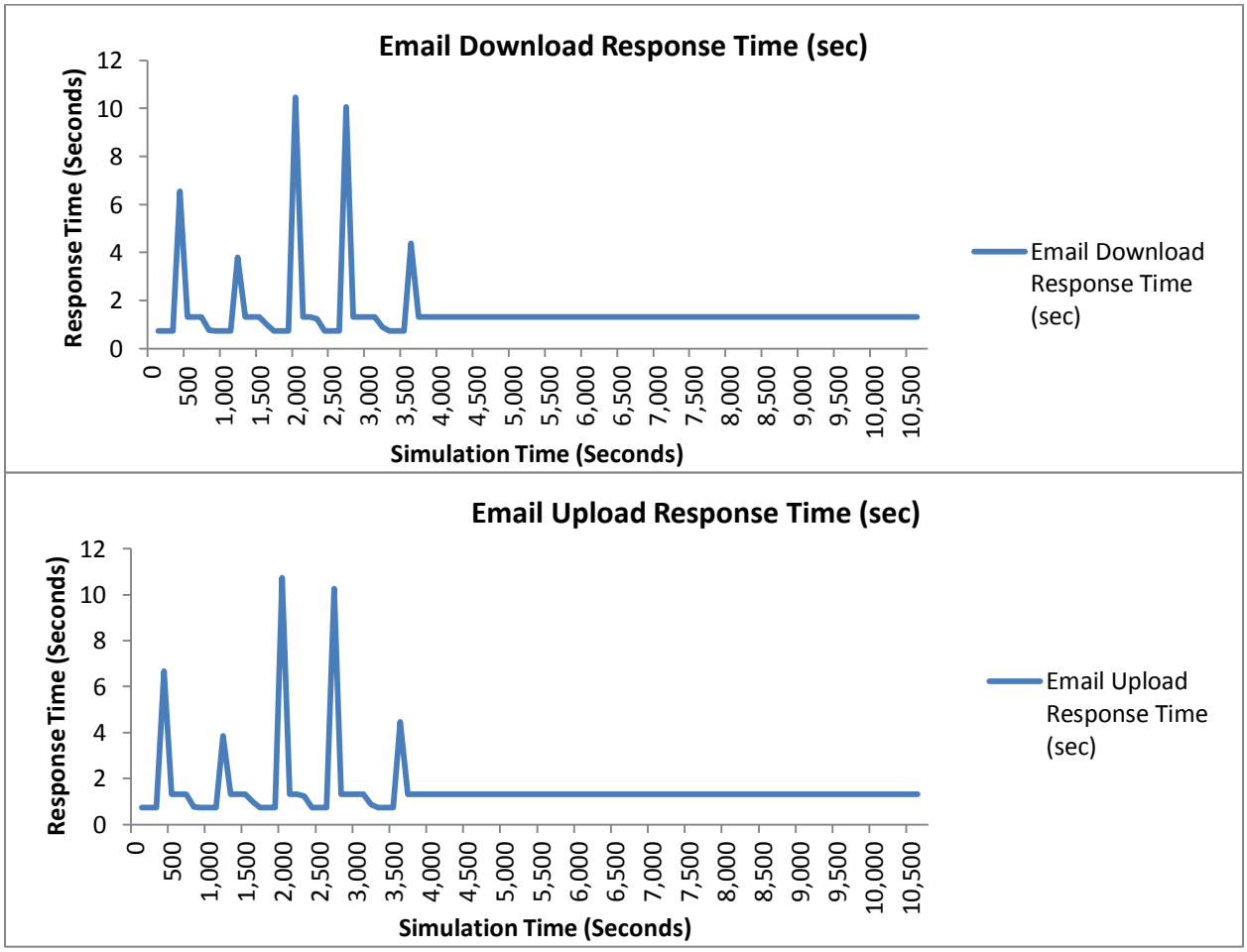


Figure 23 Email Download/Upload Response Time (Simulation 2)

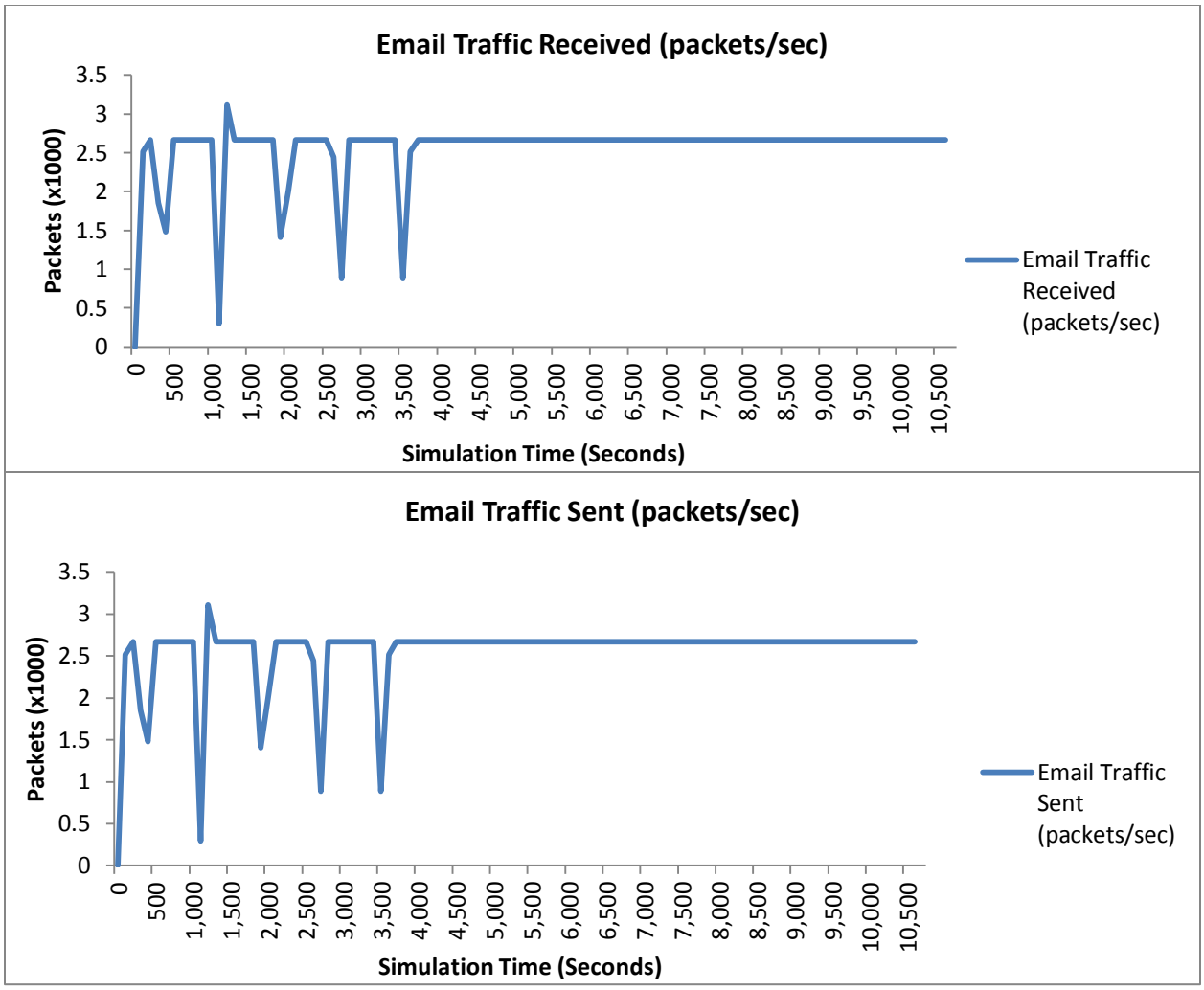


Figure 24 Email Traffic Sent/Received (Simulation 2)

Ping replies are lower than 11,000 now at convergence times and response time is delayed to 325 ms compared with 180 ms in Simulation 1. However, after second 3700, the response time stabilize on 325 due to the usage of congested and longer path through AS8. Also, note that all communications are two way, that's why we see some amount of ICMP packets being sent from Node 4 at times of convergence and disconnectivity (they are utilizing the path Node4-AS1-AS3-AS6-AS8-AS7-AS9-Node2).

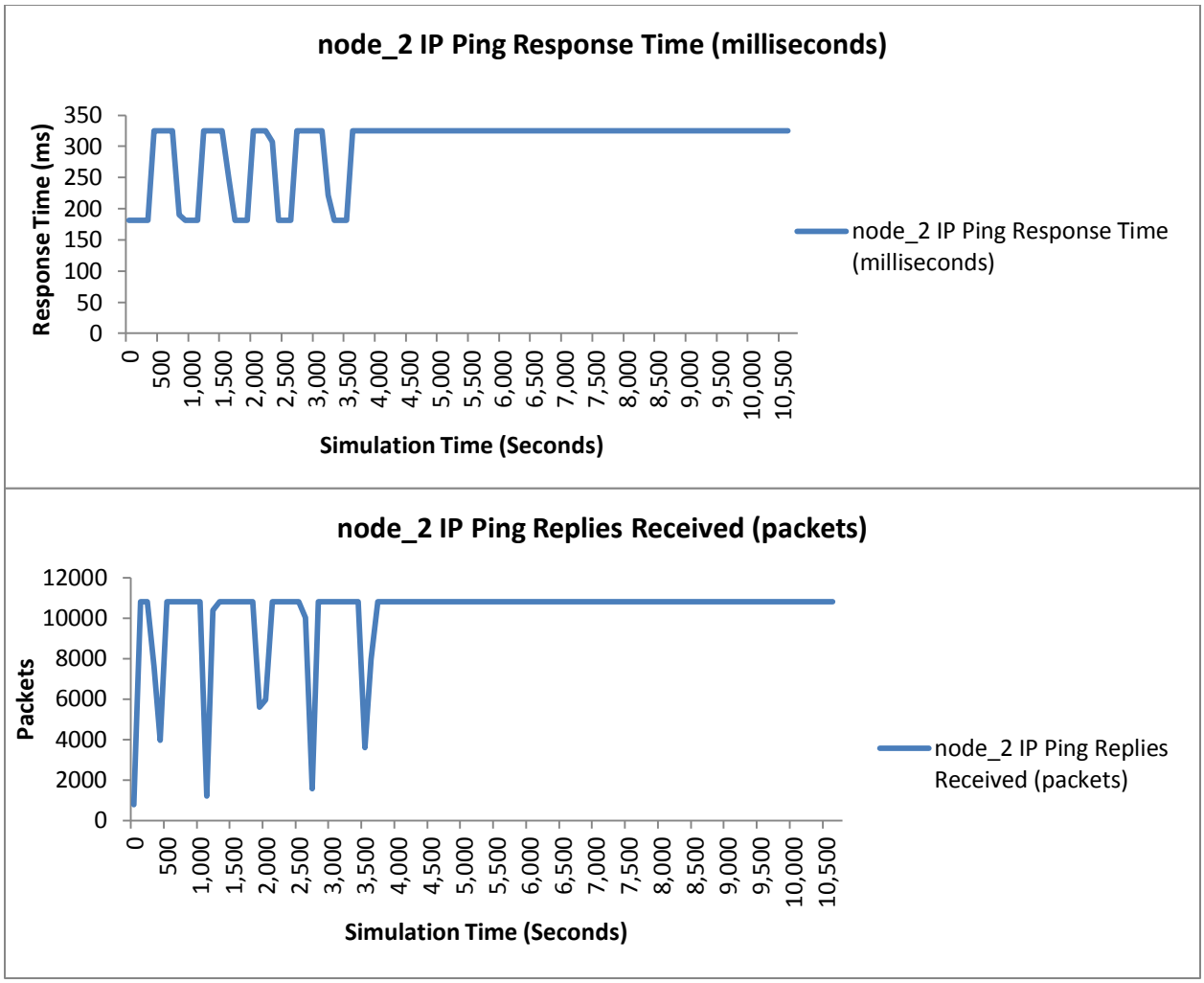


Figure 25 Ping Replies Received and Response Time (Simulation 2)

Furthermore, we can see that traffic is being dropped at time of convergence because AS7 is forwarding traffic to AS5 which does not have a route to the destination. However AS7 didn't receive an update from AS5 yet. For Example, at the first convergence at 400-600 seconds, around 0.21 (x100000) are being dropped in the overall network. When AS7 receives that update, it shifts traffic to AS8. After 3700 seconds, the network stabilized on using AS8 due to the failure and no IP packets were dropped after that.

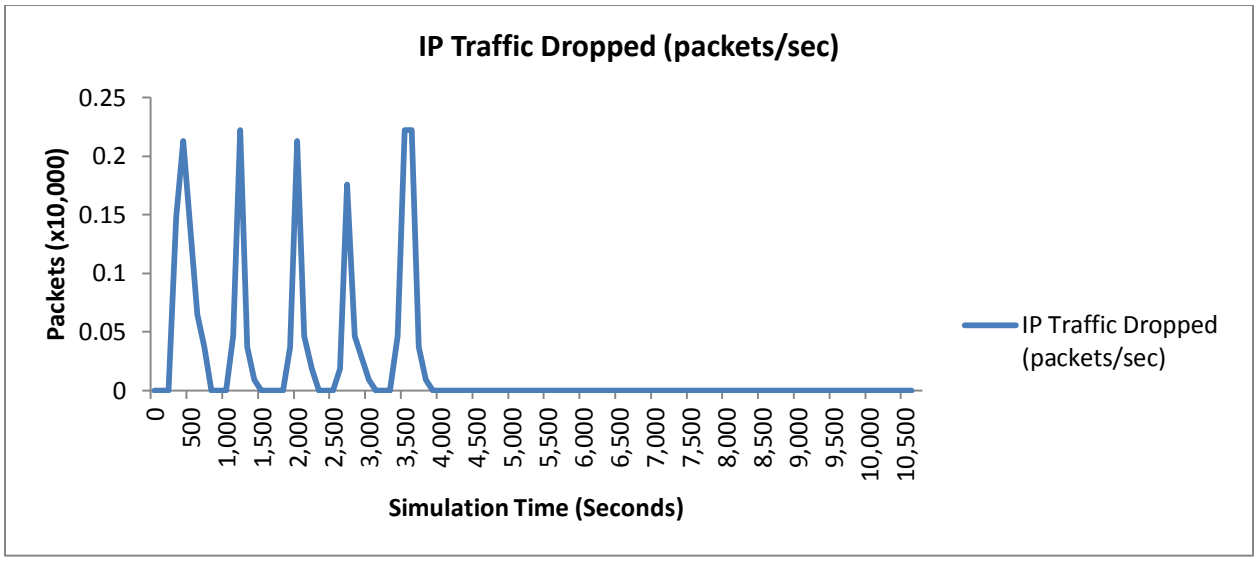


Figure 26 IP Packet Drops (Simulation 2)

Chapter Four

Problem Description and Simulation

In the previous chapter we explain the problem that currently exists in the implemented BGP standard and illustrate it through simulation. As mentioned before, several solutions were proposed to solve the problem but unfortunately none of them can effectively solve the problem without side effects. This chapter focuses on showing AS path aggregation as a method to reduce BGP Churn and reduce convergence time. We focus on presenting this method through simulations that show the shortcomings we found in this solution. In the next chapter we propose our algorithm that solves BGP churn problem in two steps:

- Path Selection Problem at the Aggregator.
- AS Path Shortening Problem.

4.1 AS Path Selection Problem

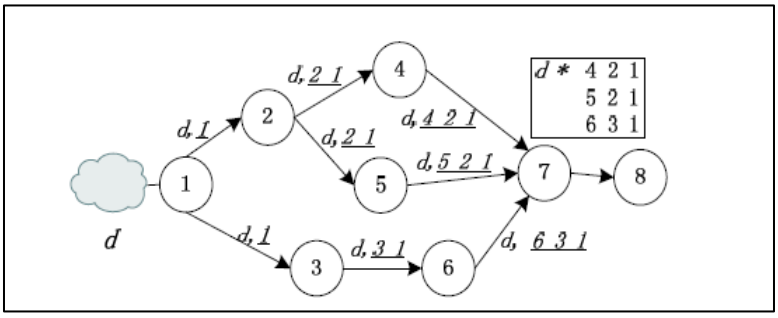


Figure 27 Test Topology from [9]

Consider the topology in Figure 27, which is reproduced from [9], the link between AS1 and AS2 flaps resulting in updates and withdrawals sent to AS7 which is CAGG enabled. Assume AS7 aggregates the paths into 1 {2,3,4,5,6} 7 and send it to AS8 which suppresses unnecessary updates. But what happens when AS7 receives traffic from AS8 while the link 1-2 is flapping? AS7 is still experiencing path

exploration and that result in forwarding traffic to AS4 when it is available while this might result in packet loss. In this case, AS7 should forward traffic to AS6 which doesn't experience any flapping or has no instability history. Wang et al didn't specify this procedure in their paper [9,26].

If AS7 is still using the standard BGP selection method, then all data forwarded to AS4 and AS5 might be lost due to the link flapping which is shown in our simulation. The aBGP [9, 26] is successful in reducing BGP updates due to the suppression of the unnecessary updates. It is also successful in reducing the convergence time needed to reach a stable network. However, it is unsuccessful in maintaining a low traffic delay or zero packet loss after the cutoff threshold is used. Hence, we need a mechanism that takes into consideration the history of the link stability at times of instability.

We show in our simulation how all desired outcomes are achieved using AS path aggregation but data traffic is still being affected as shown in application statistics. We change the standard BGP protocol into aBGP following the ideas mentioned earlier. AS7 is implementing aBGP which aggregates AS-PATHs for penalized routes. The details of aBGP programming are mentioned in Appendix B. Note that aBGP is working per prefix per neighbour. However, for simplicity, we are currently experimenting with only one prefix.

4.1.1 AS 7 Convergence statistic (Simulation 3)

In this section, we notice that AS7 is still going through path exploration and this reflects on the BGP convergence activity as shown below. On the other hand, once the route reaches its cut off threshold, updates are suppressed and only one route is advertised with aggregated AS-Path as shown in Figure 29. We also notice that the overall convergence duration in the network is reduced by approximately 30% which is consistent with the results found by Wang et al. The highest duration of convergence is now 5.2 seconds for the overall network compared to around 7.2 in standard BGP as shown in figure 30. However, upstream routers like AS9 only go through convergence until the cut off threshold is reached.

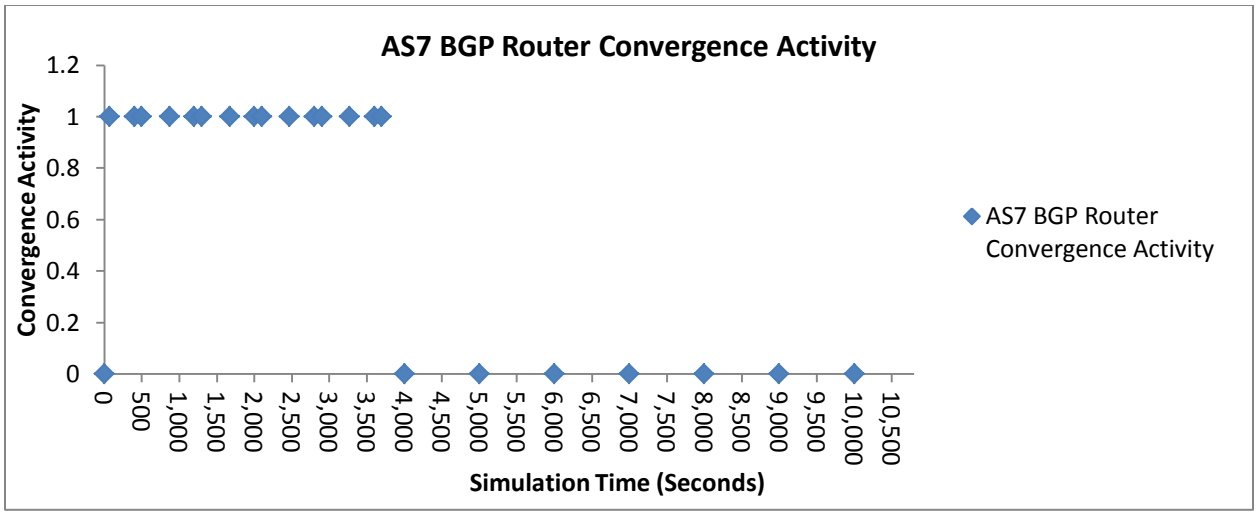


Figure 28 AS7 Convergence Activity (Simulation 3)

It is important to notice that now there are less updates sent by AS7. Amount of received updates are still the same since the convergence occurs downstream from AS7. After the threshold is reached, we can see that 0 updates are propagated by AS7. This is an over 70% reduction of BGP churn compared to simulation 2.

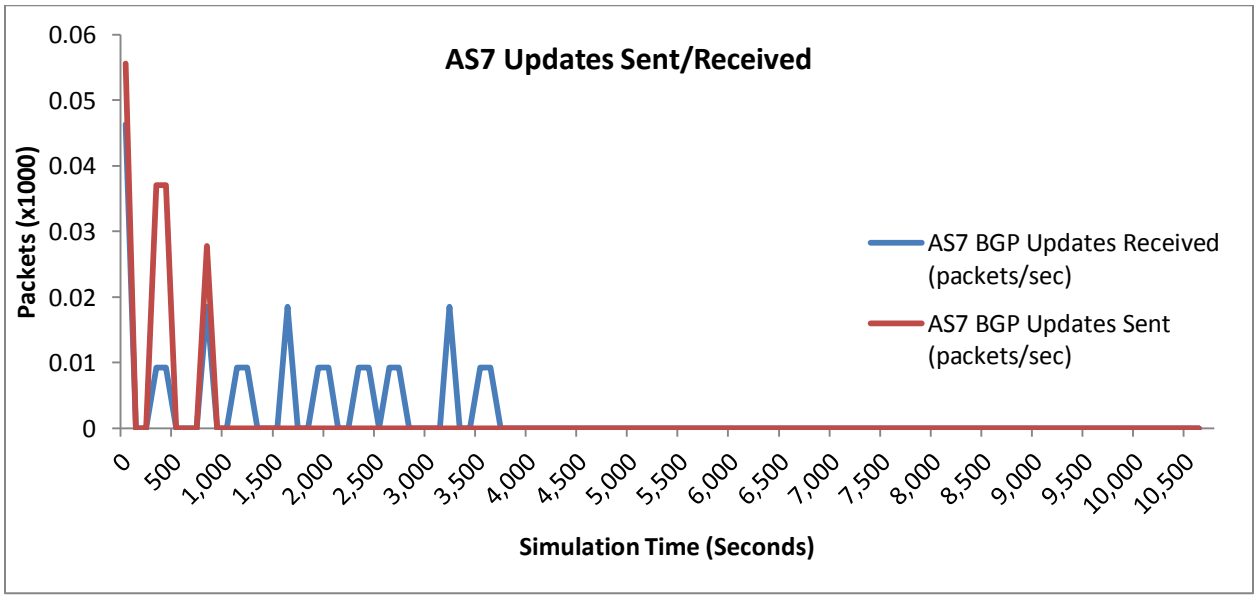


Figure 29 BGP Traffic AS7 (Simulation 3)

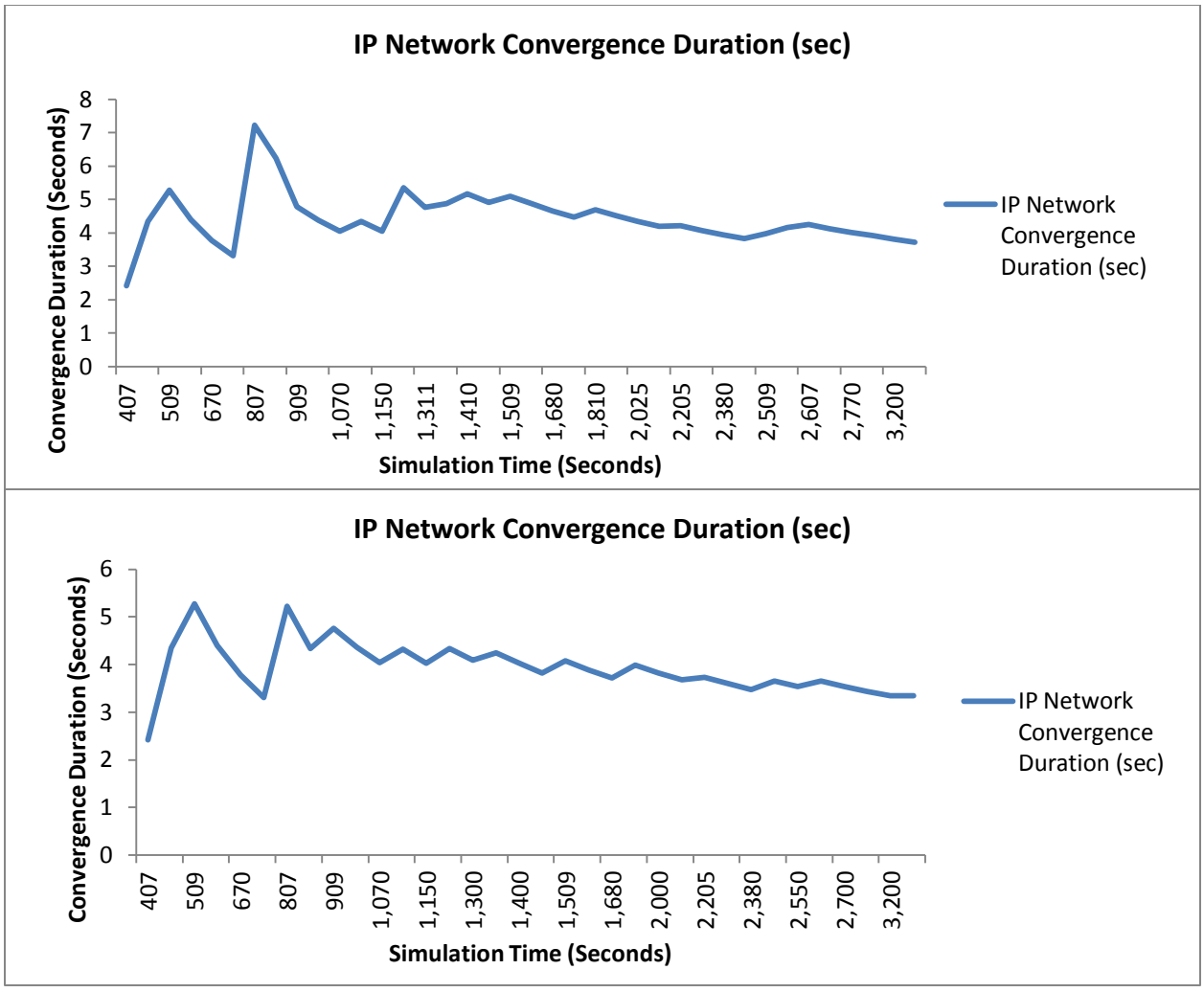


Figure 30 Network Convergence Duration from Simulation 2 above Compared to Simulation 3

4.1.2 AS 9 Convergence statistic (Simulation 3)

aBGP [9] effect can be greatly noticed in upstream routers such as AS9. We observe that network convergence activity is only happening in the period between 400 and 800 seconds where the cut off threshold is achieved at around 800. The first update received at second 70 causes an initial penalty of 1000. Then, the withdraw at second 400 makes the penalty 1500. At 800, the second advertisement pushes it to 2500 which is over the cut off threshold (2000) and CAGG Algorithm is activated. This cause all subsequent updates to be suppressed and greatly reduce BGP churn compared to Simulation 2.

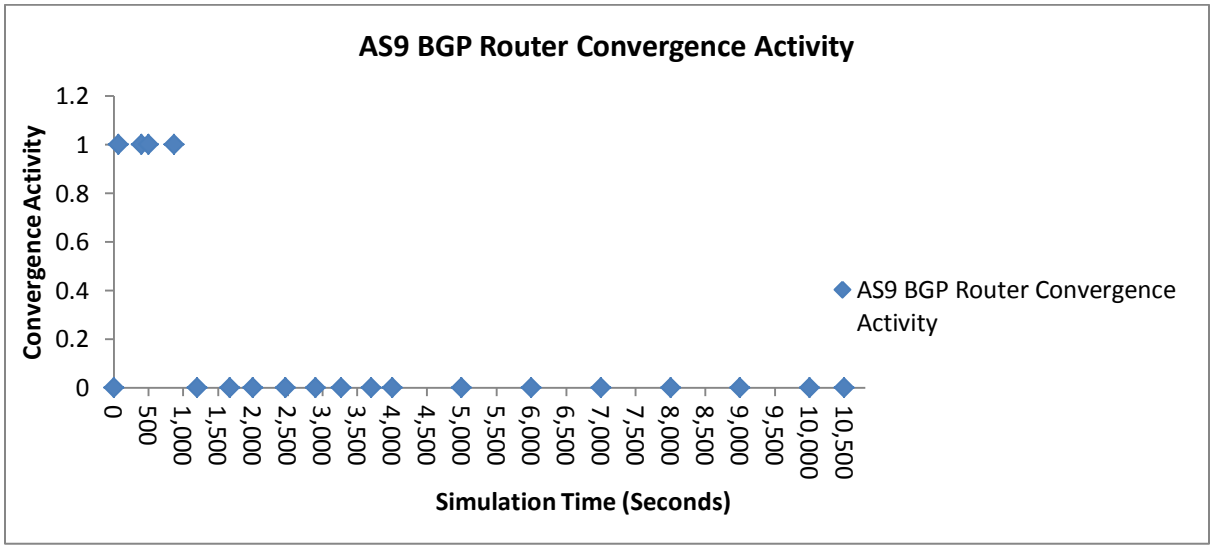


Figure 31 AS9 Convergence Activity (Simulation 3)

Furthermore, BGP updates received by AS9 are greatly reduced due to the suppression of unnecessary updated caused by link flaps. At second 800, the CAGG update is received from AS7 and nothing is received after that.

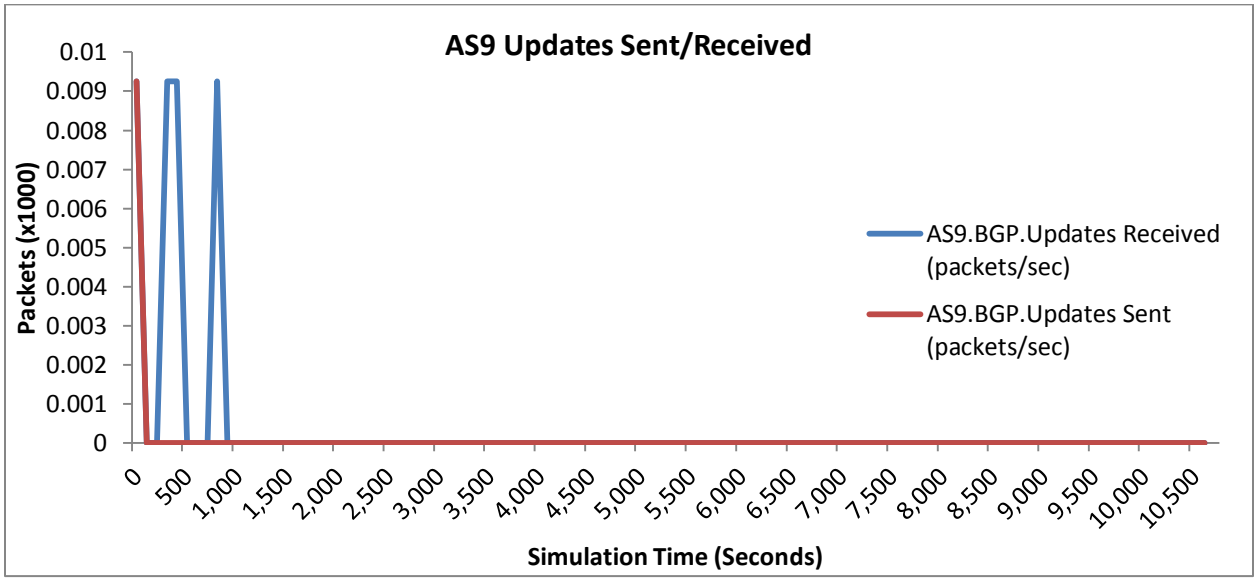


Figure 32 BGP Traffic AS9 (Simulation 3)

4.1.3 Application Statistics (Simulation 3)

Although we have seen a great improvement to BGP convergence and updates, we can only see a slight improvement to the delay and response time of Email Download because AS9 is not converging and installing new routes. Traffic is forwarded immediately to AS7 where a portion of packets is dropped when path exploration happens. Meanwhile, AS7 is still going through path exploration with downstream routers demonstrating the problem described earlier. Amount of traffic sent/received is the same as in the previous case where we still see a drop in traffic even after the network has converged at second 800.

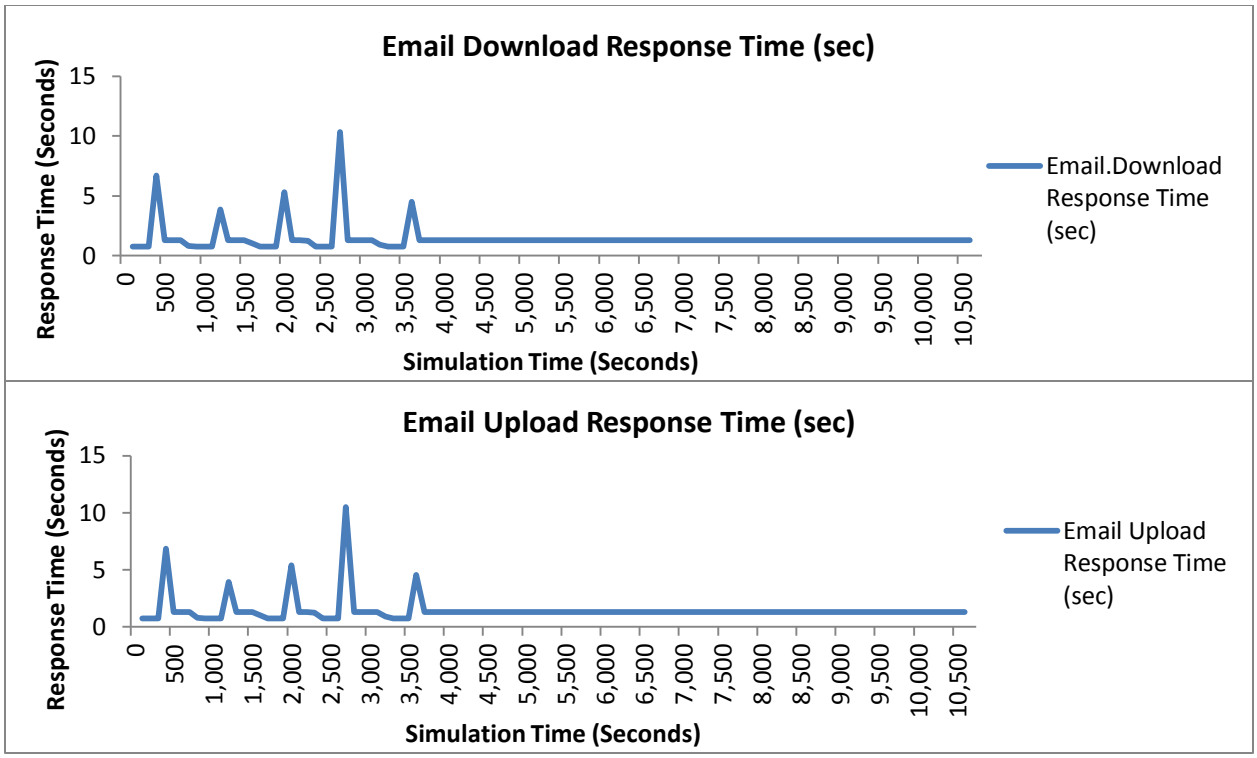


Figure 33 Email Download/Upload Response Time (Simulation 3)

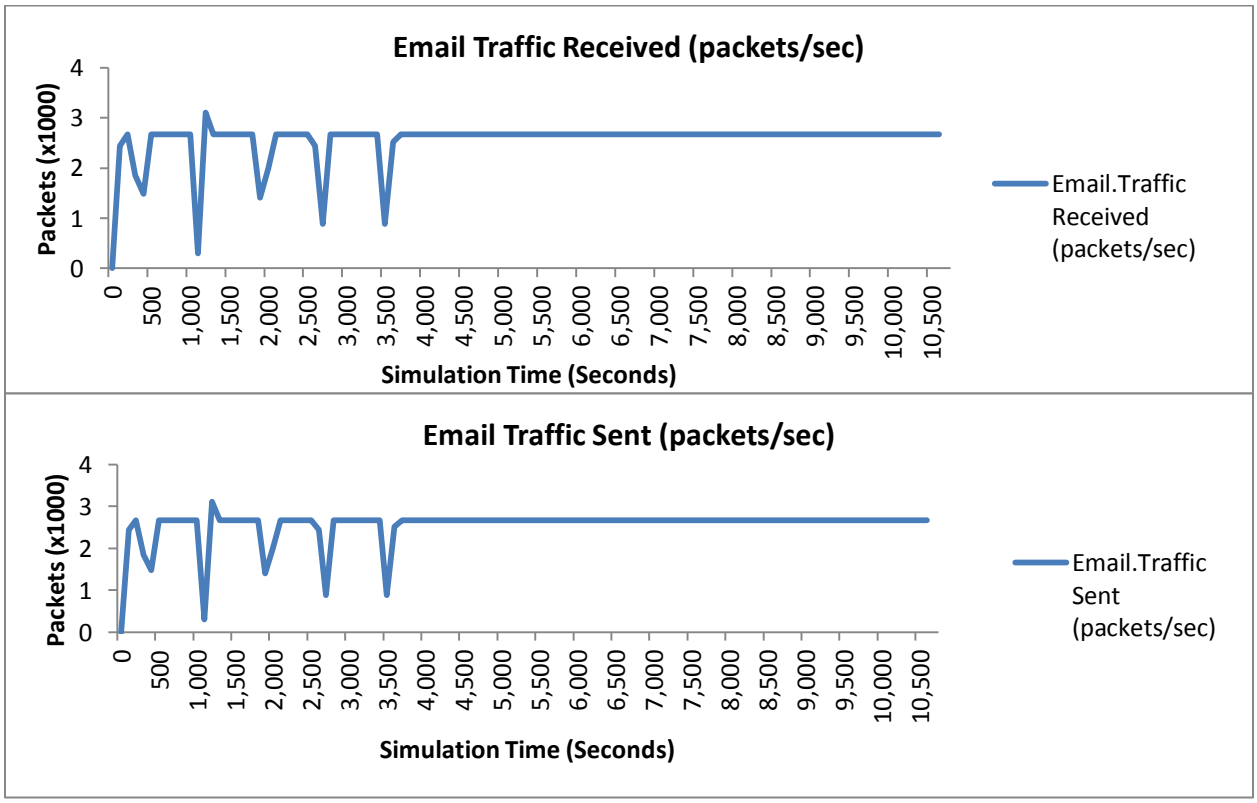


Figure 34 Email Traffic Sent/Received (Simulation 3)

After the cut off threshold is reached at second 800, ping replies are still dropping due to the reasons mentioned earlier. However, the response time is exactly the same as in the last scenario. aBGP could not achieve an improvement to the ICMP traffic in this case.

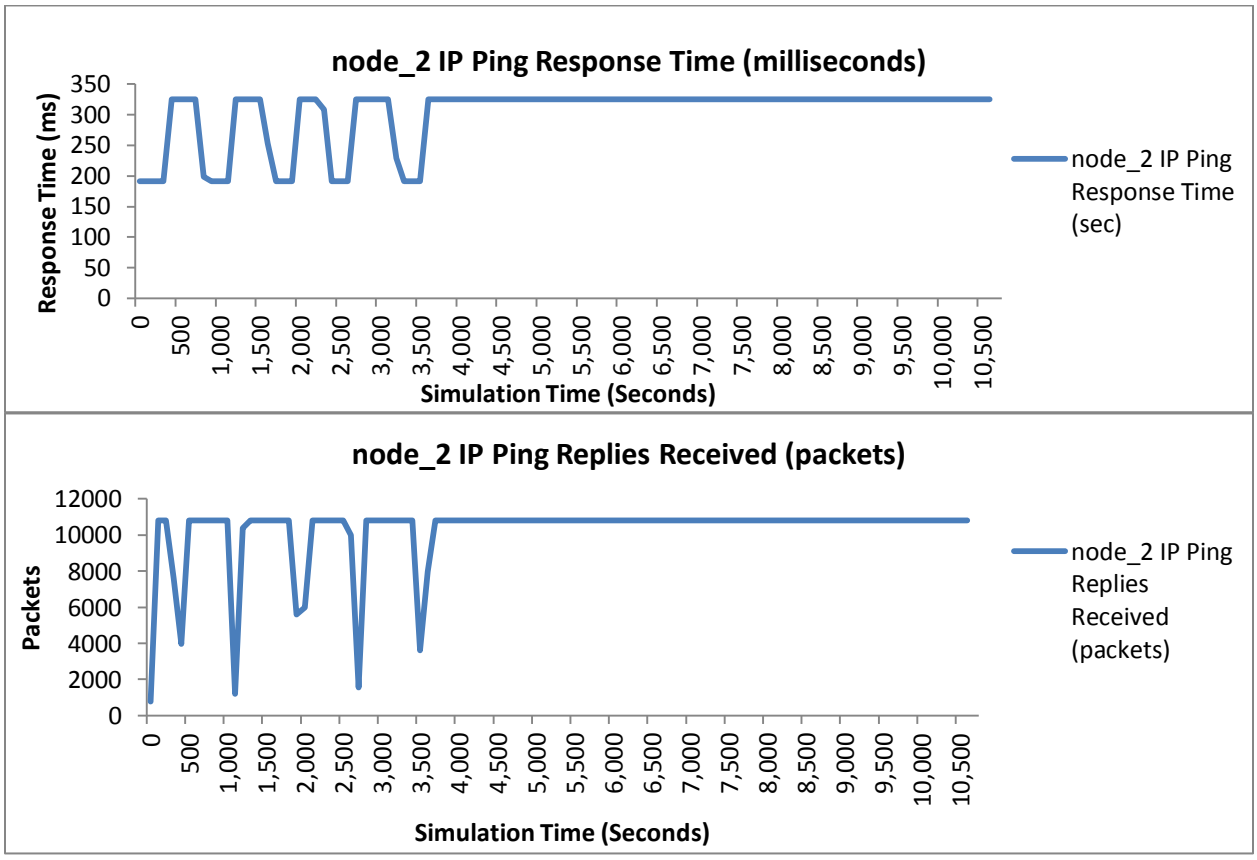


Figure 35 Ping Replies Received and Response Time (Simulation 3)

Also, figure 36 shows the overall traffic drop throughout the entire network. Initially, during convergence more traffic is dropped but once the CAGG threshold is reached, less traffic is dropped; 11000 in this scenario versus around 21000 packets in the previous scenario. This is due to stabilization of AS9 at the 800th second. The emphasis here is that, although the network is stabilized after the threshold, drops still exist due to the convergence downstream from AS7.

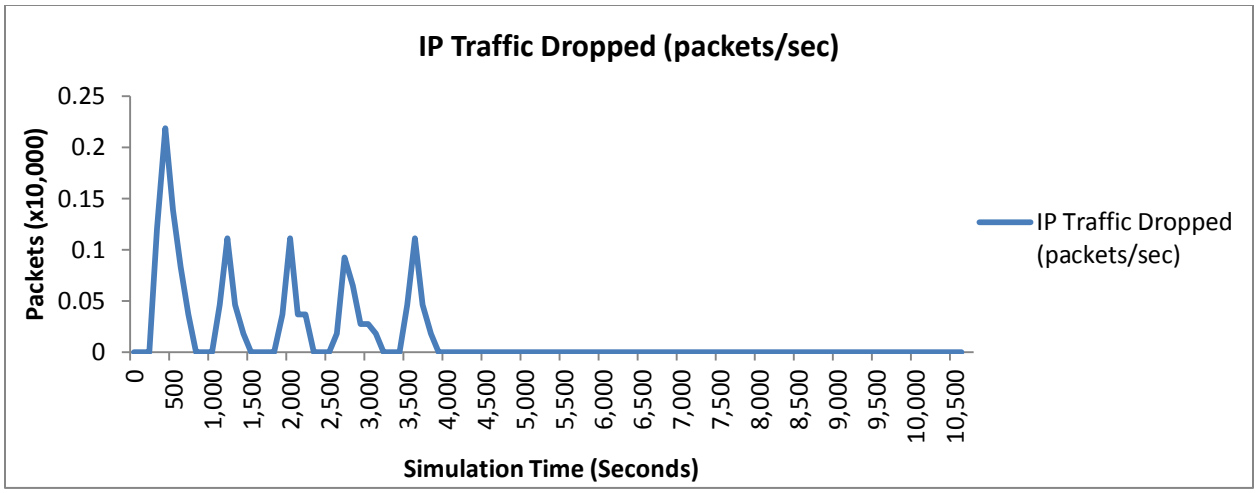


Figure 36 IP Packet Drops (Simulation 3)

4.2 AS Path Shortening Problem

The problem of AS path shortening is actually a side effect of the process of aggregation. As mentioned in [9], "Another feature of function agg worthy to be mentioned is that the output path's AS PATH length is assured to be shorter than any of the member paths involved in this aggregation. In fact, this is common for an aggregation". Furthermore, after investigating many examples of AS path aggregation, we came to the conclusion that the AS path is shorter than any of the original paths except for some extreme cases where the path is longer or of the same length. Compared to prefix aggregation, which results in a shorter mask but less preferable route, CAGG results in a prefix with shorter AS path that is more preferable compared to other available routes. For example, using our test network, the original path to the server from AS9 is 7,4,2,1 which is of length four. After path aggregation, it becomes 7 {2,3,4,5,6,8} 1 which is of length 3. This might have an impact on the BGP decision process for an upstream AS and other ASes start directing traffic towards the aggregator AS, which encounters path exploration or damping downstream.

In our algorithm we consider making the aggregated path less attractive to other ASs. Since the resulting aggregated path is shorter than any of the individual paths, we need a mechanism to make this path longer. Wang et al did not mention any problems related to this issue or any mechanism to ensure that upstream ASes will not choose the aggregated path while another stable path is available.

Our simulation focuses on studying traffic behavior after the AS path aggregation. In the following figure, we introduce an extra path to AS9 that receives two paths of the same attributes from two different eBGP neighbours but it only chooses one.

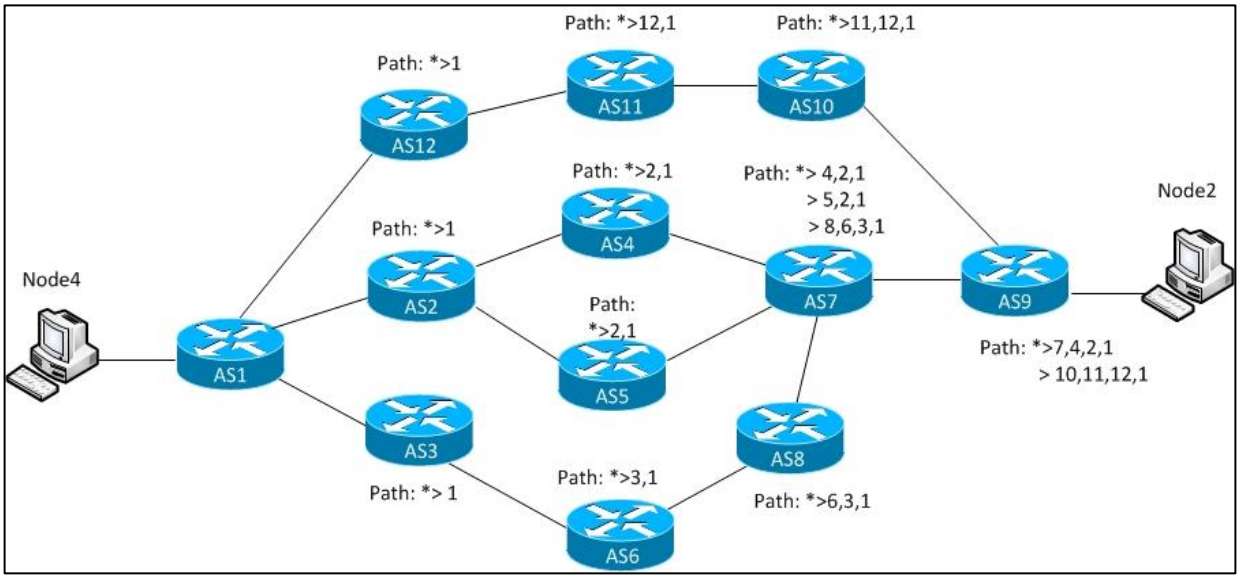


Figure 37 Modified Test Network for Simulation 4

We take into consideration some scenarios to demonstrate the existence of the problem described earlier:

- AS9 initially chooses the path through AS7 rather than AS10 because it receives this first (refer to BGP selection algorithm [19]).
- AS9 initially choose the path through AS10 rather than AS7 because it was received first.
- AS9 requests AS10 to lower the MED to make it a preferable route.

4.2.1 AS Path Shortening Problem – Scenario 1

In this scenario we experience the problem when AS9 initially chooses the path through AS7 rather than AS10 because it is received first. In this case, we notice that the network initializes on using the path through AS7. The first withdraw happens around second 400 and then AS9 chooses the path through AS10 instead; based on the BGP selection algorithm. Then, a new advertisement is sent at second 800 pushing the cut off limit to 2500 and causing AS7 to send the CAGG route to AS9. The new received route is 3 paths long : 7, {4,5,2,8,6,3}, 1.

This path is instantly preferred over the existing stable path 10,11,12,1. Of course, all traffic starts to go through AS7 experiencing the delay and drops explained in simulation 3. We have set up a simulation with this scenario and focused on link utilization which is a good metric to understand traffic behavior in OPNET. The following statistics show the actual utilization of links that shows this behavior.

Looking at Figures 38 and 39, we see how the link AS7-AS9 is utilized at the beginning of the simulation and at 400th second the link is not used and the traffic is shifted to AS10. At 800th second, the new CAGG route is advertised and link AS9-AS7 is used for the rest of the simulation. Remember that by the end of the flap at 3700th, AS7 continues advertising the CAGG route due to the accumulation of penalties and still uses the path through AS8. Hence, all traffic goes through AS7 and then AS8 which is the longest path possible.

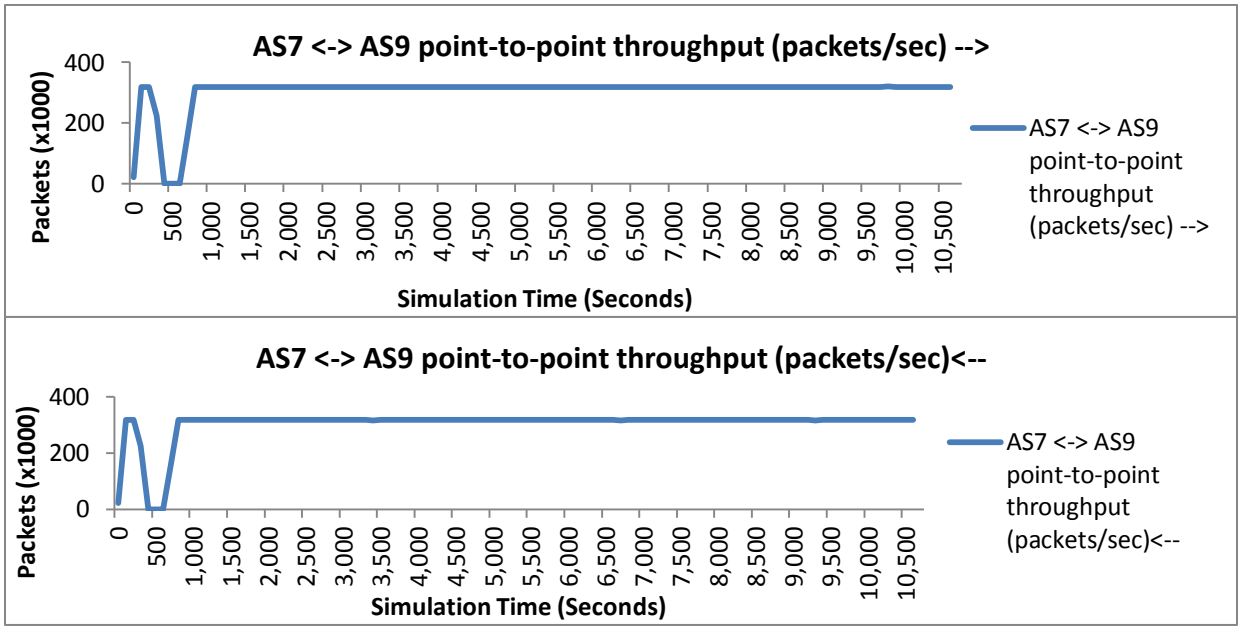


Figure 38 AS9-AS7 Link Utilization - AS Path Shortening Scenario 1 (Simulation 4)

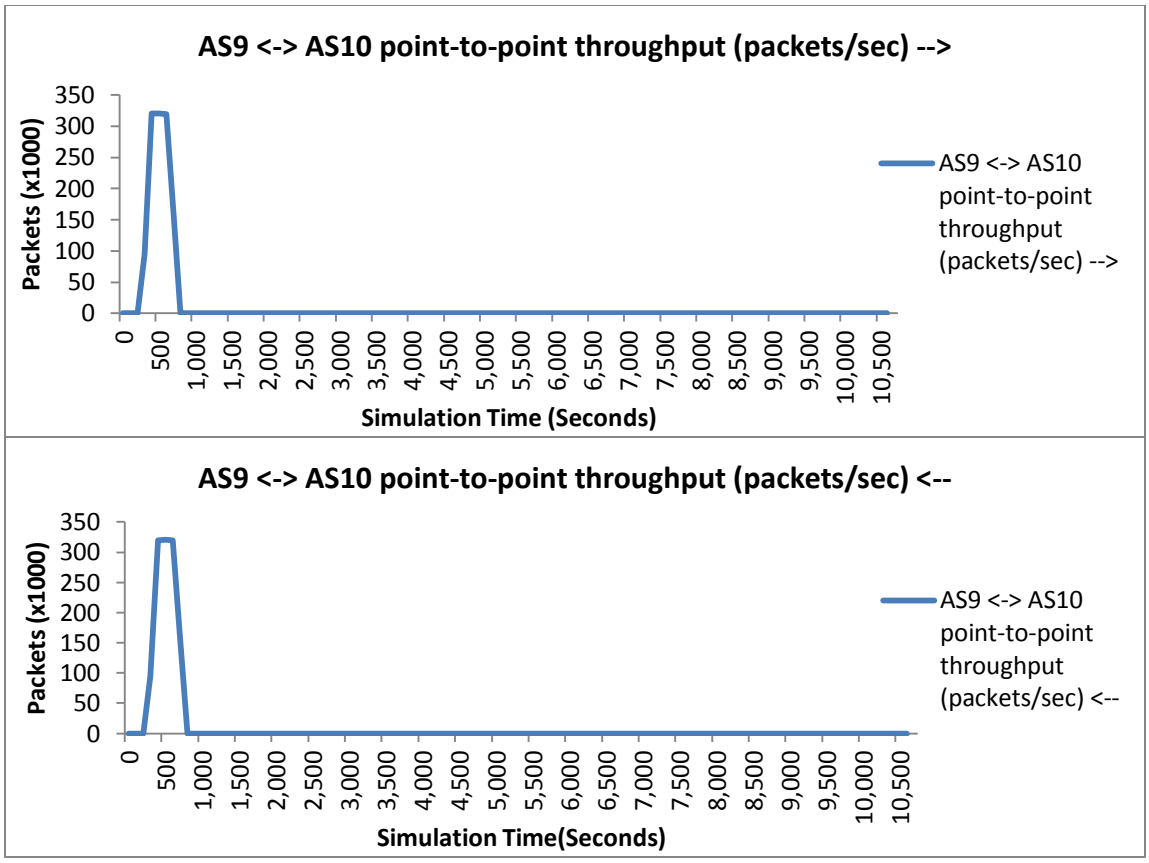


Figure 39 AS9-AS10 Link Utilization - AS Path Shortening Scenario 1 (Simulation 4)

4.2.2 AS Path Shortening Problem – Scenario 2

In this scenario, we investigate the situation where AS9 initially chooses the path through AS10 rather than AS7 because it is received first or because AS9 requests AS10 to lower the MED Attribute to make it a preferable route due to commercial agreement or any other reason. In this scenario, AS9 begins with choosing AS10 as a preferred route. At second 400, the first withdraw and update is received from AS7, however, this does not affect the traffic because AS9 still prefers AS10 (since it is the oldest route in the routing table [19]). At this point, everything is stable and traffic continues traversing the desired path through AS10. After second 800, AS7 sends a shorter path due to the AS path Aggregation. AS9 changes its path to AS7 because even if MED is used, AS-path length has precedence over MED attribute.

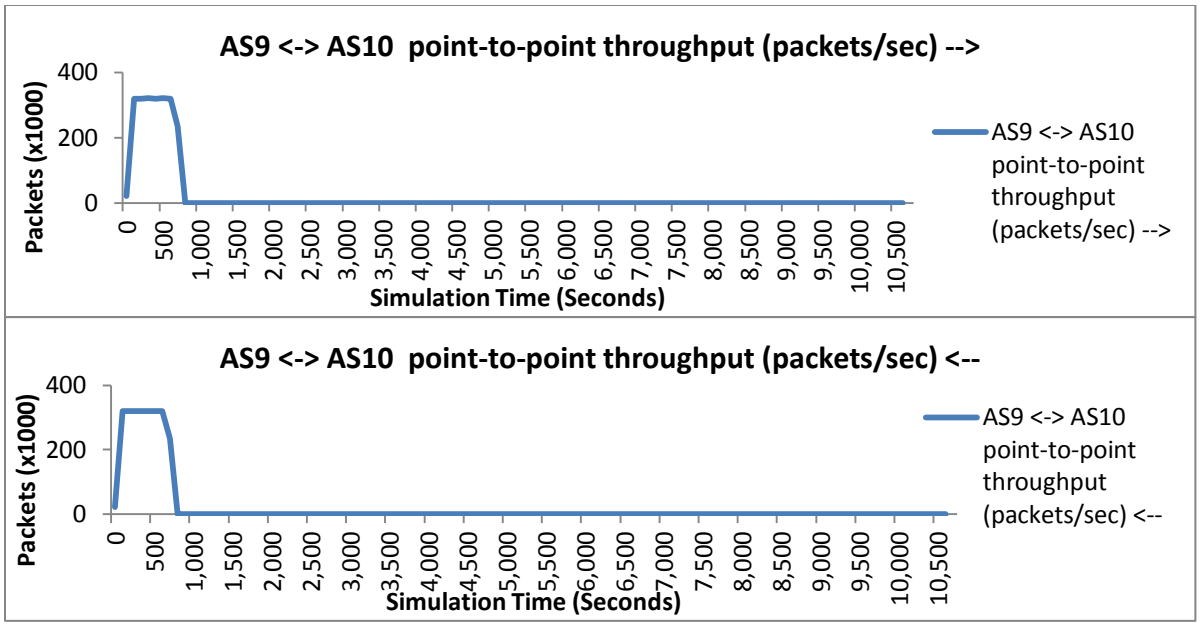


Figure 40 AS9-AS10 Link Utilization - AS Path Shortening Scenario 2 (Simulation 4)

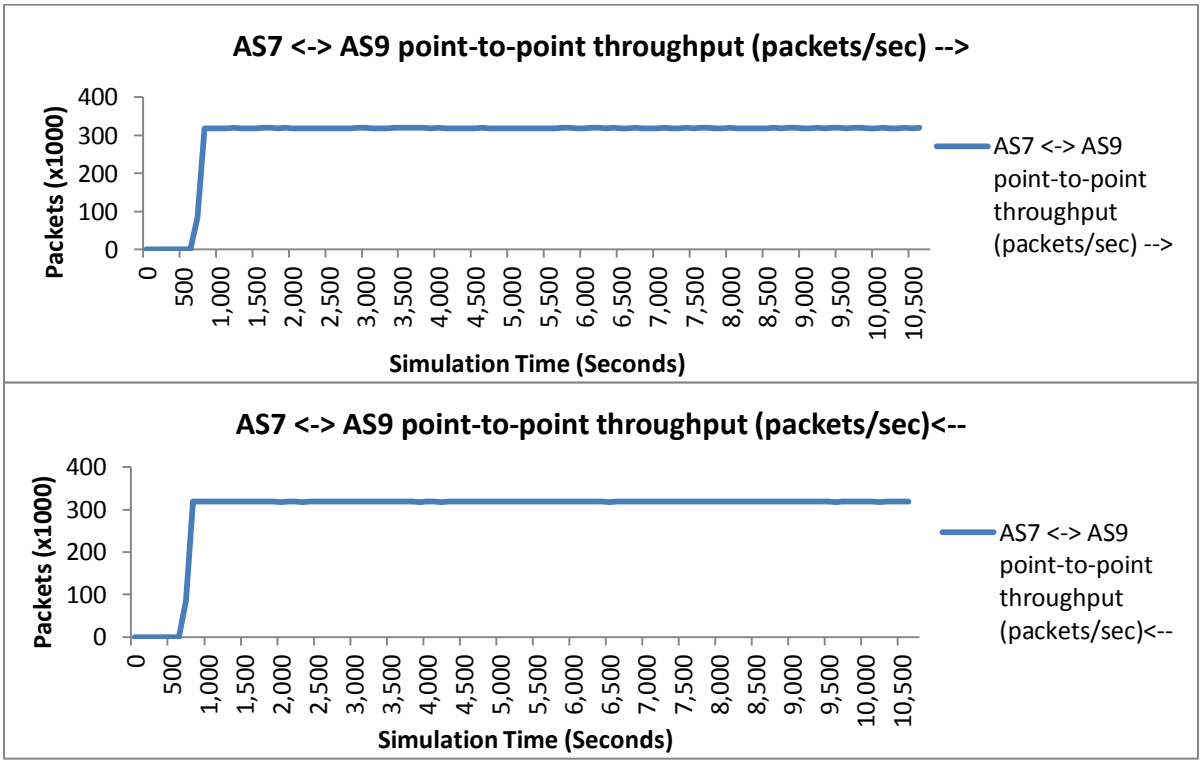


Figure 41 AS9-AS7 Link Utilization - AS Path Shortening Scenario 2 (Simulation 4)

BGP selection algorithm tries to avoid link flaps by using the oldest route in the BGP table when a tie occurs. However, we have seen some cases where AS path aggregation is forcing ASes to use unwanted paths or paths with a history of flaps and path exploration. Previous simulations in both scenarios gave similar results to simulation 3 from an application point of view. Hence, in order to reach a desirable outcome, we need to change the path selection algorithm at the aggregator and change the way CAGG constructs and aggregates path. In the next chapter, we propose our algorithm for solving both problems, describe its implementation in OPNET, and discuss results that show a great improvement to application behavior while preserving the achieved results of network convergence.

Chapter Five

Stable Path Aggregation (SPAGG)

In this chapter, we propose our solution: Stable Path Aggregation (SPAGG) Algorithm. We focus in our algorithm on preserving the achieved outcomes incurred by the original CAGG approach while changing key aspects to achieve better reachability results. An Internet draft identified two stages of convergence for BGP, Optimality and Reachability. It also classified reachability as more important to reach than optimality [41]. We saw how the work in [9] could reach reachability as fast as possible, but could not maintain this reachability for certain periods of time because of not focusing on optimality. We propose two changes to the existing CAGG [9] algorithm and describe our proposed algorithm in details. We also explain how the proposed algorithm is implemented using C language in OPNET Modeler. We use the exact same test network described earlier but with our new SPAGG BGP. We also examine simulation results and compare them to results obtained in simulation 3 and 4. We clearly see a great improvement in the reachability and optimality of the network while achieving satisfactory BGP convergence statistics which are better than any convergence results achieved by recent proposals like PED [8].

5.1 Stable Path Aggregation (SPAGG) Algorithm

We divide our algorithm into two parts, each with its own computation and workflow. However, both solutions interconnect from an implementation point of view. We describe each solution in the same order of problems described earlier. SPAGG consists of two components:

- Stable Path Selection.
- AS path prepending.

5.1.1 Stable Path Selection

We designed an extension to the CAGG algorithm with the purpose of choosing the most stable path available after aggregating AS paths. This algorithm is triggered each time the CAGG algorithm is invoked in order to avoid disruption and delay in traffic as it is demonstrated earlier in Simulation 3. Brighten Godfrey et al proposed a new approach for route selection in [10]. We are inspired by their research outcomes that led us to create our modified version of Stable Path Aggregation algorithm. The scheme in [10] defines three metrics of trade-off that should be taken into consideration when trying to reach a stable path selection. Those metrics are:

- Interruption, which is the event that denotes the path, selected by some AS changes or is withdrawn entirely (i.e., a transition to the disconnected state).
- Availability, which is for a particular source-destination pair defined as the fraction of time that the source has a route to the destination.
- Deviation, which compares a sequence of selected paths against the best path sequence, and is defined as the fraction of time that the route matches the sequence in the best path.

The aim of our algorithm is to provide the highest level of availability, while trying to keep deviation in an acceptable range. Also, interruption is handled by path aggregation in this case. The following is a list of definitions of SPAGG components. Let:

- P_d = {a_1, a_2, a_3 ….. a_n}, be the path vector for destination d.
- P_d^1 = {a_1, a_2, a_3 ….. a_n}, be the first P_d.
- P_d^2 = {a_1, a_2, a_3 ….. a_n}, be the second P_d.
- τ_{P_d}, be the time elapsed since P_d is inserted in the BGP table.

- comp(P_d^1, P_d^2), be a function that compares P_d^1 and P_d^2 against the standard BGP selection process.
- $\tau_{P_d^1}$, be the time elapsed since P_d^1 is inserted in the BGP table.
- $\tau_{P_d^2}$, be the time elapsed since P_d^2 is inserted in the BGP table.
- ω, be the acceptable time that P_d should spend in the BGP table.
- P_d^N = {a_1, a_2, a_3 ….. a_n}, be the P_d received from neighbour N.
- P'^N_d = {$á_1$, $á_2$, $á_3$ ….. $á_n$}, be the new aggregated path vector.
- $l = |P_d^N|$, be the length of the path vector.
- $l' = |P'^N_d|$, be the length of the new aggregated path vector.

Our algorithm, adds an extra condition for path selection, where τ_{P_d} of the selected path should be equal to or larger than ω. We leave ω to be a configurable parameter that affects the amount of deviation acceptable for the service provider. The algorithm is described as the following:

Input: P_d^1, P_d^2, $\tau_{P_d^1}$, $\tau_{P_d^2}$, ω

Output: P_d , which is chosen path after processing.

if comp(P_d^1, P_d^2) = P_d^1 **AND** $\tau_{P_d^1} \geq \omega$

THEN Choose P_d^1

else

if comp(P_d^1, P_d^2) = P_d^2 **AND** $\tau_{P_d^2} \geq \omega$

THEN Choose P_d^2

else

if $\tau_{P_d^1} > \tau_{P_d^2}$

THEN Choose P_d^1

else

if $\tau_{P_d^2} > \tau_{P_d^1}$

THEN Choose P_d^2

else

Choose the preferred path based on standard BGP selection algorithm

Return P_d

Figure 42 SPAGG - Stable Path Selection Algorithm

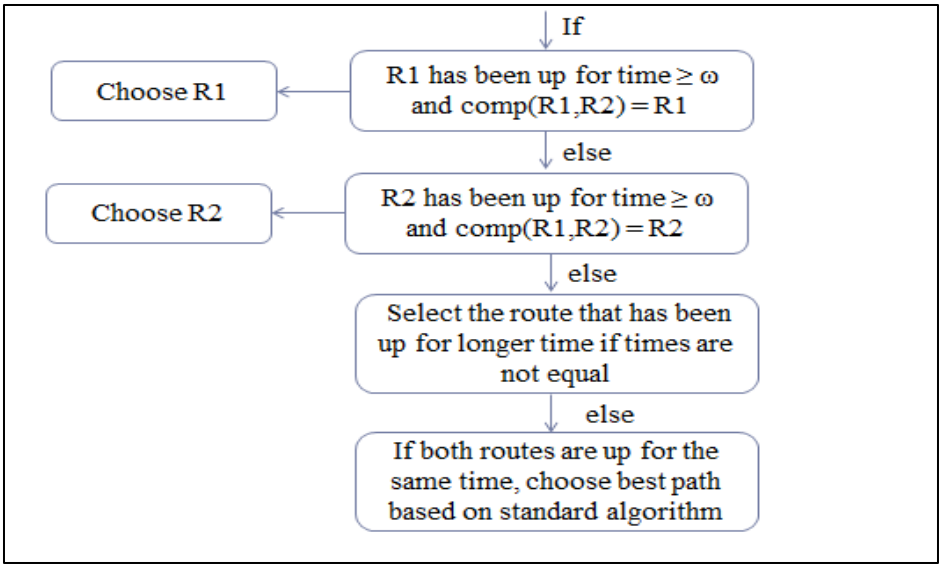


Figure 43 SPAGG - Stable Path Selection Workflow

Our algorithm processes all available pairs in the BGP table. The first two steps try to choose the most preferable path based on the standard selection process and that satisfy the stability condition. If this does not work, we choose the path that has been for the longest time in the BGP table. This approach is actually implemented currently in Cisco routers as one of the steps of [19] in order to minimize the effect of link flaps on router and network convergence. Also, the selection algorithm mentioned in [10] assumes "that recently advertised routes are more likely to be withdrawn soon". Our algorithm gives an implicit importance to route penalty since penalized routes are flapping and this indicated their short age in the BGP table which we try to avoid. Looking at our test network in figure 9, we apply our algorithm at AS7 which is the aggregator of AS paths. The problem described in section 4.1 is tested again with our algorithm. After the cut off threshold is reached, AS7 advertises the new path with aggregated ASes and triggers our selection algorithm immediately. AS7 has three potential paths to the destination:

- A route through AS4, which is experiencing advertisement and withdrawal.
- A route through AS5, which is experiencing advertisement and withdrawal
- A stable route through AS8, which is installed at the beginning of the simulation.

At the 800th second, the threshold is reached by receiving an advertisement from AS4 and AS5 when both have a Υ_{P_d} of almost 0 to 1 second. AS7 compares both routes to the third route through AS8 and selects it due to its stability. All traffic is immediately forwarded through AS8, and although flapping is still

present downstream, AS7 does not go through path exploration as long as the penalty is over the half-life threshold. This process results in a 100% improvement in traffic delay and packet drops compared to simulation 3 as we see later in this chapter. Once the penalty for destination d is below half-life threshold, aggregation algorithm stops advertising aggregated AS paths and our selection algorithm ceases from enforcing our selection criteria.

5.1.2 AS path prepending

We saw in the previous chapter how AS path shortening could affect the path selection decision process for upstream routers. Our solution aims to maintain path attributes that are constructed before aggregation. The only attribute that changes at the time of aggregation is the AS path length. Hence, we modified the current AS path aggregation algorithm to incorporate the mechanism of AS path prepending. We defined $l = |P_d^N|$ to be the AS path length of P_d^N. At time of convergence and after the cut off threshold is reached, we construct $P'^N_d = \{á_1, á_2, á_3 \ldots\ldots á_n\}$. If l' is smaller than l, we prepend P'^N_d with the necessary instances of $á_1$ to make $l' = l$. This results in an aggregated path of equal length to the last best known AS path which does not affect the decision process of upstream routers.

If an upstream router uses a more preferred attribute than path length to control traffic, then the BGP selection process is not affected by the Prepend procedure. Also, if it uses AS path length or a lower preference attribute like MED, then the decision process is not affected as well, and all traffic is directed to the same neighbour before aggregation. The following is the algorithm used when the decision of aggregating a path is taken:

Input: P_d^N, l, $á_1$
Output: P'^N_d, l'
if $l = l'$
    End
else
  if $l > l'$
    THEN prepend P'^N_d with $(l - l')$ instances of $á_1$
Return P'^N_d, l'

Figure 44 SPAGG - AS Path Prepending Algorithm

Hence, the original aggregation workflow is changed to the following:

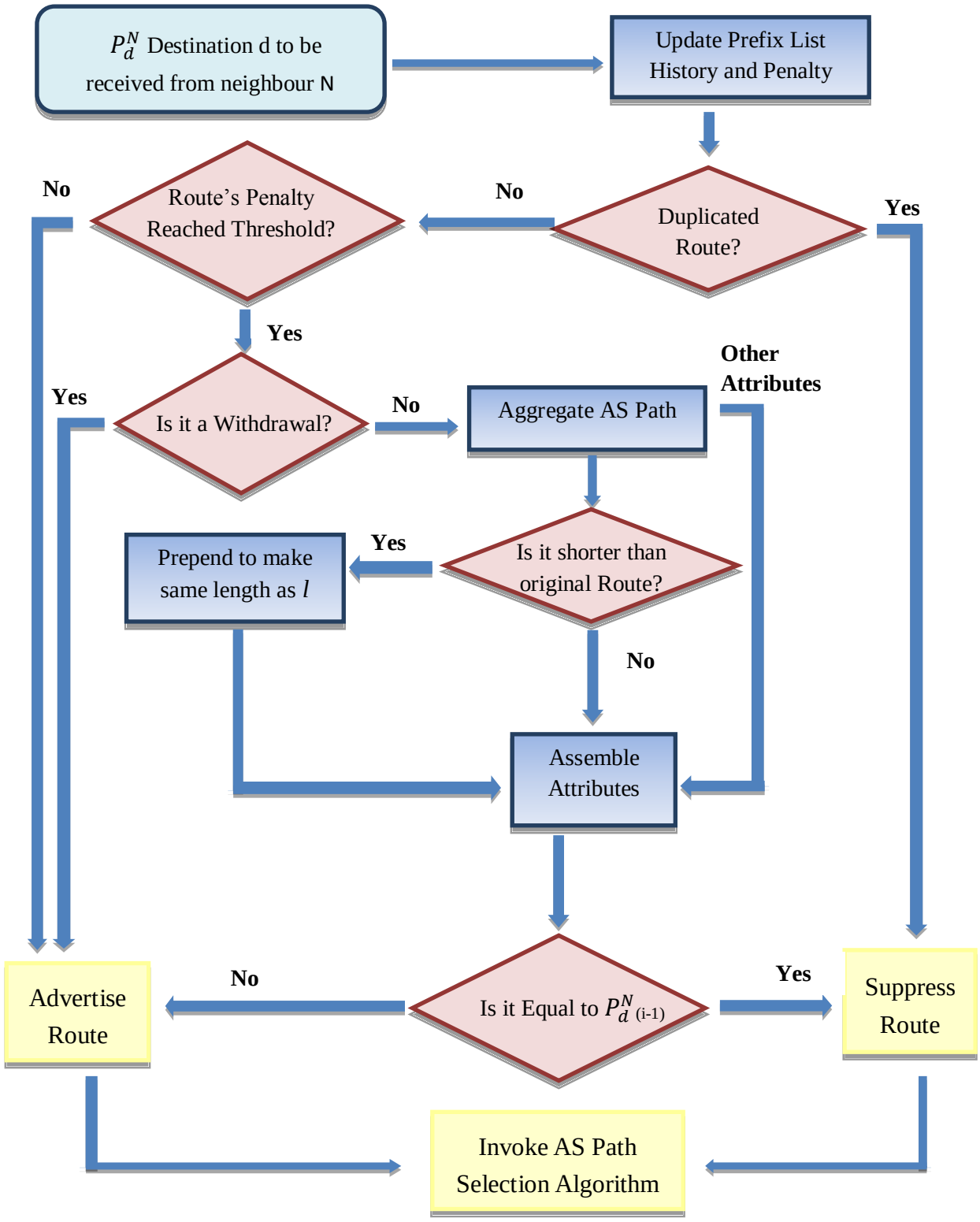


Figure 45 Stable Path Aggregation (SPAGG) Workflow

If we apply the above logic to our test network depicted in figure 37, both scenarios are solved by avoiding the flapping path because of its length. For example, in scenario 1 we found that AS9 is using the path through AS7 whenever the aggregation produces a shorter path. Instead of receiving the path 7, {4,5,2,8,6,3}, 1 which is of length 3, AS9 receives a prepended path of length 4, which is 7, 7, {4,5,2,8,6,3}, 1. The new length is equal to the best last known AS path length, which triggered the aggregation algorithm. The idea behind our algorithm is not to affect the decision process of any AS except when the AS itself has taken some steps to enforce using the flapped link, like using the local preference attribute or the weight attribute in Cisco routers. We see in the results section how all traffic uses the path through AS10. This is because it is of length 4 while the aggregated path is also of length 4, but, the path through AS10 is older in the BGP table due to its stability. Based on the standard selection algorithm [19], when two paths are equal in attributes, routers choose the oldest one.

In the second scenario, the path through AS10 is initially used because it is received first or because it has a lower MED value. Although flapping happens in the path through AS7, AS9 is not affected at all. When the threshold is reached, the aggregated path 7, 7, {4,5,2,8,6,3}, 1 is sent to AS9. The new aggregated path is of length 4. Since all paths have length 4 now, we find that AS9 prefers using AS10 as this path is either older in the BGP table or it has a lower MED. Again, we do not affect the choice of the service provider due to AS paths shortening. It is impossible to cover all scenarios in this thesis, but our algorithm acts as a safety mechanism in any scenario and it is definitely safer than the idea of shortening AS paths.

5.2 SPAGG Implementation

In this section, we discuss how path aggregation is implemented along with our algorithms that are explained above. As mentioned in this document, we implemented our algorithm in OPNET Modeler modules in C language and Proto-C. We show the code workflow and explain codes of some important functions. The process of coding our SPAGG BGP starts by going through thousands of lines of pre-implemented code. Then, we identify areas of modification. Some changes are necessary to the C headers of the current module in OPNET Modeler. Reading chapter 3 is necessary for understanding parts of the code and interaction between functions. We implement changes in two phases: 1) under the main *bgp* process 2) under the *bgp_conn* process. We show the workflow of the main functions modified and a snapshot of some important modifications to attributes and constants we made.

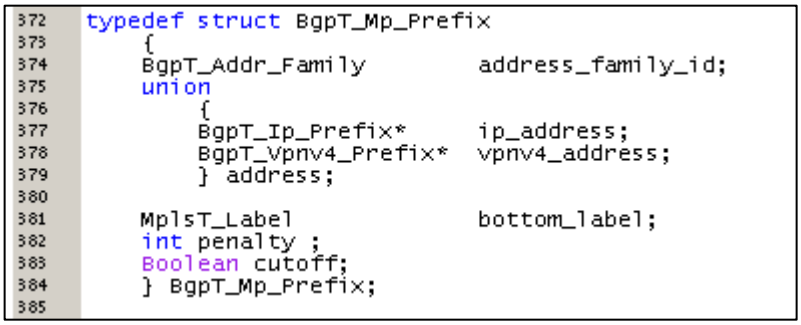
We added some new static variables that help our implementation under both processes. Following is a list of the new variables and their explanation:

Variable	Type	Description
First_time	Boolean	This is to indicate if it's the first time to handle an update message
MY_LIST	List*	History list of all subnets withdrawn or updated
node_id	Objid	This is to identify the node of concern
last_advertised_iminus1	List*	This is a history list of the last advertised CAGG route
last_advertised_i	List*	A list of routes advertised at the moment of Aggregatoin
my_updates_list	List*	A list of all reachability updates received during simulation
my_updates_time	List*	A list to register the time of receiving each reachability route
Best_AS_Path_L	Int	Length of the best path before aggregation

Table 1 Static Variables for SPAGG Implementation

Notice that static variables carry the values assigned to them during the whole simulation. The MY_LIST variable is of great importance for us. It carries all penalized routes in a shared memory space where we can always reference it. In addition to the static variables, we made some important changes to the data structures already implemented. We mention some important changes we made as follows:

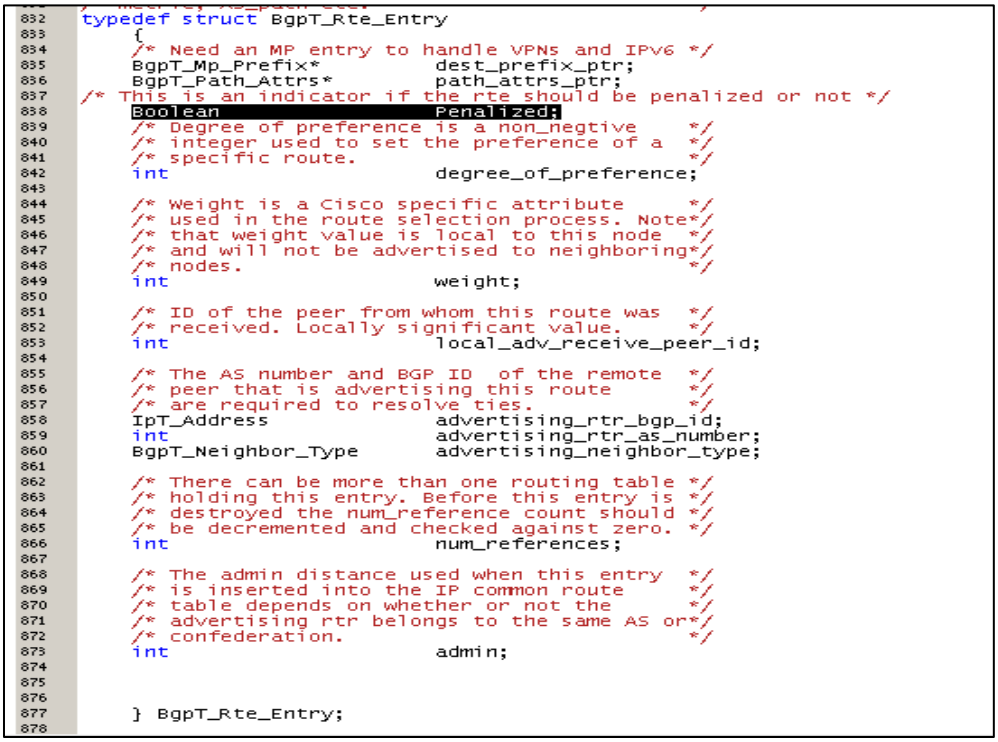
- We edited the structure defined in the header file BGP.h that is used to identify BGP prefixes to include two new variables:
    - An integer representing penalty value for each prefix.
    - A Boolean variable representing if the cut off threshold was reached or not for that prefix.



```
372  typedef struct BgpT_Mp_Prefix
373      {
374      BgpT_Addr_Family        address_family_id;
375      union
376          {
377          BgpT_Ip_Prefix*     ip_address;
378          BgpT_Vpnv4_Prefix*  vpnv4_address;
379          } address;
380
381      MplsT_Label             bottom_label;
382      int penalty ;
383      Boolean cutoff;
384      } BgpT_Mp_Prefix;
385
```

Figure 46 New BGP Prefix Structure

- We edited the structure that is used to define BGP routes (which include all attributes of the path). We added a Boolean variable to indicate if the route is penalized or not based on the cutoff attribute mentioned earlier.



```
832  typedef struct BgpT_Rte_Entry
833      {
834      /* Need an MP entry to handle VPNs and IPv6 */
835      BgpT_Mp_Prefix*         dest_prefix_ptr;
836      BgpT_Path_Attrs*        path_attrs_ptr;
837  /* This is an indicator if the rte should be penalized or not */
838      Boolean                 Penalized;
839      /* Degree of preference is a non_negtive    */
840      /* integer used to set the preference of a  */
841      /* specific route.                          */
842      int                     degree_of_preference;
843
844      /* Weight is a Cisco specific attribute     */
845      /* used in the route selection process. Note*/
846      /* that weight value is local to this node  */
847      /* and will not be advertised to neighboring*/
848      /* nodes.                                   */
849      int                     weight;
850
851      /* ID of the peer from whom this route was  */
852      /* received. Locally significant value.     */
853      int                     local_adv_receive_peer_id;
854
855      /* The AS number and BGP ID  of the remote  */
856      /* peer that is advertising this route      */
857      /* are required to resolve ties.            */
858      IpT_Address             advertising_rtr_bgp_id;
859      int                     advertising_rtr_as_number;
860      BgpT_Neighbor_Type      advertising_neighbor_type;
861
862      /* There can be more than one routing table */
863      /* holding this entry. Before this entry is */
864      /* destroyed the num_reference count should */
865      /* be decremented and checked against zero. */
866      int                     num_references;
867
868      /* The admin distance used when this entry  */
869      /* is inserted into the IP common route     */
870      /* table depends on whether or not the      */
871      /* advertising rtr belongs to the same AS or*/
872      /* confederation.                           */
873      int                     admin;
874
875
876
877      } BgpT_Rte_Entry;
878
```

Figure 47 New BGP Route Structure

- We also edited the external source file named *bgp_support.ex.c* to modify the function used to copy BGP routes. We need this modification for it to carry the penalized variable whenever the route is copied for further processing.

```
BgpT_Rte_Entry*
bgp_support_rte_entry_copy (BgpT_Rte_Entry* orig_rte_entry)
    {
    BgpT_Rte_Entry*     new_rte_entry_ptr;
    /** Makes a copy of the original route entry.           **/
    FIN (bgp_support_rte_entry_copy (orig_rte_entry))

    /* Allocate memory for the new route.                   */
    new_rte_entry_ptr = (BgpT_Rte_Entry*) op_prg_pmo_alloc (bgp_rte_entry_pmh);

    /* Set the number of references to the particular route */
    /* enrty to be 0.                                       */
    new_rte_entry_ptr->num_references = 0;

    /* I added this part to copy penalty information for the shared list of new entries*/

    new_rte_entry_ptr->Penalized = orig_rte_entry->Penalized;
    /* Copy the id of the peer that received this update    */
    new_rte_entry_ptr->local_adv_receive_peer_id = orig_rte_entry->local_adv_receive_peer_id;

    /* Copy the BGP ID of the neighbor originating this update    */
    new_rte_entry_ptr->advertising_rtr_bgp_id  = orig_rte_entry->advertising_rtr_bgp_id;

    /* Copy the AS number of the neighbor originating this update    */
    new_rte_entry_ptr->advertising_rtr_as_number = orig_rte_entry->advertising_rtr_as_number;

    /* Copy the neighbor_type.                              */
    new_rte_entry_ptr->advertising_neighbor_type = orig_rte_entry->advertising_neighbor_type;

    /* Copy the degree of preference for this route which   */
    /* be changed if required by other routers.             */
    new_rte_entry_ptr->degree_of_preference = orig_rte_entry->degree_of_preference;

    /* Set the input prefix as the destination prefix ptr   */
    new_rte_entry_ptr->dest_prefix_ptr  = bgp_support_mp_prefix_copy (orig_rte_entry->dest_prefix_ptr);

    /* Set the path attribtues                              */
    new_rte_entry_ptr->path_attrs_ptr   = bgp_support_path_attrs_copy (orig_rte_entry->path_attrs_ptr);

    new_rte_entry_ptr->weight = orig_rte_entry->weight;

    FRET (new_rte_entry_ptr);
    }
```

Figure 48 Modified bgp_support_rte_entry_copy function

The first step is to inspect a BGP packet whenever it reaches the Update Message state under the bgp parent process which invokes the child bgp_conn related to the packet received. Once the bgp_conn process is invoked, it sorts the packet based on its type. We are interested in the packets that come as a stream type which, represent an update received from the neighbour (figure 49). All modifications start at function bgp_conn_update_message_handle () that reside in the bgp_conn Established State Exit Executives. Our Code Workflow in figure 50 shows the procedure inside this function. We actually update our history list and penalties at this function.

```
case OPC_INTRPT_STRM:
case OPC_INTRPT_PROCEDURE:
    {
    /* There can be two reasons for a stream interrupt.              */
    /* 1. A message has come in from the peer.                       */
    /* 2. Another peer process has received an UPDATE message and    */
    /* the route information is being disseminated by the root.      */

    /* Get the argument memory associated with this invocation       */
    intrpt_info_ptr = (BgpT_Intrpt_Info *)op_pro_argmem_access ();

    if (intrpt_info_ptr == OPC_NIL)
        {
        /* This is case 2: another peer process has received an UPDATE */
        /* which is not being disseminated. This case also includes an */
        /* UPDATE that has been temporarily delayed because of MRAI    */
        /* timer compliance.                                           */

        /* This is an indication that an UPDATE was received by one */
        /* of the neighbor processes running in this router which   */
        /* caused the local table to be changed. This change has to */
        /* be advertised to this child's peer.                      */
        bgp_conn_new_route_indication_handle ();
        }
    else
        {
        /* This is case 1: A message has come in from the peer */
        /* Get the received packet and packet type                  */
        data_pkptr = intrpt_info_ptr->msg_pkptr;
        received_packet_type = intrpt_info_ptr->msg_type;

        /* Send out the tcp receive command for one packet.         */
        tcp_receive_command_send (tcp_hndl, 1);

        /* Check the received packet type. An Update message has to */
        /* be properly handled. If just a keep alive, reset the     */
        /* hold timer.                                              */
        if (received_packet_type == BgpC_Packet_Type_Update)
            {
            /* Handle the Update message.                           */
            bgp_conn_update_message_handle (data_pkptr);

            /* Also reset the Hold Timer.                           */
            bgp_conn_timer_reset (BgpC_Timer_Hold);
            }
```

Figure 49 bgp_process Established State Exit Executives

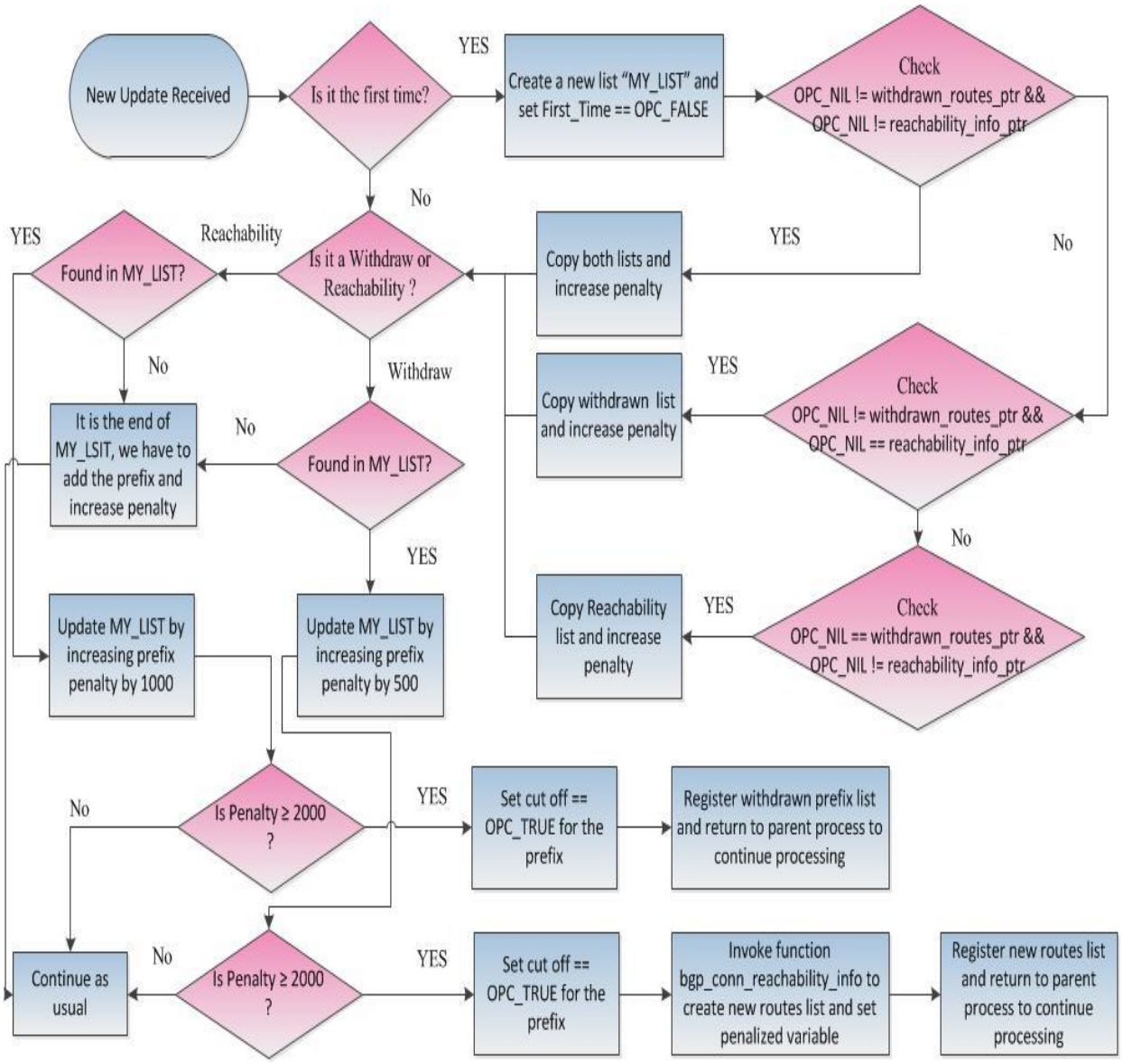


Figure 50 Modified bgp_conn_update_message_handle() function code workflow

After that, *bgp_conn* process returns the handle to the parent process with the lists of withdrawn routes and the lists of new routes. In this part of the code, we create the aggregated route and decide to send it or not as follows:

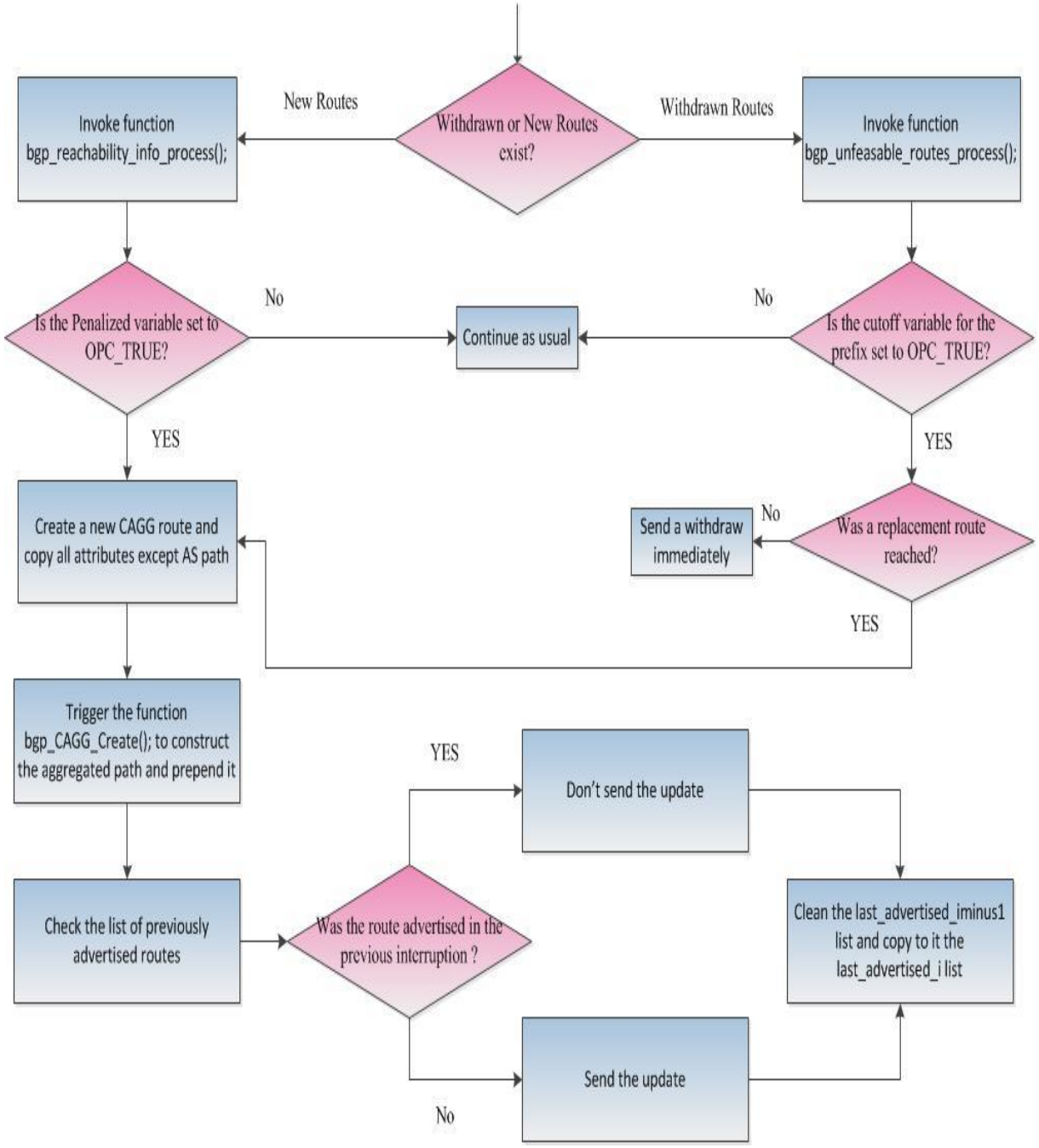


Figure 51 bgp_update_message_State_enter_exicutives code workflow

Finally, the process of best path selection is invoked while processing new routes where we modified the code to reflect the following workflow:

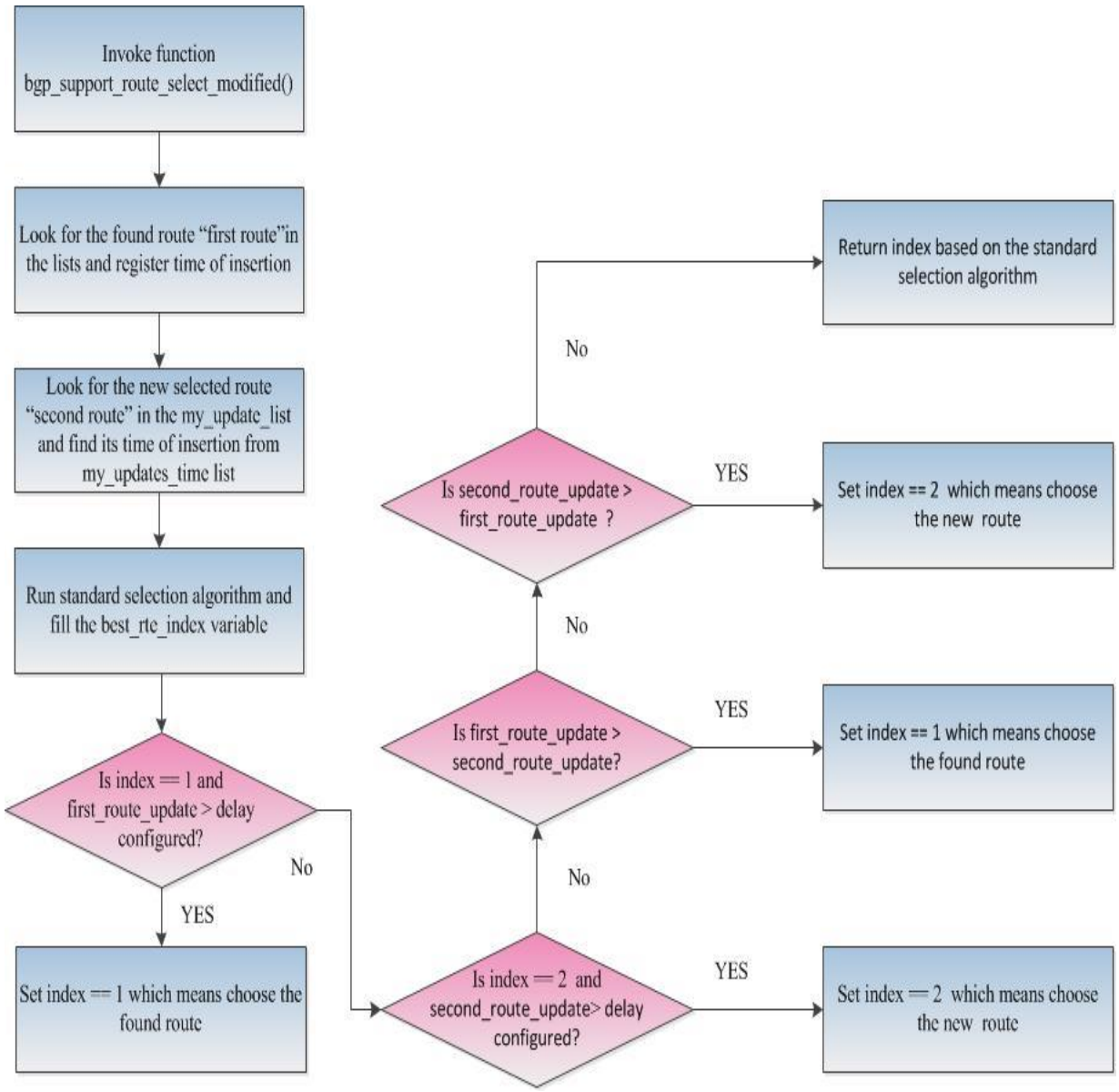


Figure 52 bgp_support_route_select_modified() Code Workflow

5.3 SPAGG Results

In this section, we examine the results of running our version of SPAGG BGP and focus on the improvements compared to previous results in simulation 3 and 4. We explain our results in two parts: simulation 5 and simulation 6.

5.3.1 Results of Simulation 5

This simulation compares with simulation 3 where previously we could see the improvement in AS9 convergence activity and the overall duration of the network while traffic delays still exist, and as a result, applications suffer from traffic drops. This is mainly because AS7 always chooses the path through AS4 and then through AS5 to finally choose the path through AS8. Path exploration occurs at AS7 although it successfully reduces BGP updates for upstream routers. In this simulation we continue to achieve the same convergence activity, duration, and BGP updates statistics as in simulation 3. However, we can see that after the 800th second, when the cutoff threshold is reached, no delays or drops are observed. This is despite the fact that link between AS1 and AS2 is still flapping. The following steps are the sequence of events:

1. An update is received from AS4 and AS5 for the prefix under study at 800th second.
2. Cutoff threshold is reached at 800th second when penalty is 2500 per prefix per neighbour.
3. The CAGG process is invoked and a new route with aggregated path is constructed and advertised to AS9.
4. Then, the process of path selection is invoked and the three paths (from AS4, AS5, AS8) are evaluated based on the workflows mentioned in previous section.
5. The path through AS8 is chosen since it has been in the routing table for the longest time and its penalty is 0.
6. AS9 stops receiving updates from AS7 while the flap is happening and AS7 continues to use the stable path through AS8.

After second 500, we see a 100% improvement in reachability and traffic delays. In figures 53, we see how email traffic only experiences an increase in delay between second 400 and 500, and that is when the first flap occurred and packets are dropped briefly at AS5. However, after the first path exploration cycle passes and the selection naturally stables on using the path through AS8, we can see that there is not a single drop or increase of delay. This is because when the second convergence at second 800 happens, our algorithm is invoked and consistently chooses AS8 instead of going through the path exploration cycle again. Our algorithm is capable of stabilizing the network for the rest of the simulation.

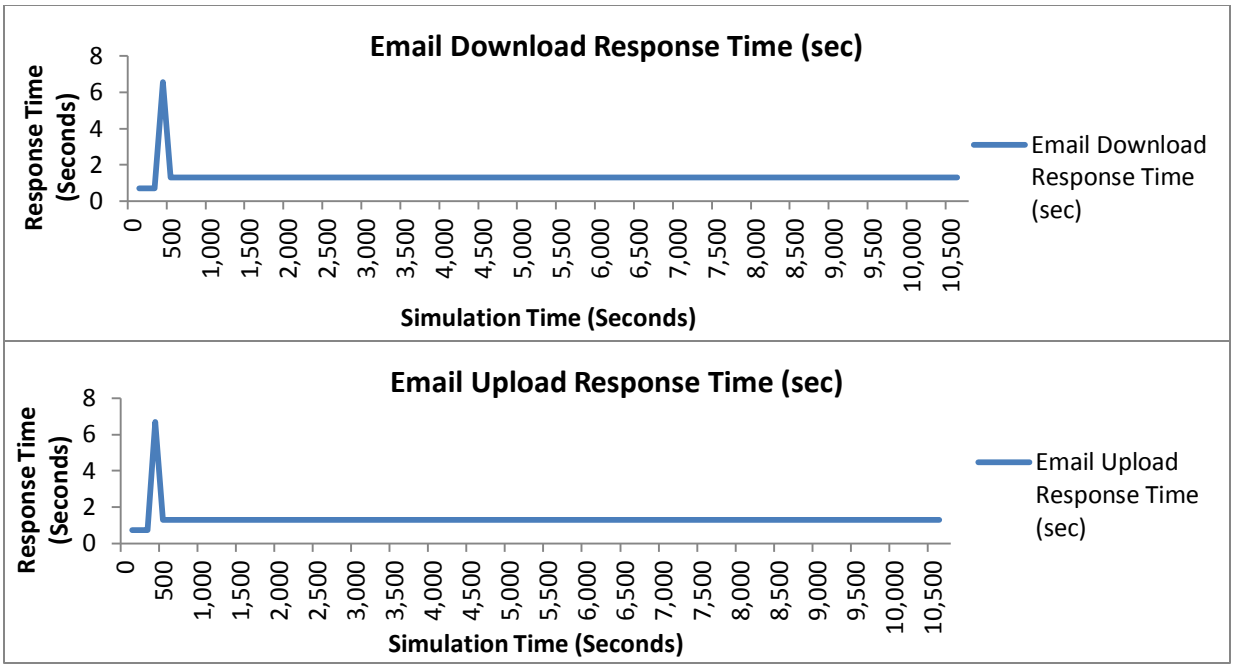


Figure 53 Email Download/Upload Response Time (Simulation 5)

Traffic sent/received is also following the same pattern where traffic is dropped briefly during the first path exploration.

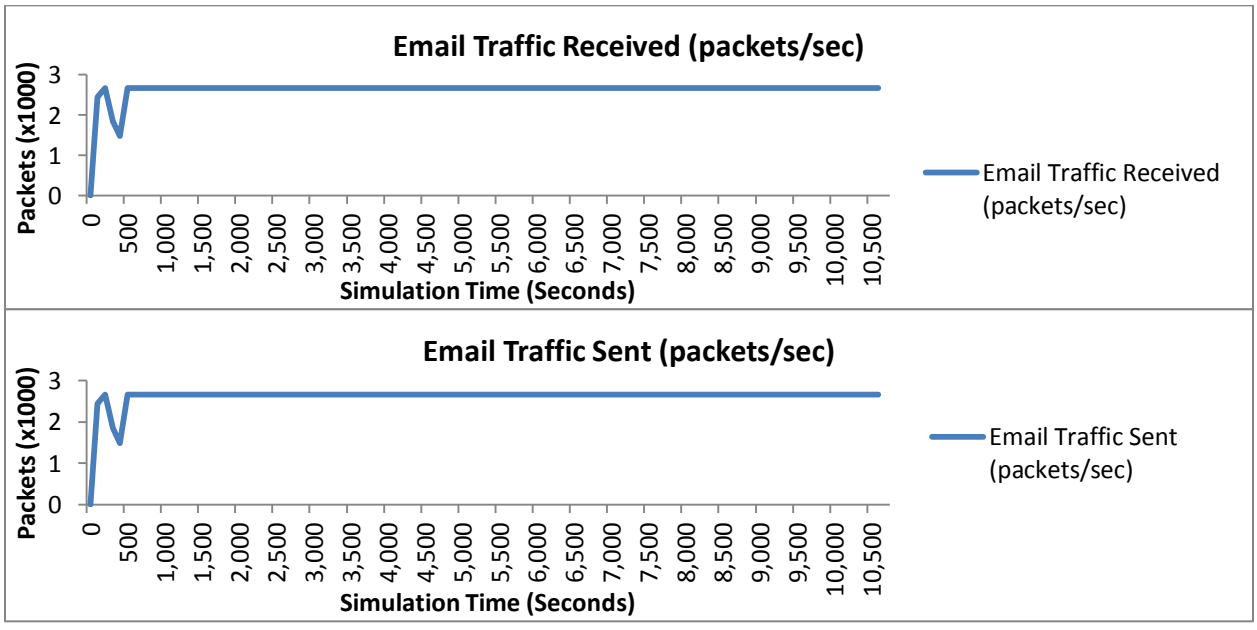


Figure 54 Email Traffic Sent/Received (Simulation 5)

If we look at the ICMP traffic, we find a similar result where traffic drops are now limited to the period between 400 and 500 seconds and we do not find any drops in subsequent periods although flapping is

taking place in the network. Also, ICMP delay is stabilized at almost 330 ms and this is a result of using a longer path.

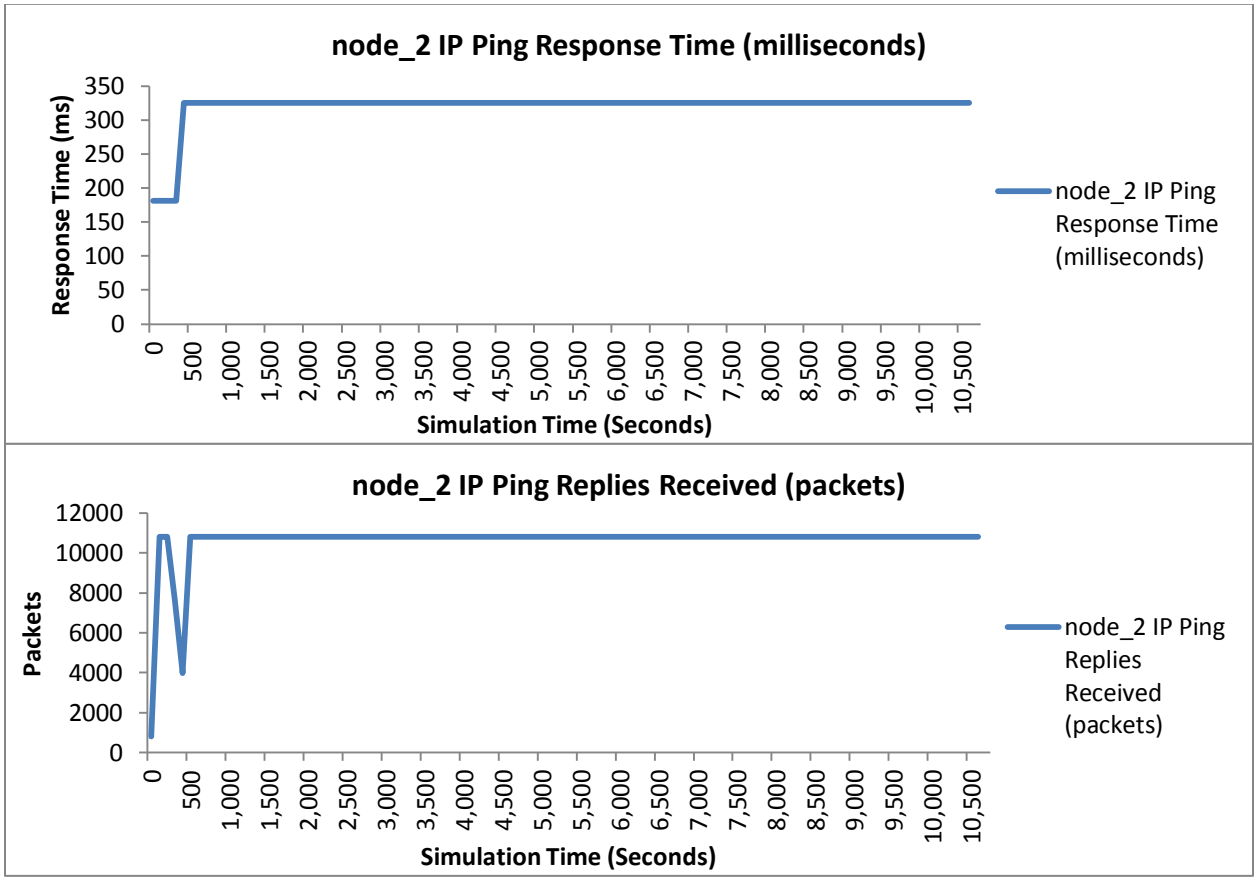


Figure 55 Ping Replies Received and Response Time (Simulation 5)

For traffic drops, we see a 100% improvement after second 500. We only see a drop of 23000 packets in the first exploration period and we do not see any drops after that due using a stable path for the rest of the simulation.

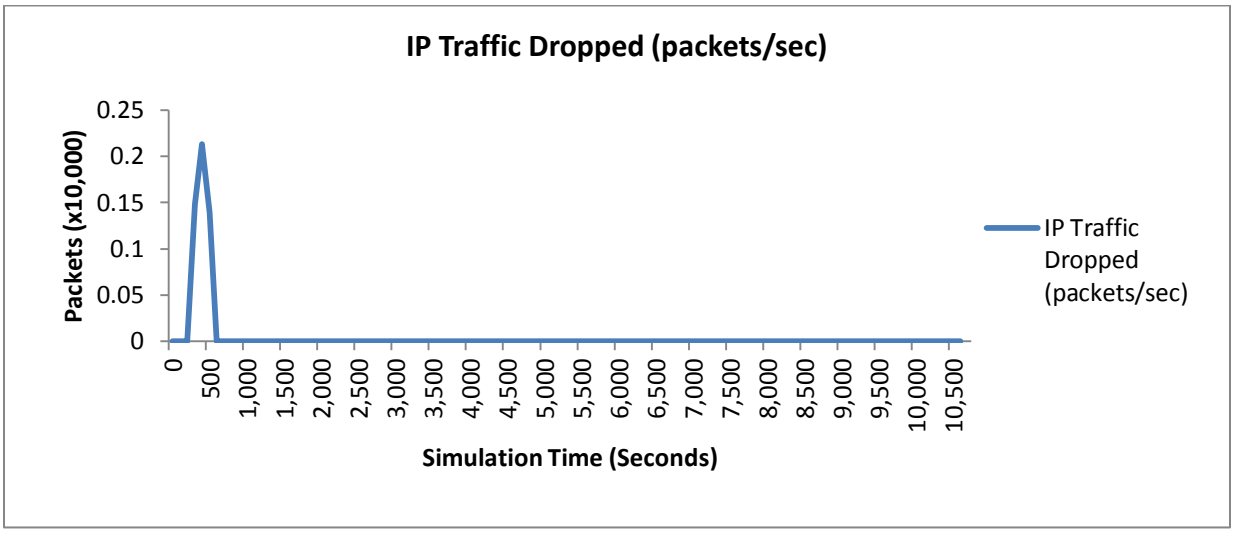


Figure 56 IP Packet Drops (Simulation 5)

5.3.2 Results of Simulation 6

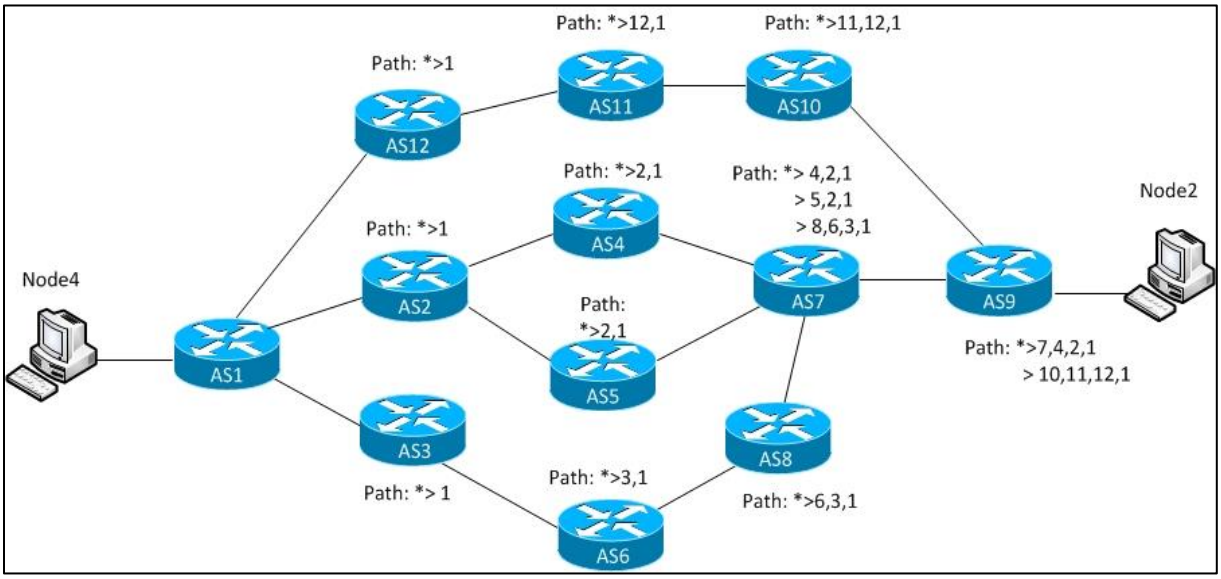


Figure 57 Modified Test Network for Simulation 6

In this simulation, we approach the solution of the AS path shortening problem. We show that no matter what is the scenario, our AS9 uses the intended path. The key point in this simulation is that we prepend the aggregated path to make it the same length as the last known best path penalized used at AS7. At second 800 when AS7 activates AS path aggregation, it checks the first path under penalty which was 7,4,2,1. In our algorithm, the resulting path is: 7,7,{4,5,8,3,2}1 which is of length 4. Hence, when AS9 receives this new path, it still prefers its stable path 10,11,12,1 because it is already chosen in the BGP table for its precedence. We can see in figures 58 and 59 how AS7-AS9 is only used at the beginning of the simulation and at second 400 the first update happen, causing the natural shift to AS10. After that, at second 800 the new CAGG path is received, but to the contrary of the previous scenario, it does not cause AS9 to shift back to AS7. Hence, traffic does not experience any delays or drops due to the usage of a stable path which is through AS10.

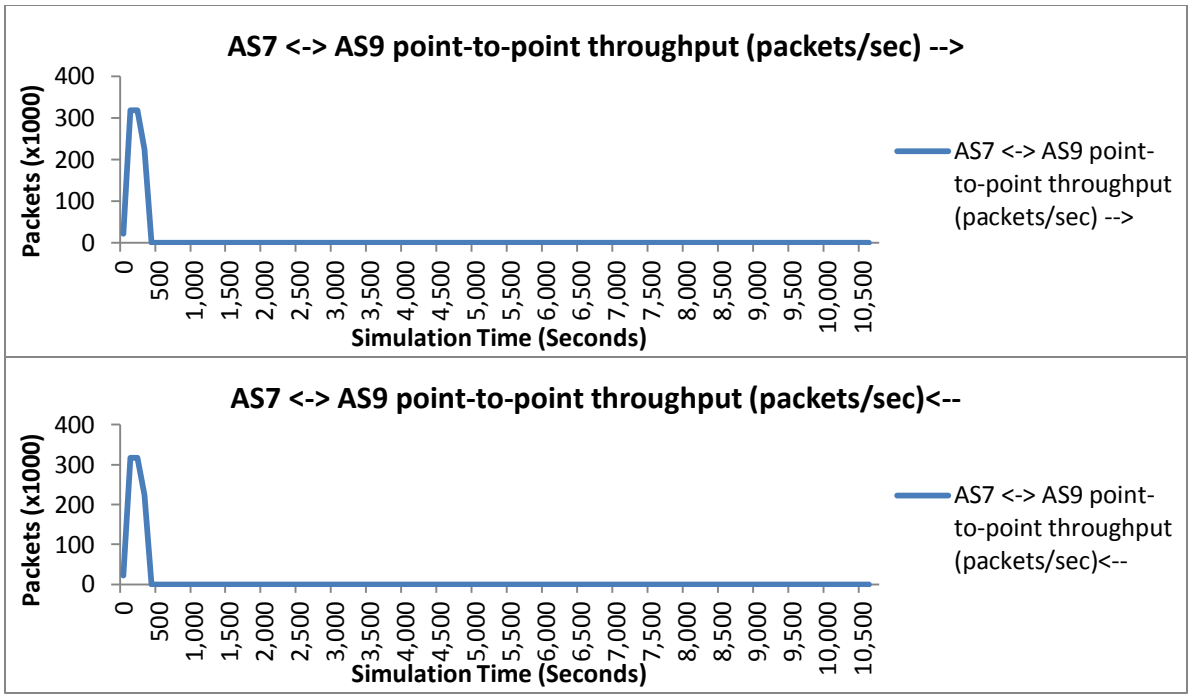


Figure 58 AS7-AS9 Utilization for Simulation 6

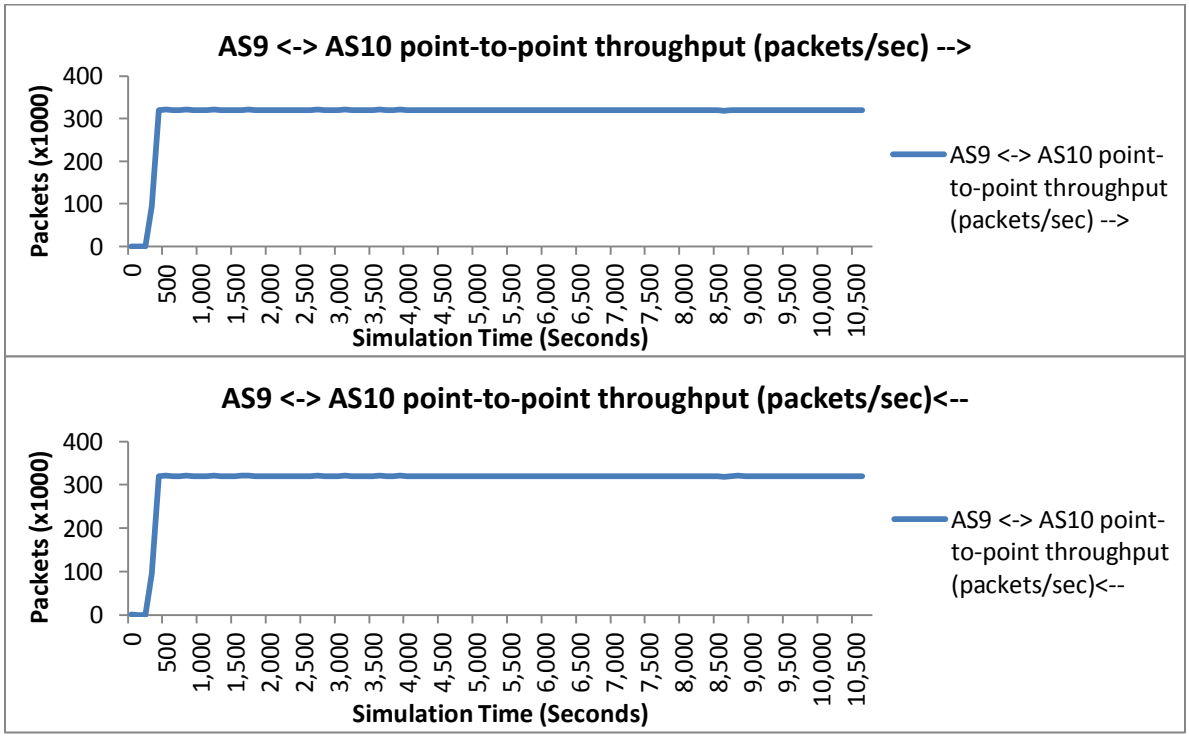


Figure 59 AS9-AS10 Utilization for Simulation 6

Even if the original path at the beginning of the simulation was through AS10 due to the usage of MED attribute, all the flaps and the aggregation do not affect our path selection or traffic flow. We set up our simulation to choose the path through AS10 initially. The simulation shows that this stable preferred path was used during the whole time and was not affected by receiving the aggregated AS path as seen in simulation 4.

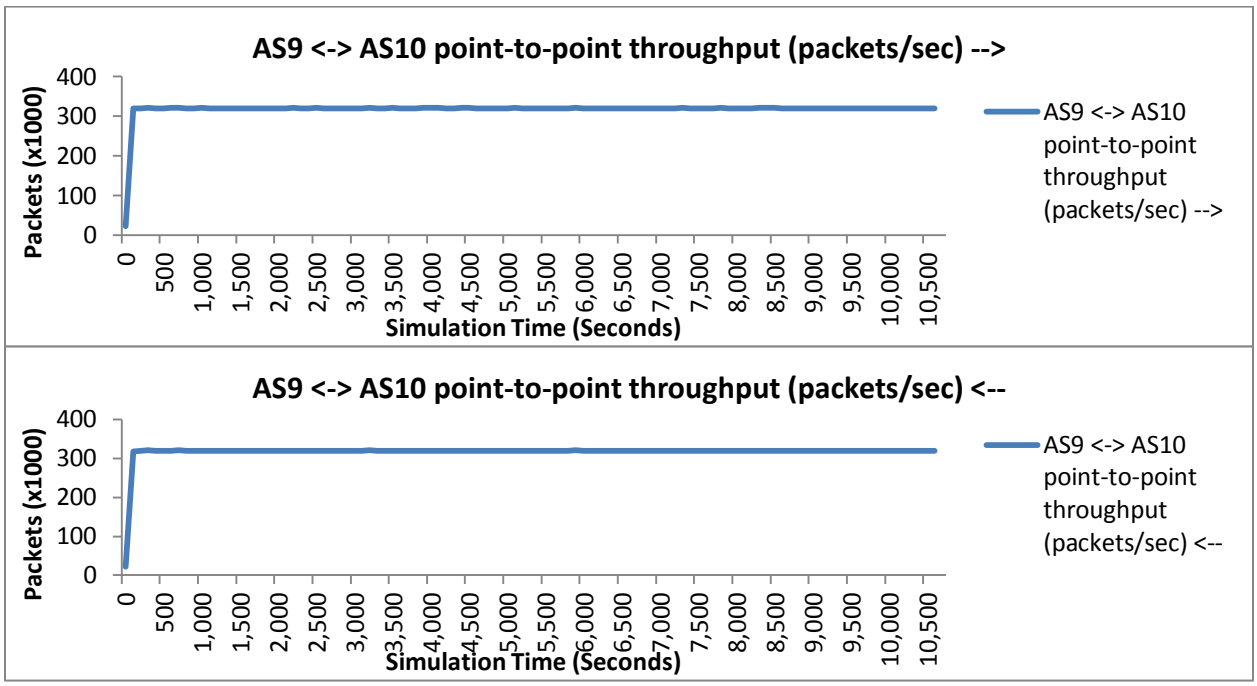


Figure 60 AS9-AS10 Utiliztion for Simulation 6

Chapter Six

Conclusion and Future Work

In this research, we examined one of the unresolved problems associated with the operations of BGP routing protocol. Route flaps and path exploration are still affecting service provider networks when it comes to fast convergence and maintaining a healthy network. We analysed some of the solutions proposed in the past, and explained why they have not been implemented or adopted by network vendors. Unfortunately, solutions like Path Exploration Damping are proven successful to lower the amount of updates sent by BGP but it is also proven to extend convergence duration in some scenarios. Hence, we focused on the most recent solution of AS path aggregation which, is proven to be successful in suppressing unnecessary updates while also shortening the duration of BGP convergence. However, in our study of this solution we come to the conclusion that Churn Aggregation suffers from shortcomings that can be devastating to the data traffic or the service providers' choice of best path. We explained in details two types of problems that can arise: Path Selection Problem at the aggregator, and the AS Path Shortening Problem. We showed in details through simulation how both problems exist in a network that applies the aBGP protocol described in [9,26]. After that, we proposed two solutions for the path selection and AS path shortening problems, and described our new modified algorithm: SPAGG. We implemented our algorithm in OPNET simulator by programming all changes in C language and Proto-C. We also simulated by recreating the scenarios with our new modified version of SPAGG BGP. The simulation results show that the proposed algorithm addressed both path selection and AS path shortening problems without packet delays or drops after a certain point. Further, the problem of AS path shortening is completely solved in SPAGG by prepending the AS path of the aggregated path.

In summary, our research on improving the AS path aggregation algorithm resulted in several contributions like:

- An analysis of previous solutions to the problem of link flaps and path exploration.
- A deep analysis on how Churn Aggregation affects data traffic after handling link flaps and showing this negative effect in simulation.
- A new modified algorithm that is based on AS path aggregation to resolve the problems of BGP churn and path exploration. In our algorithm, we show how we solve the problem of shortening AS paths that can lead to inconsistencies in the policies of AS path selection.
- An algorithm for AS path selection that is invoked after aggregating an AS path at the aggregator router. This algorithm tries to find the most stable route at times of convergence even if it is not the most preferable based on the standard BGP selection criteria.
- An implementation of our solutions described in code workflows and carried out in C language.
- Simulation results of our solution showing the improvement over any of the other simulations.

Finally, there are several areas of future work. It is worth doing the sensitivity analysis of ω in the future with several scenarios by varying the value of ω to clearly see the effect of this configurable parameter on the convergence process. Also, the algorithm can be further tested by implementing it partially or entirely in the network to achieve faster convergence. On the other hand, the solution of AS path aggregation is distinct in the sense that it does not need a dramatic change in the protocol itself. Unlike most of the other solutions, there is no need to create a new attribute field or change the mechanism of the protocol operations. All the changes mentioned in this document can be part of the router's software image. This creates an opportunity to export the tasks of AS path aggregation and path selection to external software. The software can act as a monitoring tool where it builds history lists and enforces the aggregated path to be sent through the router at times of convergence. It would also enforce using the stable route. However, some changes to the router's software might be needed. This idea would unburden the router from saving long lists of history prefixes.

References

[1] Y. Rekhter, T. Li, S. Hares. *A Border Gateway Protocol, RFC 4271 (BGP-4)*, January 2006.

[2] Iljitsch van Beijnum. *Building Reliable Networks with the Border Gateway Protocol*, O'Reilly Media, 2002.

[3] K. Lougheed, Y. Rekhter. *A Border Gateway Protocol*, RFC 1105 (BGP-1), June 1989.

[4] Cisco Document ID: 26634. *Bgp case studies 4 - route flap dampening* http://www.cisco.com/en/US/tech/tk365/technologies tech note09186a00800c95bb.shtml.

[5] RIPE Routing Working Group. *Recommendations on route-flap damping*, RIPE DocumentID378, http://www.ripe.net/docs/ripe-378.html.

[6] Alex Fabrikant et al. *There's something about MRAI: Timing diversity can exponentially worsen BGP convergence*, Proceedings of IEEE INFOCOM, 2011.

[7] P. Jakma. *Revised default values for the bgp minimum route advertisement interval*, Internet Draft, Sun Microsystems, 2010, http://tools.ietf.org/html/draft-ietf-idr-mrai-dep-02.

[8] Geoff Huston, Mattia Rossi, and Grenville Armitage. *A Technique for Reducing BGP Update Announcements through Path Exploration Damping*. IEEE Journal On Selected Areas in Communications, VOL. 28, NO. 8, 2010.

[9] Wang Xiaoqiang and Olivier Bonaventure. *Stabilizing BGP Routing without Harming Convergence*. 14th IEEE Global Internet Symposium (GI) at IEEE INFOCOM, 2011.

[10] P. Brighten Godfrey, Matthew Caesar, Ian Haken, Yaron Singer, Scott Shenker, Ion Stoica. *Stabilizing Route Selection in BGP*, University of Illinois at Urbana-Champaign, 2010.

[11] OPNET Official Website. http://www.opnet.com/solutions/network_rd/modeler.html

[12] RouteViews Project, University of Oregon. http://www.routeviews.org/

[13] F. Baker. *Requirements for IP Version 4 Routers*, RFC 1812, June 1995.

[14] Sam Halabi, Danny McPherson. *Internet Routing Architectures*, Cisco Press, Second Edition, August 23, 2000.

[15] Rahul Sami, Michael Schapi, Aviv Zohar. *Searching for Stability in Interdomain Routing*, Proceedings of IEEE INFOCOM, pp. 549-557, 2009.

[16] Wendell Odom, Rus Healy, Naren Mehta. *CCIE Routing and Switching Certification Guide Third Edition*, Cisco Press, 2008.

[17] Martin James. *CCIE Routing and Switching Practice Labs*, Cisco press, June 10, 2008.

[18] Bahauddin Kazi. *Policy Disputes in BGP: Analysis, Detection and Proposed Solution*, Master's Thesis, Ryerson University, 2012.

[19] Cisco Documentation, *BGP Best Path Selection Algorithm*, Document ID: 13753, 2006, http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a008004431.shtml

[20] C. Villamizar et al. *BGP Route Flap Damping*, RFC2439, 1998.

[21] Zhuoqing Morley Mao et al. *Route Flap Damping Exacerbates Internet Routing Convergence*, Proceedings of ACM SIGCOMM, 2002.

[22] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. *Delayed Internet Routing Convergence*, Proceedings of ACM SIGCOMM, 2000.

[23] Nenad Lasković. *BGP WITH AN ADAPTIVE MINIMAL ROUTE ADVERTISEMENT INTERVAL*, SIMON FRASER UNIVERSITY, 2006.

[24] Wang Wenhua et al. *New MRAI Setup Mechanism for Enhancing BGP Route Convergence*, ICAT XXII International Symposium, 2009.

[25] S. Hares and A. Retana. *BGP 4 implementation report*, Internet Draft, Oct. 2004. http://www.ietf.org/internet-drafts/draft-ietf-idrbgp-implementation-02.txt/

[26] W. Xiaoqiang and O. Bonaventure. *Stabilizing bgp routing without harming convergence*, Technical Report, http://u.sohu.com/download/10/12941224217240688676116, 2010.

[27] OPNET Modeler Solution Overview, OPNET Technologies Inc, 2011, http://www.opnet.com/solutions/brochures/R&D_NEMs.pdf

[28] OPNET Modeler 15 Help Documentation, OPNET Technologies Inc, 2011.

[29] Zheng Lu, Hongji Yang. *Unlocking the Power of OPNET Modeler*, Cambridge Press, 2012.

[30] *Introduction to OPNET Modeler*, OPNET Technologies, 2007.

[31] Brian W. Kernighan and Dennis M. Ritchie. *C Programming Language* (2nd Edition), 1988.

[32] Steve Summit. *C Programming FAQs: Frequently Asked Questions*, 1995.

[33] Chris Sells and Jon Flanders Ian Griffiths. *Mastering Visual Studio .Net*, March 2003.

[34] Joe Mayo. *Microsoft Visual Studio 2010: A Beginner's Guide*, 2010.

[35] Wei Sun, Zhuoqing Mao, and Kang Shin. *Differentiated BGP Update Processing for Improved Routing Convergence*. Proceedings of the 2006 IEEE International Conference on Network Protocols, November 2006.

[36] Dan Pei, Matt Azuma, Dan Massey, and Lixia Zhang. *Bgp-rcn: improving bgp convergence through root cause notification*, The International Journal of Computer and Telecommunications Networking, Volume 48 Issue 2, June, 2005.

[37] T. Griffin and B. Premore. *An experimental analysis of BGP convergence time in Network Protocols*. Ninth International Conference on Network Protocols, Nov, 2001.

[38] Jaideep Chandrashekar, Zhenhai Duan, Zhi-Li Zhang, and J. Krasky. *Limiting path exploration in bgp*, INFOCOMM 24th Annual Joint Conference of the IEEE Computer and Communications Societies, 2005.

[39] TG Griffin. *Analysis of the MED oscillation problem in BGP*. Proceedings of IEEE International Conference on Network Protocols (ICNP), 2002.

[40] Dimitri Papadimitriou et al. *Path-vector Routing Stability Analysis*, ACM SIGMETRICS Performance Evaluation Review, Volume 39 Issue 3, December 2011.

[41] T. Li and G.Huston. *BGP Stability Improvements*, Internet-Draft (Informational), Jun. 2007. http://tools.ietf.org/html/draft-li-bgp-stability-01

[42] Rachit Agarwal, Virajith Jalaparti, Matthew Caesar, P. Brighten Godfrey. *Guaranteeing BGP Stability With a Few Extra Paths*, Proceedings of IEEE 30th International Conference, Distributed Computing Systems (ICDCS), June 2010.

[43] Hasan T. Karaoˇglu, Murat Y¨uksel, and Mehmet H. G¨unes. *On the Scalability of Path Exploration Using Opportunistic Path-Vector Routing*, Proceedings of IEEE International Conference, Communications (ICC), June 2011.

[44] Luca Cittadini et al. *From Theory to Practice: Efficiently Checking BGP Configurations for Guaranteed Convergence*, IEEE Transactions on Network and Services Management, Volume 8, Issue 4, Page(s): 387 – 400, 2011.

[45] Bin Wang. *The Research of BGP Convergence Time*, Information Technology and Artificial Intelligence Conference (ITAIC), IEEE Joint International, August 2011.

[46] Huaming Guo et al. *On the convergence condition and convergence time of BGP*, Computer Communications Journal, Volume 34, Issue 2, Pages 192–199, February 2011.

[47] Soroush Haeri, Dario Kreˇsi´c, Ljiljana Trajkovi´c. *Probabilistic Verification of BGP Convergence*, Network Protocols (ICNP), Proceedings of the 19th IEEE International Conference, October 2011.

[48] Pei-chun Cheng et al. *Route Flap Damping with Assured Reachability*, North American Network Operators' Group, NANOG51, February 2011.

[49] Mattia Rossi. *BGP Path Exploration Damping (PED)*, Centre for Advanced Internet Architectures (CAIA), Swinburne University of Technology, 2011

[50] Zhi Qi. *AS Path-Prepending in the Internet and Its Impact on Routing-Decisions*, Institute of Network Architecture Technical University of Munich, 2006.

Appendix A

Complete Simulation Results

A.1 Simulation 1 Results

A.1.1 FTP

Like Email traffic, Download/Upload response time is constant on almost 1.9 seconds during the simulation without any change due to the absence of network convergence activities.

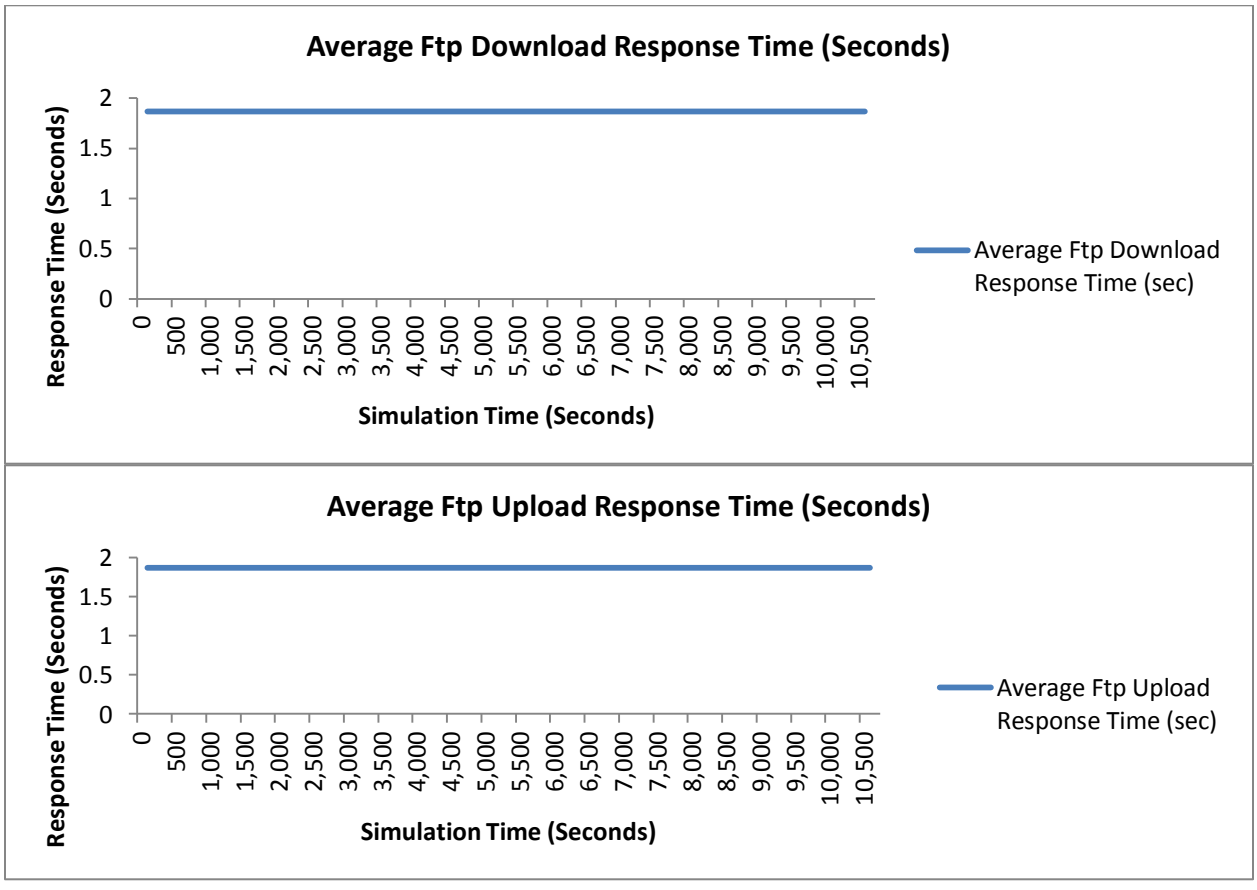


Figure 61 FTP Download/Upload Response Time (Simulation 1)

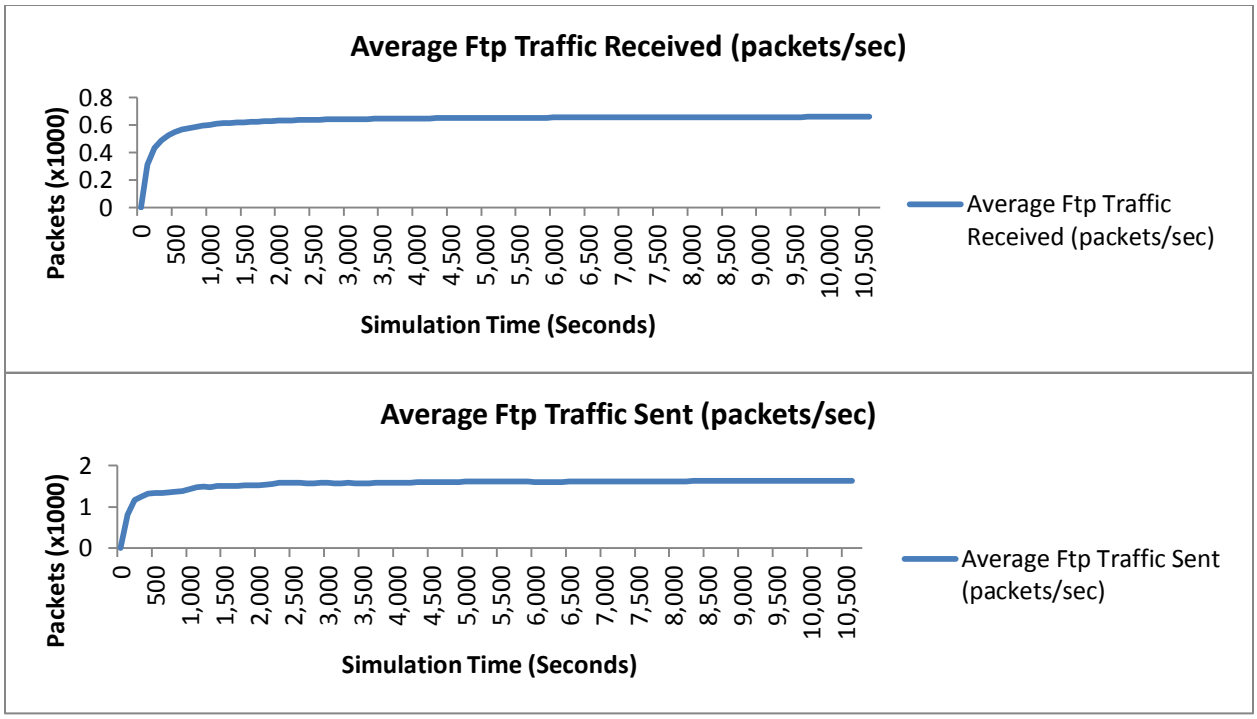


Figure 62 FTP Traffic Sent/Received (Simulation 1)

A.1.2 HTTP

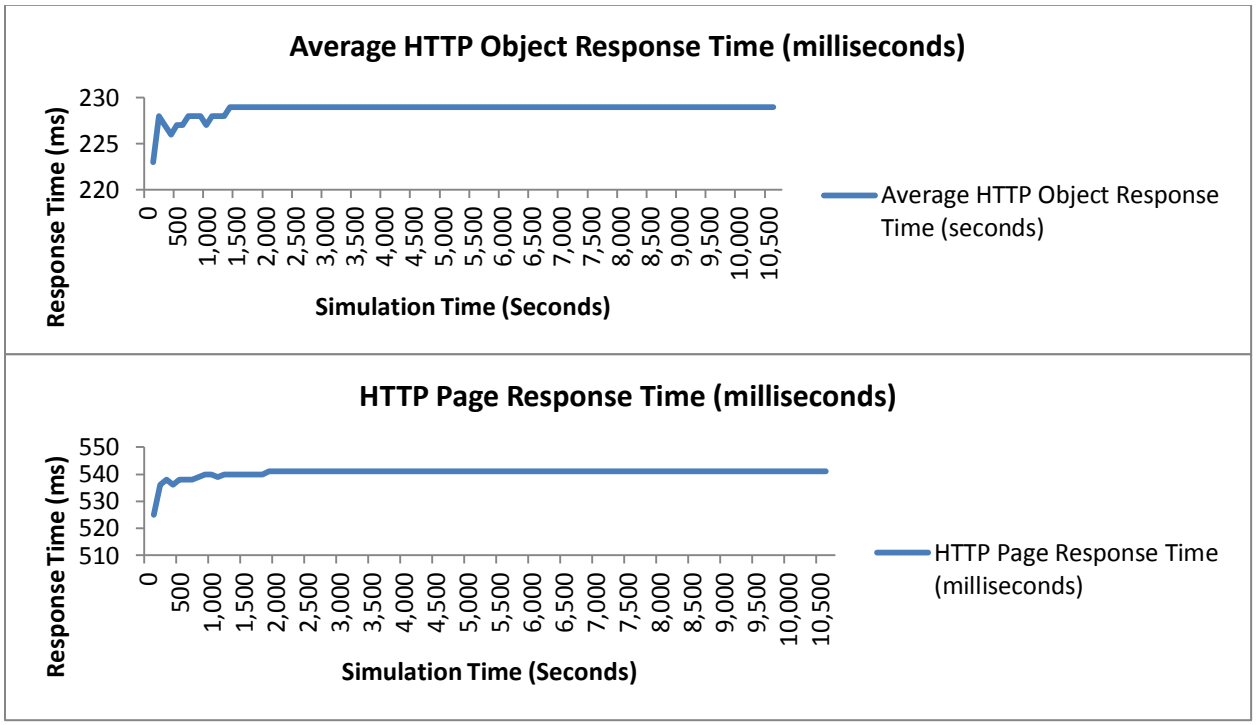


Figure 63 HTTP Object Response/Page Response Time (Simulation 1)

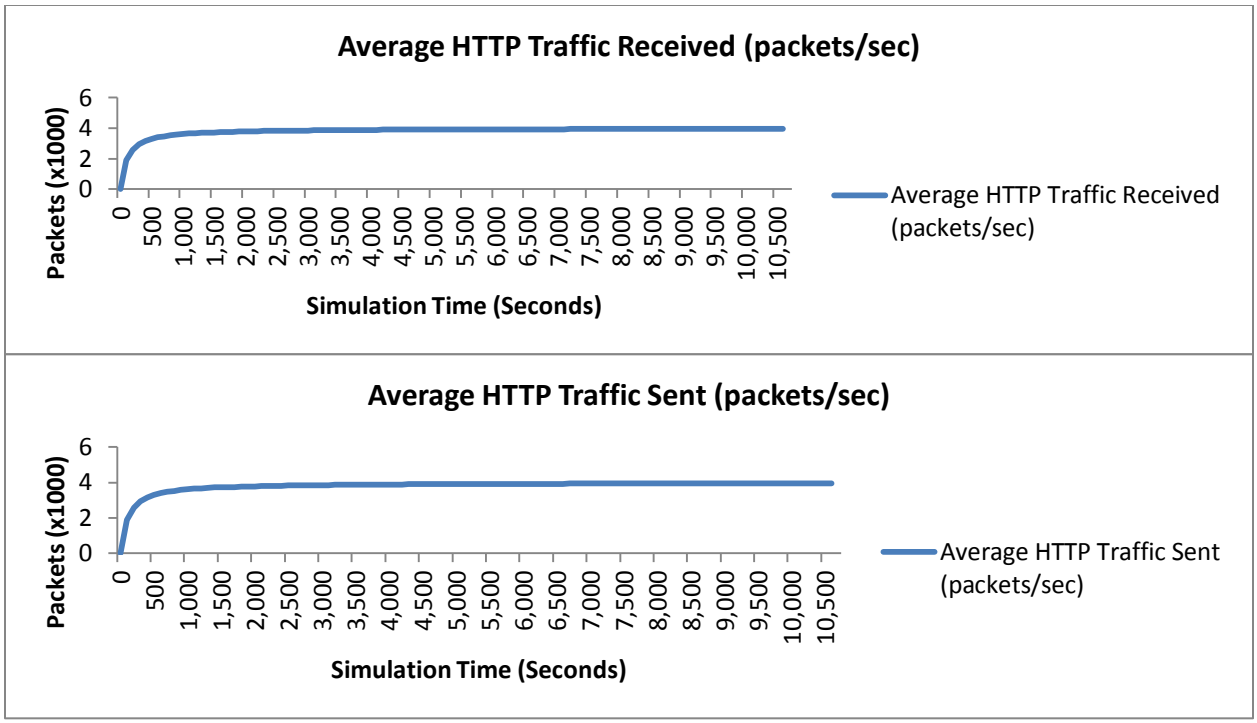


Figure 64 HTTP Traffic Sent/Received (Simulation 1)

A.1.3 Links Utilization

In order to see how traffic is moving in OPNET, the best way is to check on link utilization. As we expected, link AS7-AS9 is utilized throughout the simulation and traffic is over 300,000 packets/second. The same utilization is expected on link AS7-AS4 as shown. Also, link AS7-AS8 is not being utilized because no failures are happening that force a change of path.

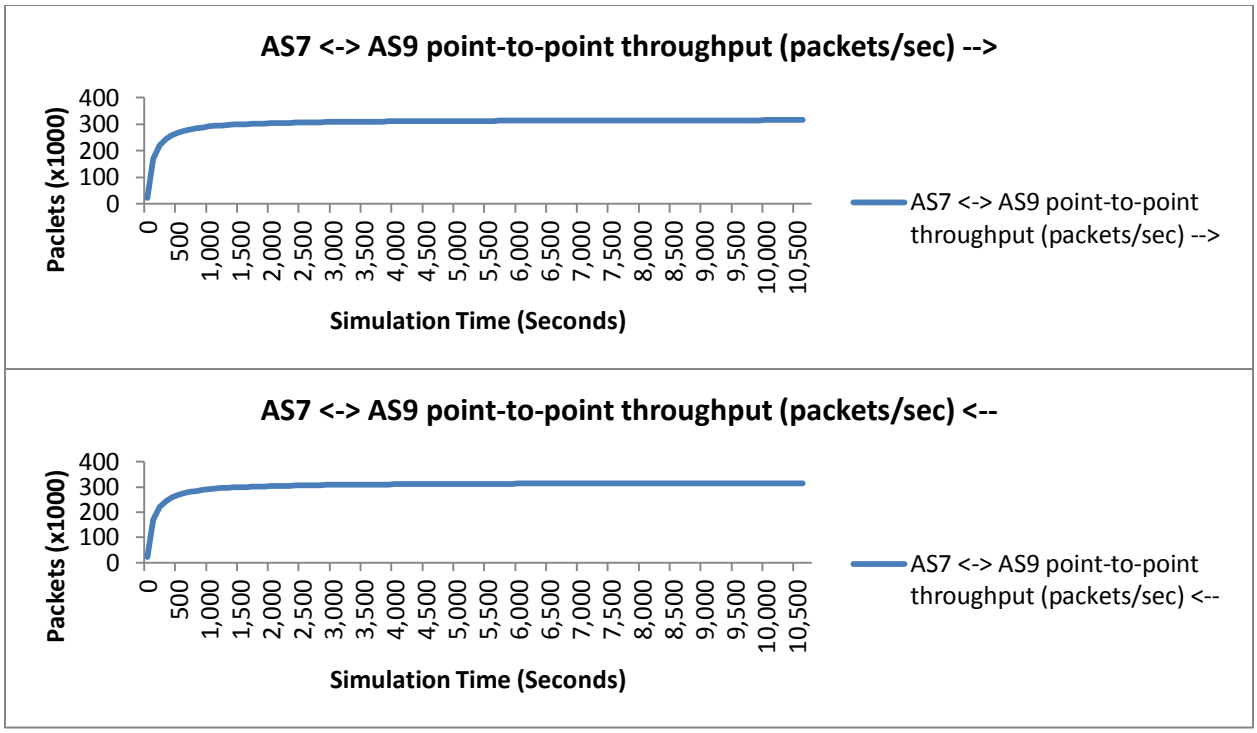


Figure 65 Link AS7-AS9 Utilization (Simulation 1)

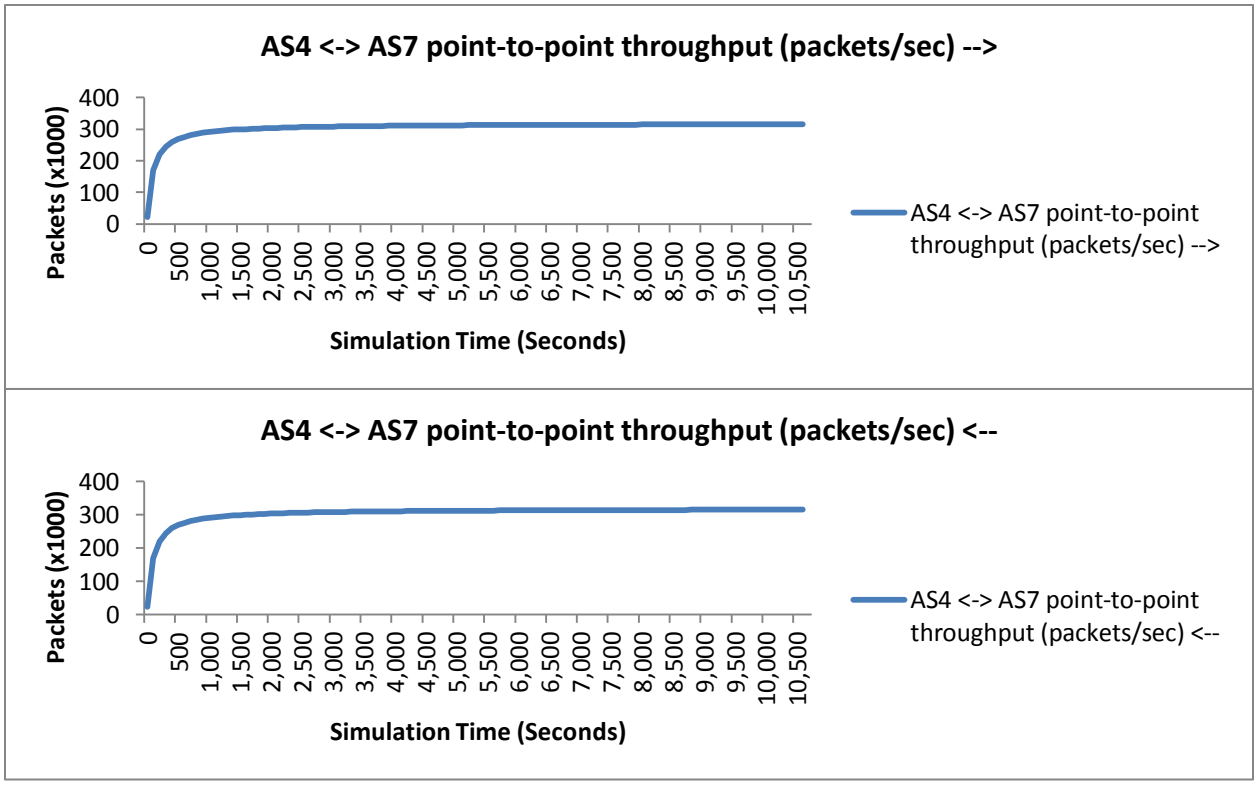


Figure 66 Link AS7-AS4 Utilization (Simulation 1)

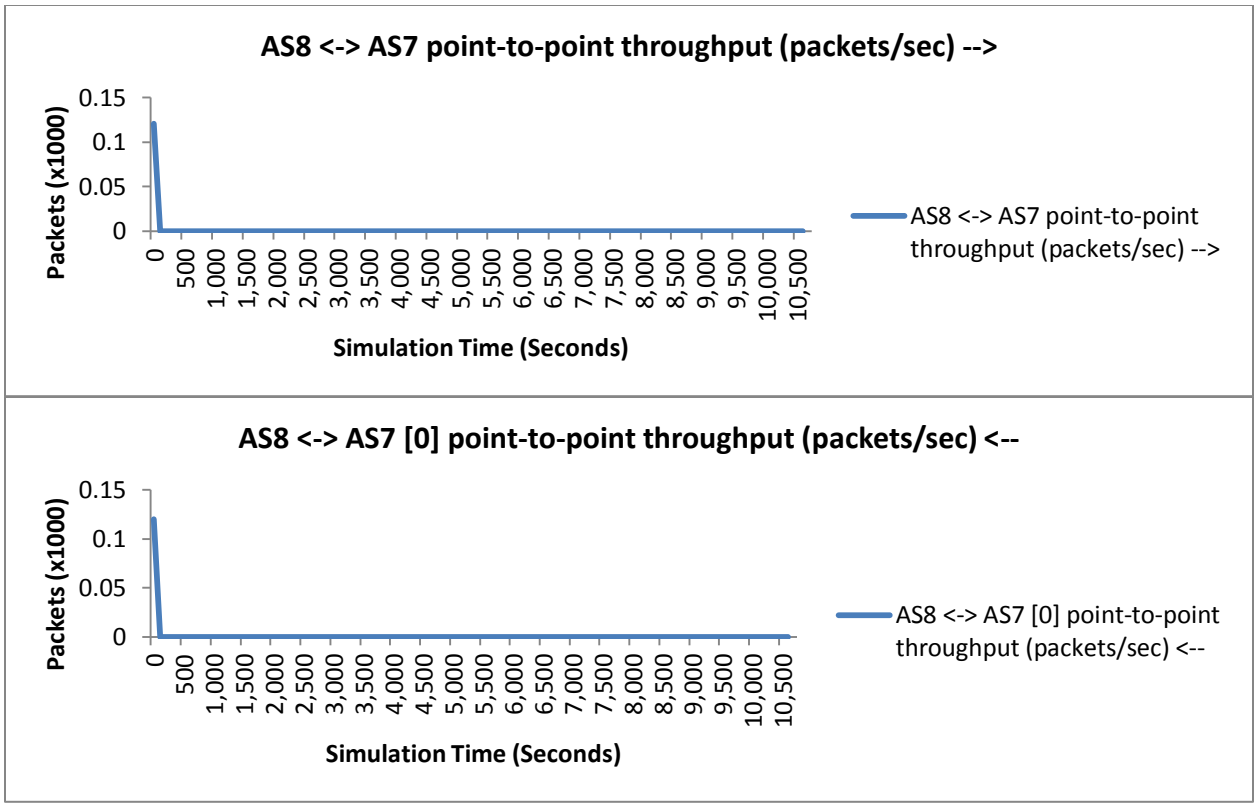


Figure 67 Link AS7-AS8 Utilization (Simulation 1)

A.2 Simulation 2 Results

A.2.1 FTP

FTP had a Download Response time of 15 seconds compared to 1.9 (average) seconds in the first scenario and an Upload response time of 16 seconds compared to 1.9 (average) seconds. After 3700 seconds, the delay stabilizes and the traffic is back to normal.

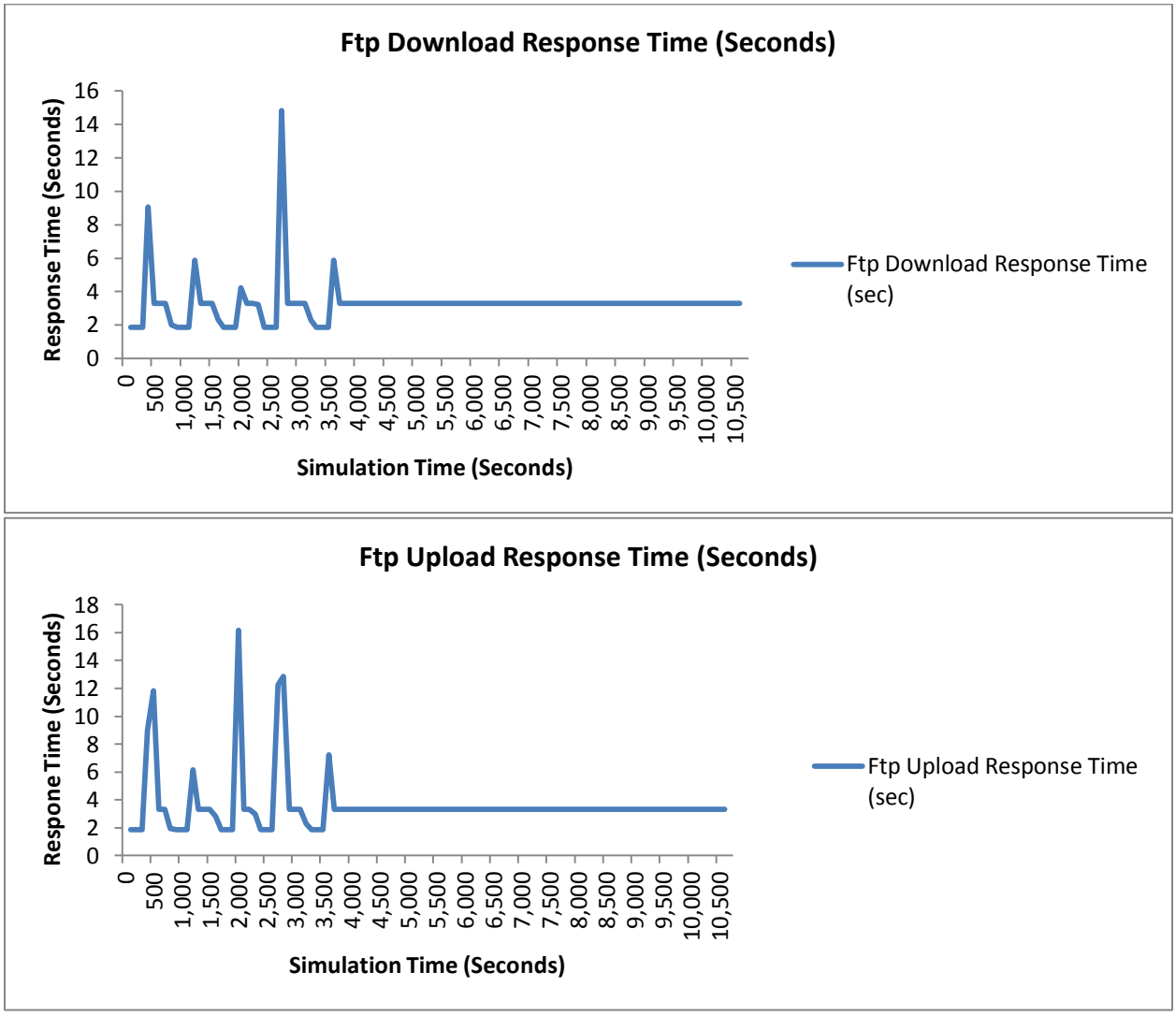


Figure 68 FTP Download/Upload Response Time (Simulation 2)

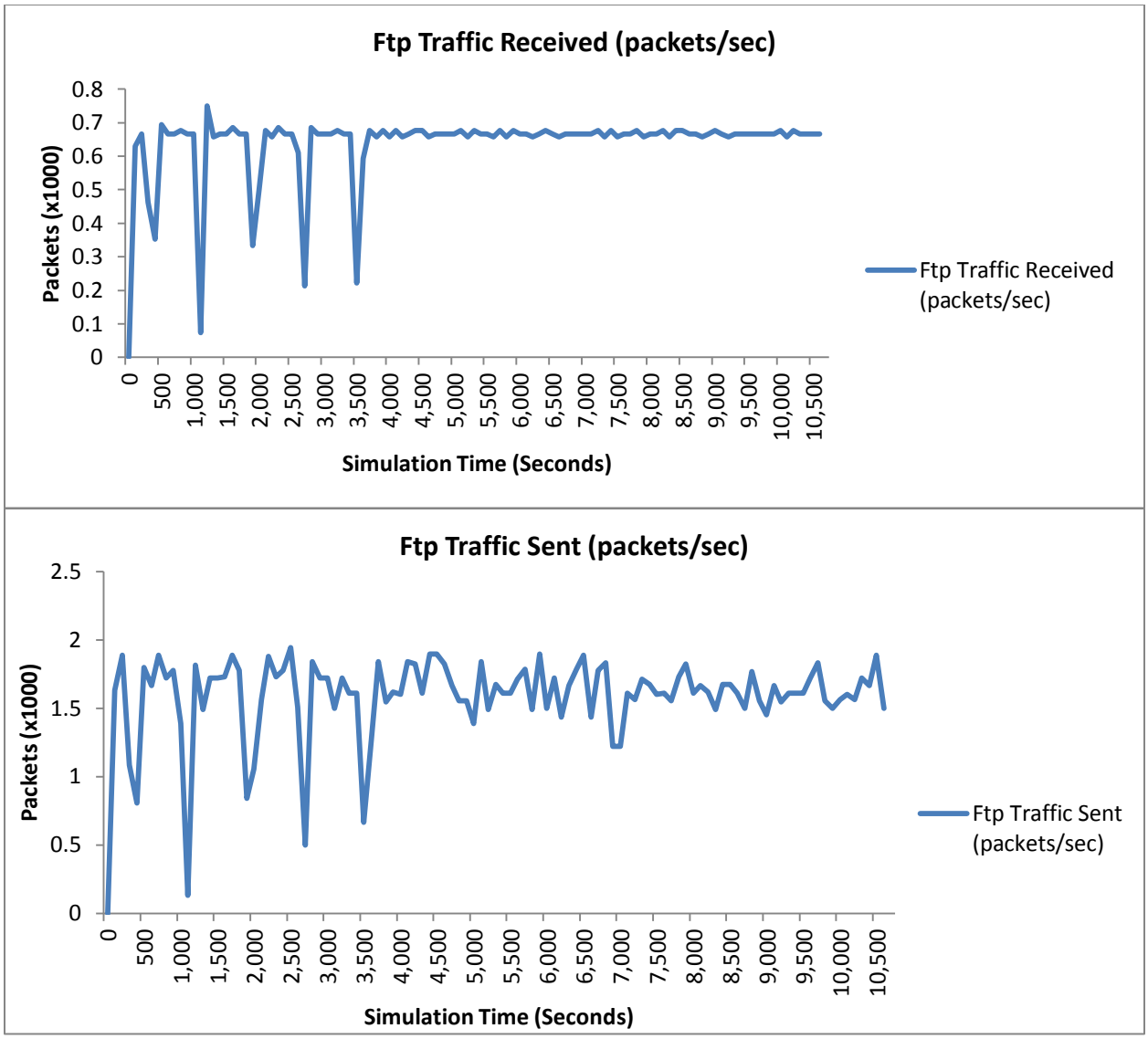


Figure 69 FTP Traffic Sent/Received (Simulation 2)

A.2.2 HTTP

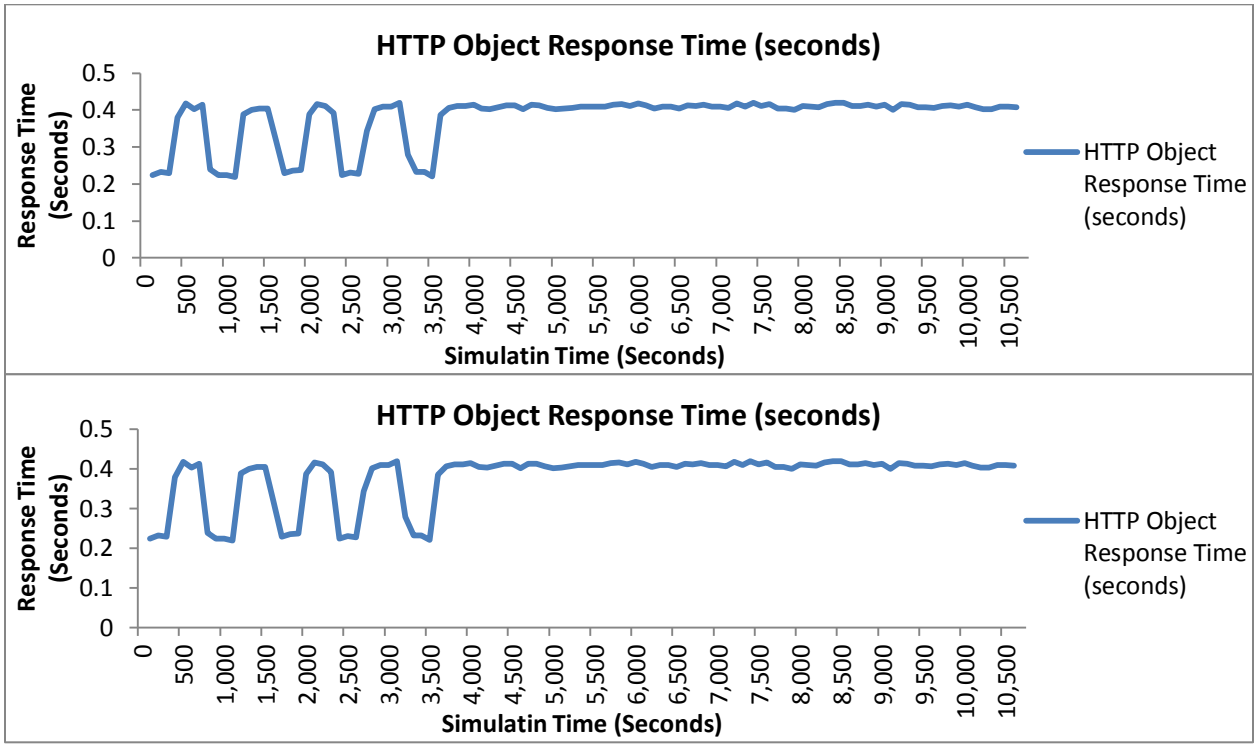


Figure 70 HTTP Object Response/Page Response Time (Simulation 2)

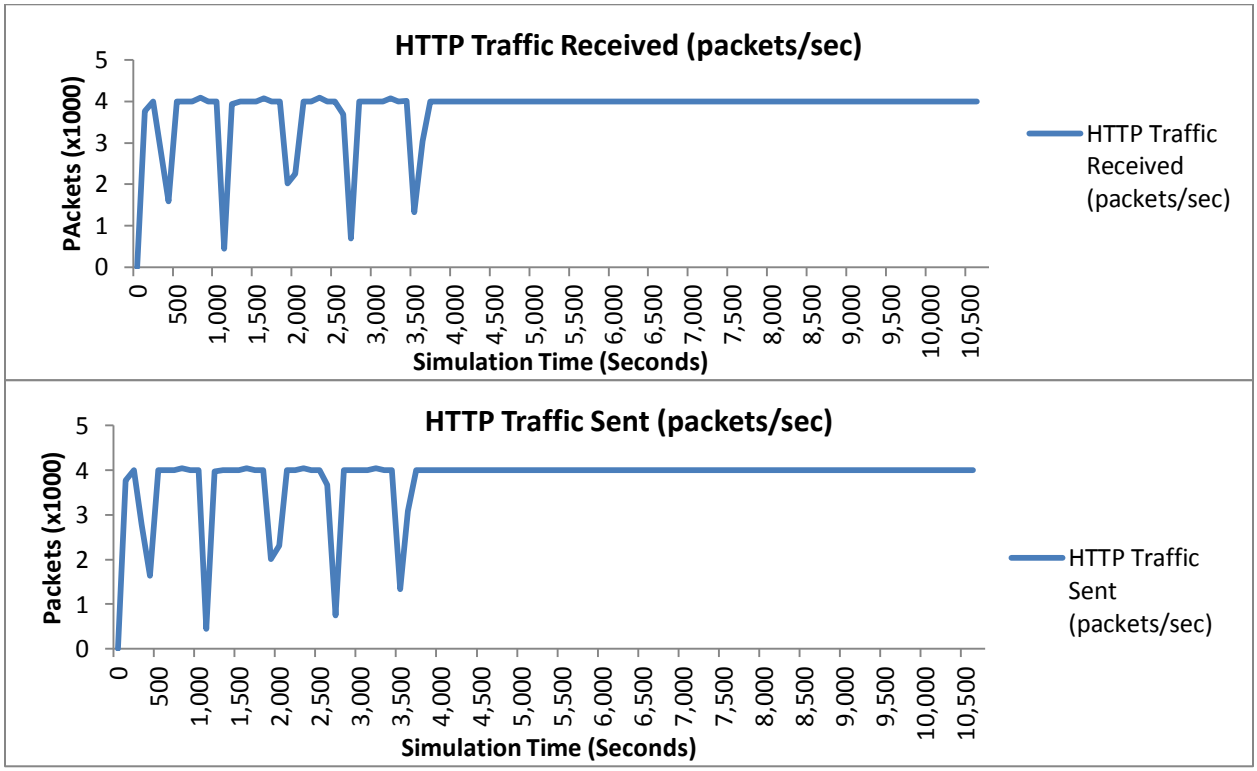


Figure 71 HTTP Traffic Sent/Received (Simulation 2)

A.2.3 Links Utilization

The below link traffic shows how this link between AS7 and AS9 is always being used but in cases of network convergence, traffic is affected. AS9 would send less traffic during convergence for a brief time, and AS7 would noticeably send lower traffic because it is sending only one way communication (from Node4 – Node 2) at time of convergence. Replies for the requests initiated from Node 4 are being dropped at AS5. On the contrary, link between AS7 and AS4, is not used during periods of failure and after 3600 seconds.

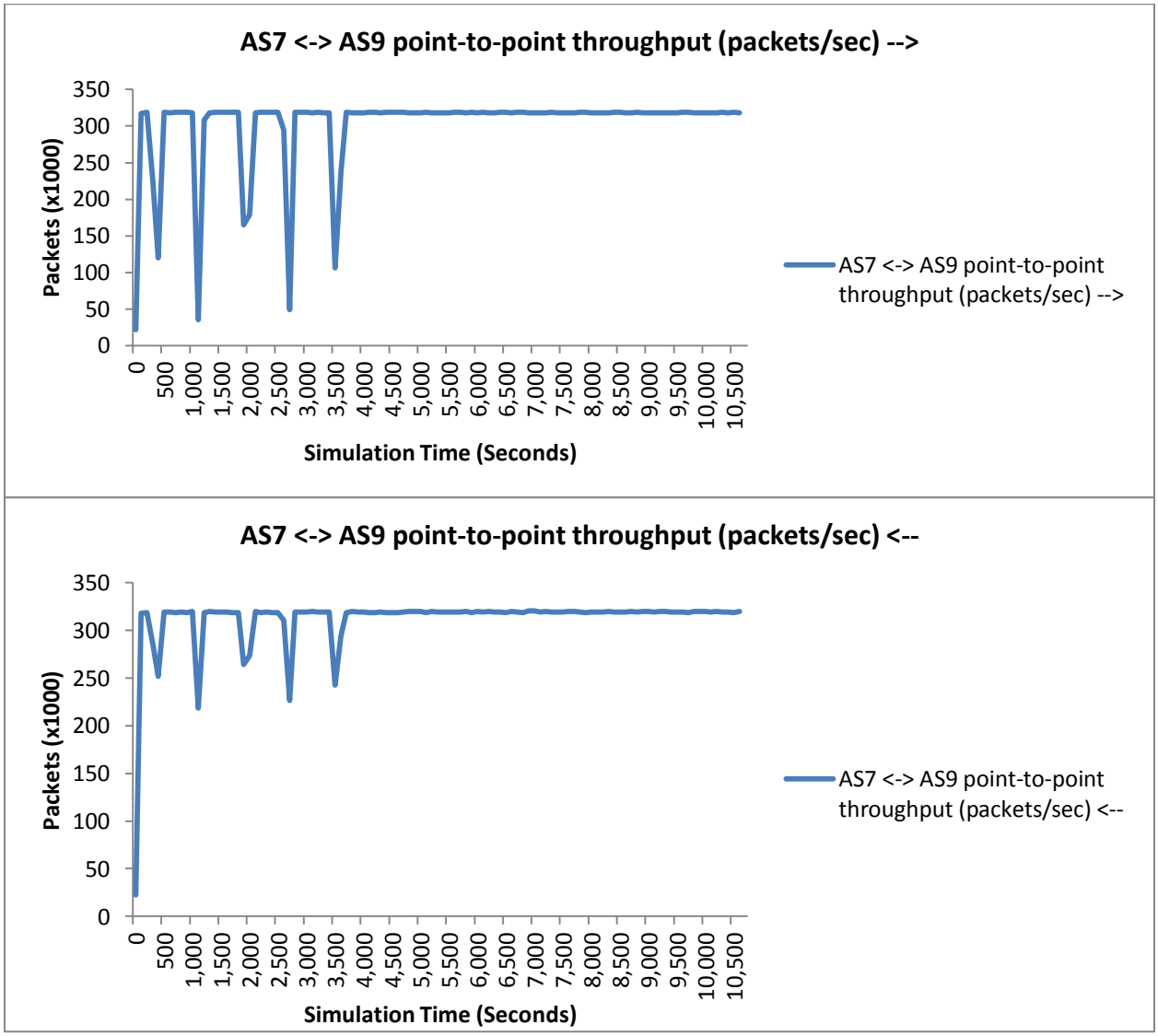


Figure 72 Link AS7-AS9 Utilization (Simulation 2)

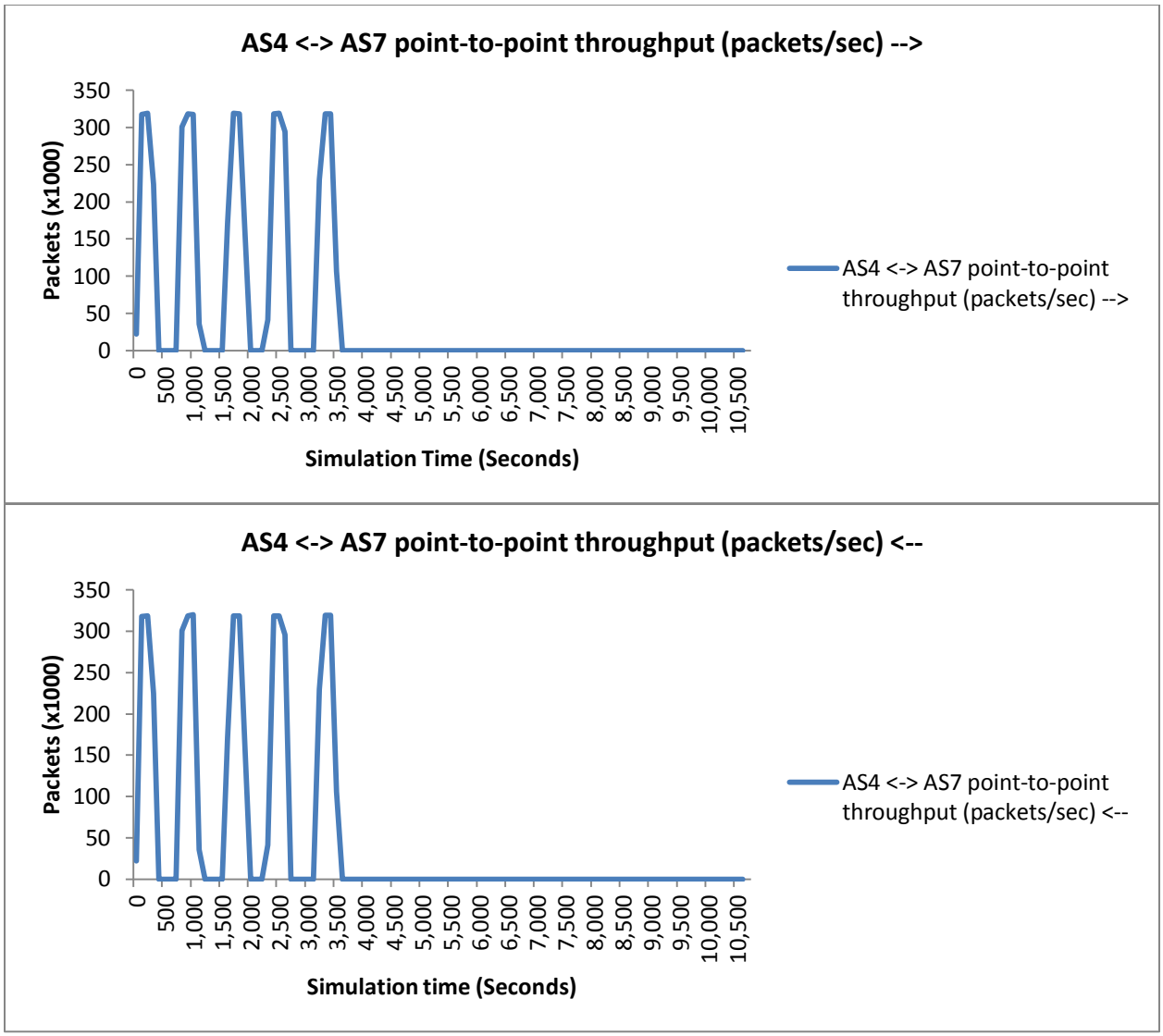


Figure 73 Link AS7-AS4 Utilization (Simulation 2)

However, link AS7-AS8 link is up all the time but less preferred by BGP. Being used after the flapping link is totally down. Also, being used after 1 hour due to the permanent failure.

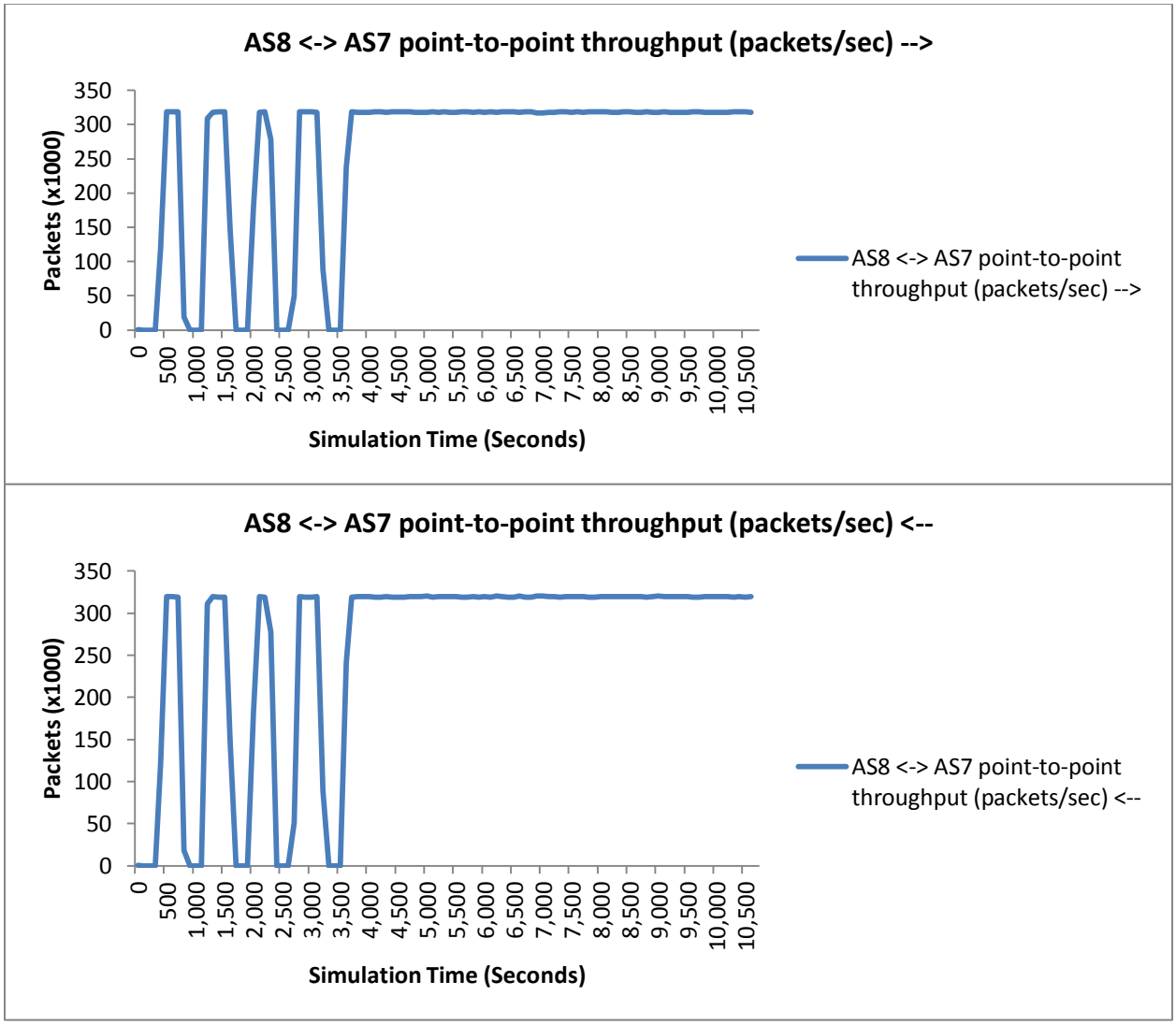


Figure 74 Link AS7-AS8 Utilization (Simulation 2)

A.3 Simulation 3 Results

A.3.1 FTP

Like Email traffic, FTP is experiencing a slightly lower delay due to this stabilization most of the time. At some points like 3600 seconds, the delay is more due to the congestion of more connections being open and transmitted. Remember that AS9 now is sending more traffic because it is not going through convergence after 800 seconds. Traffic is picking up after the 800 second where it is approaching normal traffic limits.

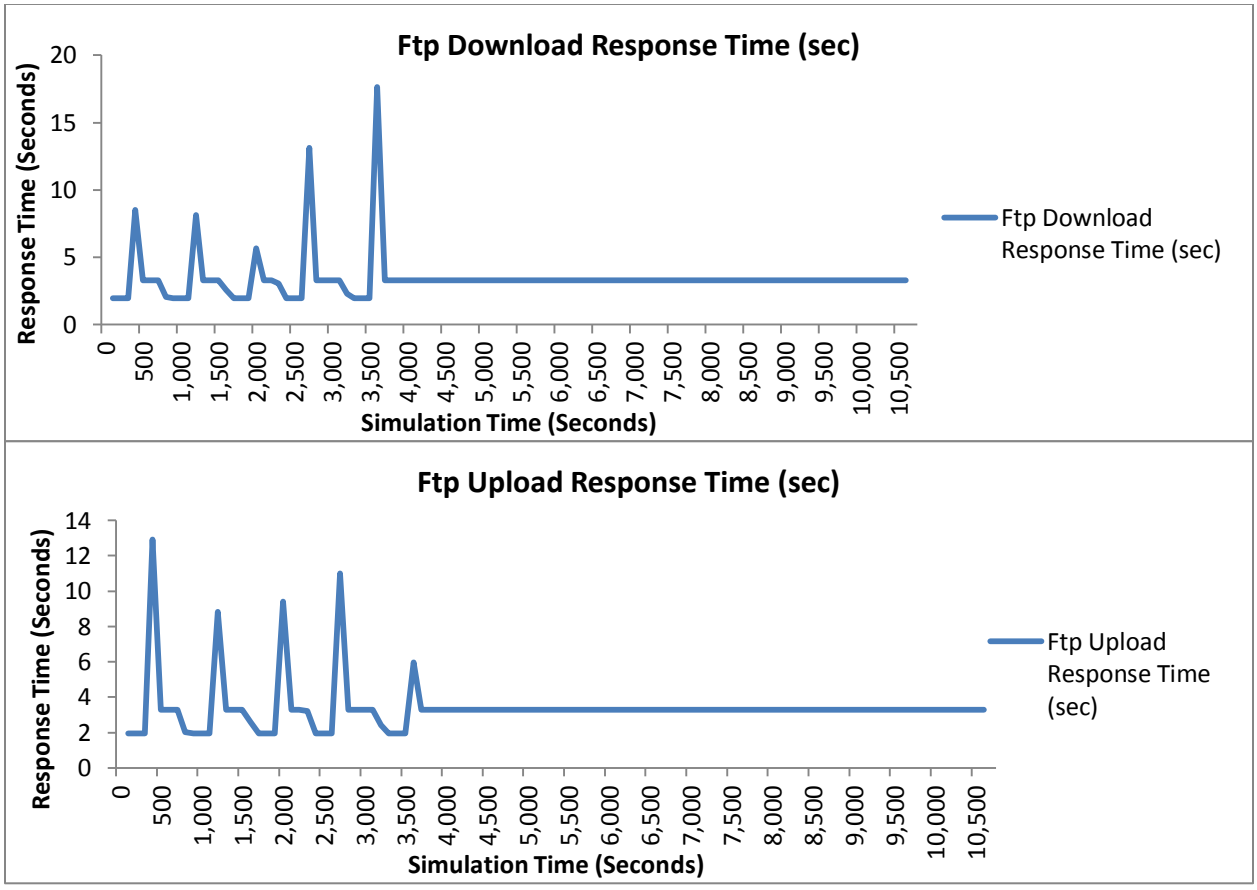


Figure 75 FTP Download/Upload Response Time (Simulation 3)

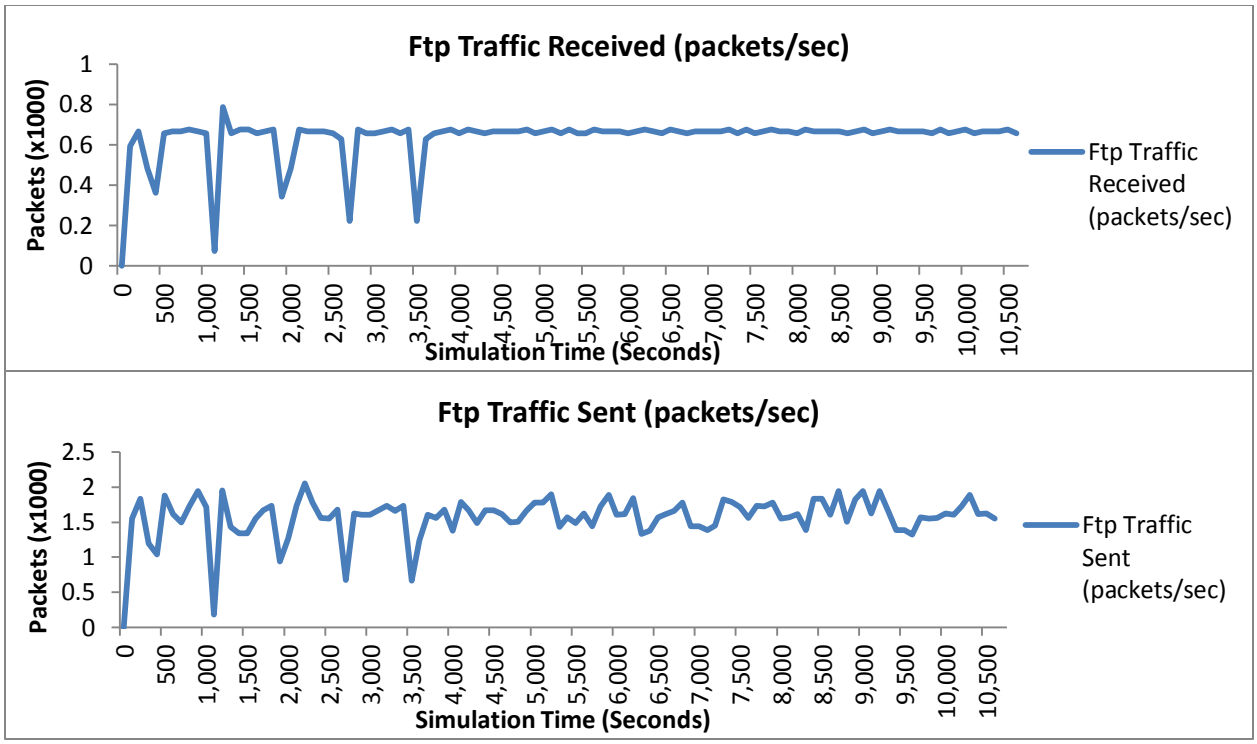


Figure 76 FTP Traffic Sent/Received (Simulation 3)

A.3.2 HTTP

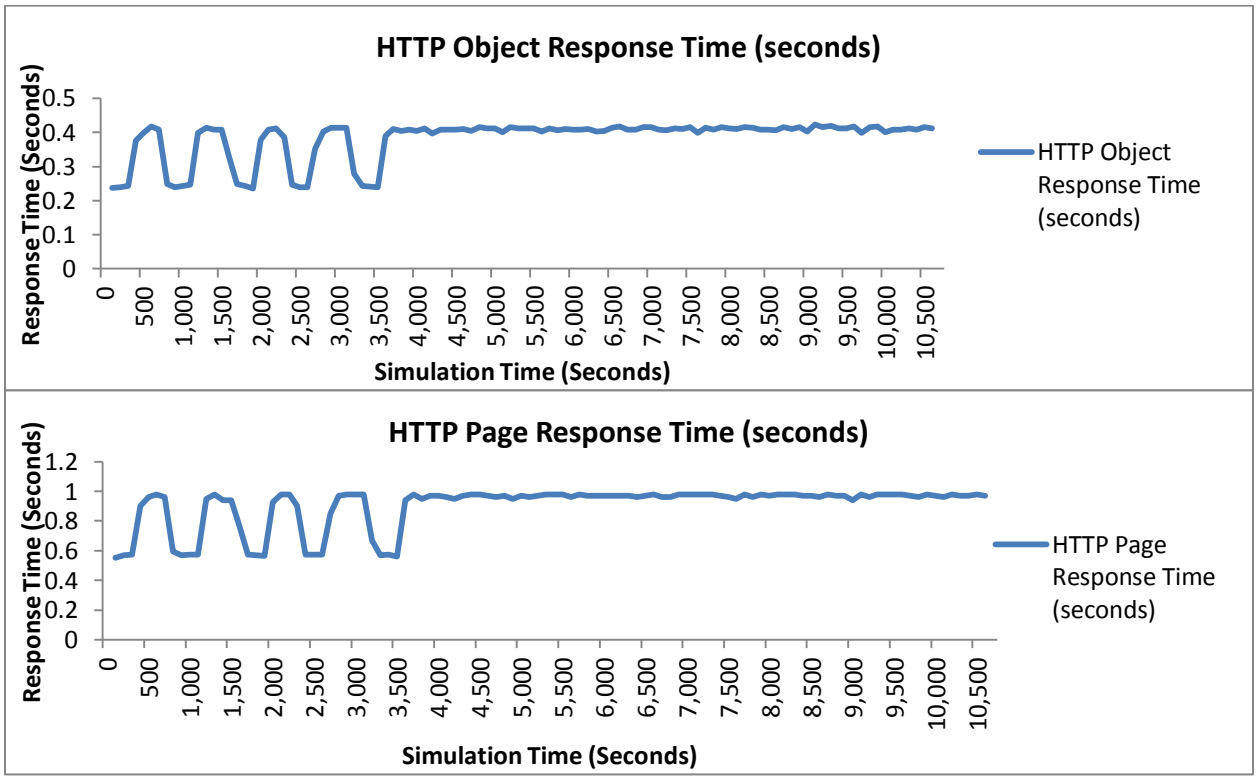


Figure 77 HTTP Object Response/Page Response Time (Simulation 3)

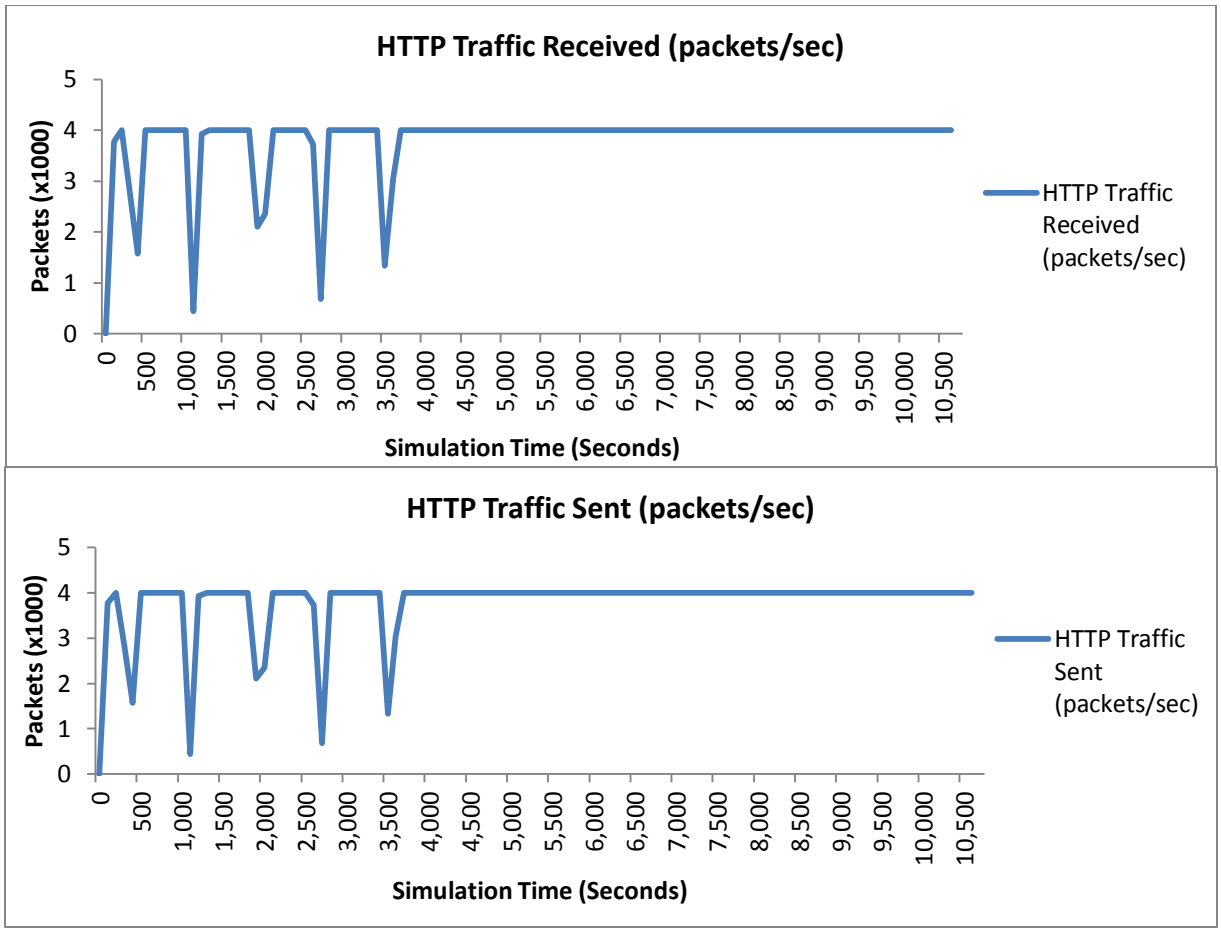


Figure 78 HTTP Traffic Sent/Received (Simulation 3)

A.3.3 Link Utilizations

The below link traffic shows how this link between AS7 and AS9 is always being used but in cases of network convergence that AS7 go through, traffic is affected. On the contrary, link between AS7 and AS4, is not used during periods of failure and after 3700 seconds. The utilization is still the same as the previous simulation since all traffic is bi-directional, and when drops happen, TCP connections from the server throttle down and less traffic is being sent from AS9 to AS7.

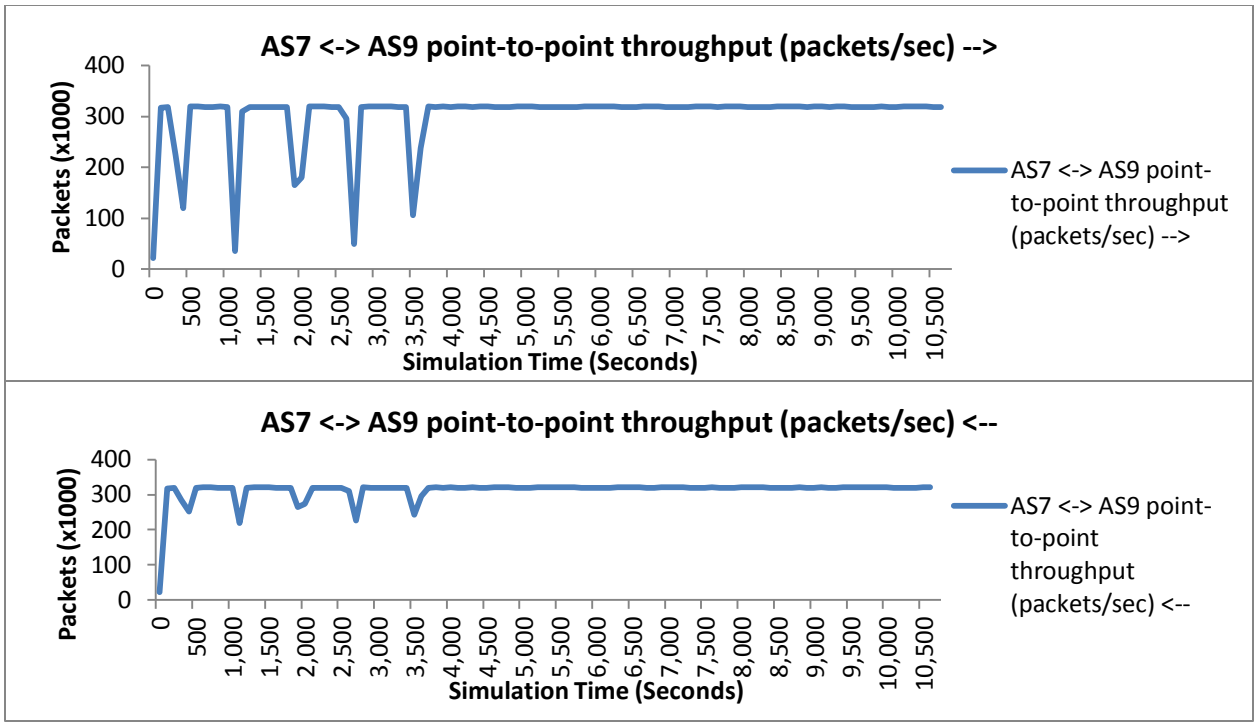


Figure 79 Link AS7-AS9 Utilization (Simulation 3)

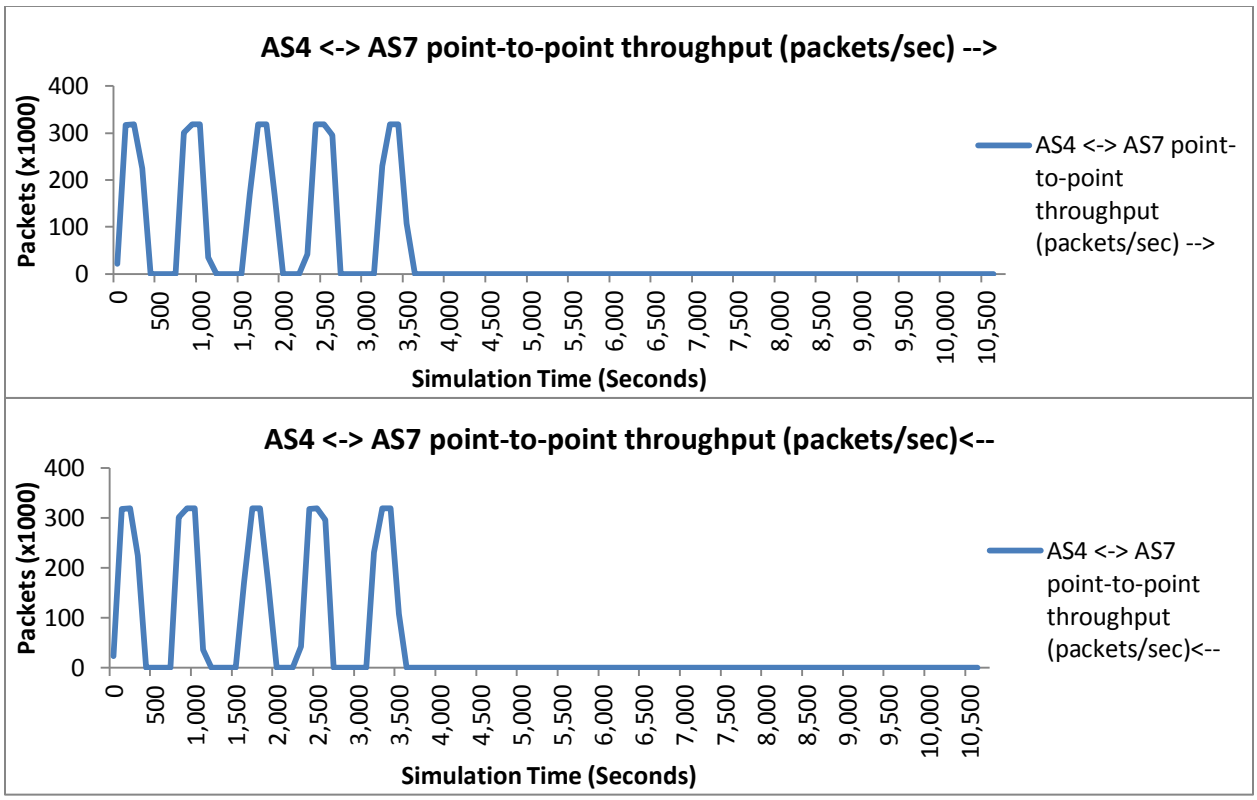


Figure 80 Link AS7-AS4 Utilization (Simulation 3)

As expected, the link between AS7-AS8 is being utilized fully after the 3700 seconds. Before that, the link is intermittently used because of AS7 instability

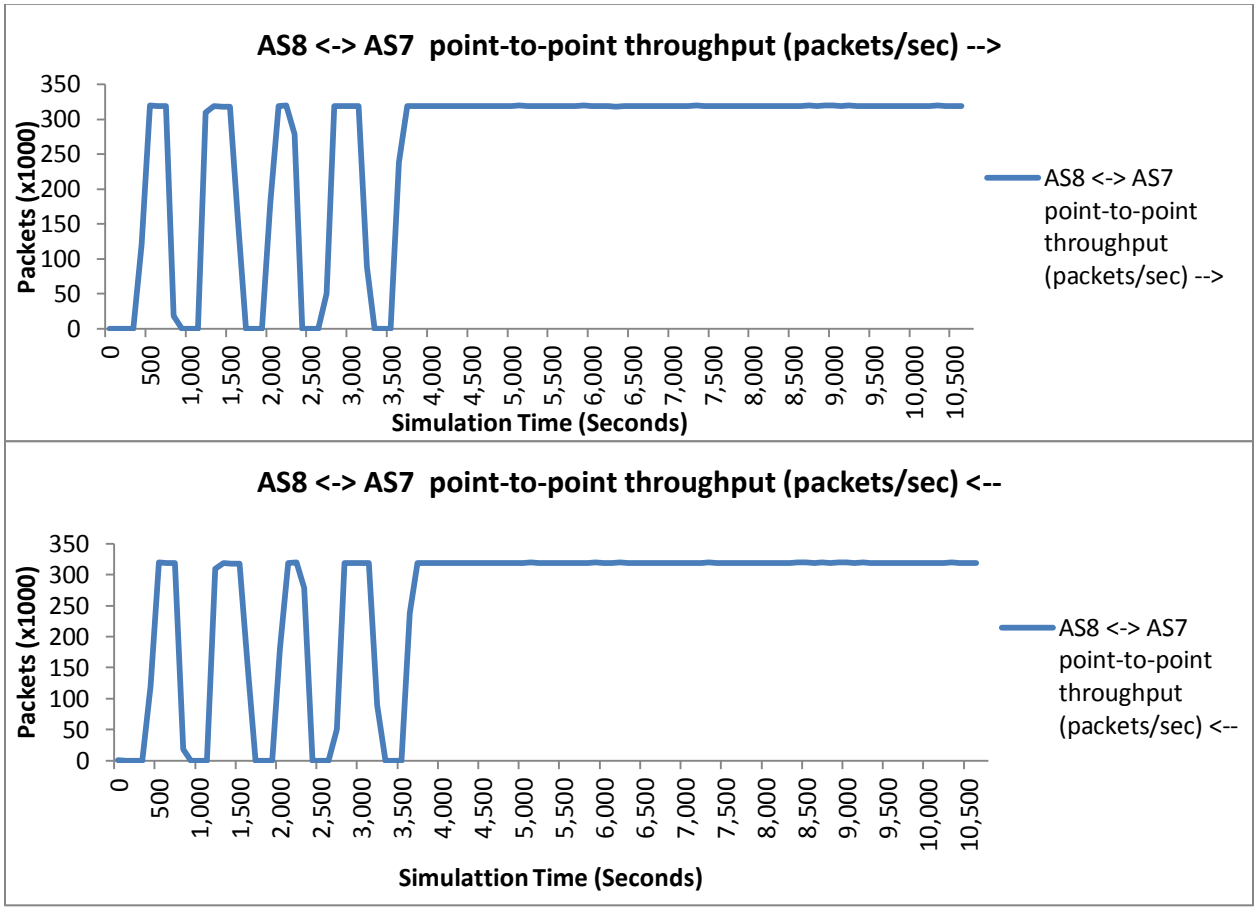


Figure 81 Link AS7-AS8 Utilization (Simulation 3)

A.4 Simulation 5 Results

A.4.1 FTP

Like Email traffic, we only see a brief period of high delay and low traffic during the first exploration. Our algorithm is successfully stabilizing the network for the rest of the simulation.

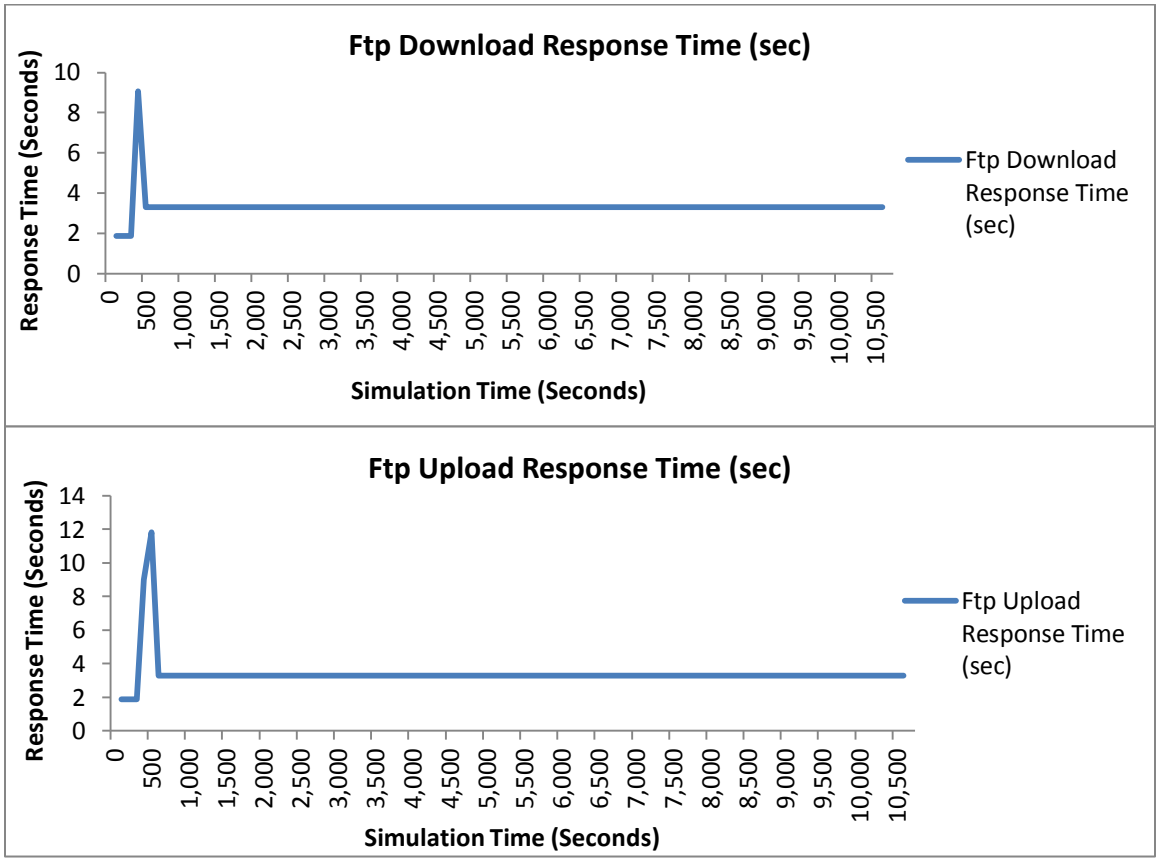


Figure 82 FTP Download/Upload Response Time (Simulation 5)

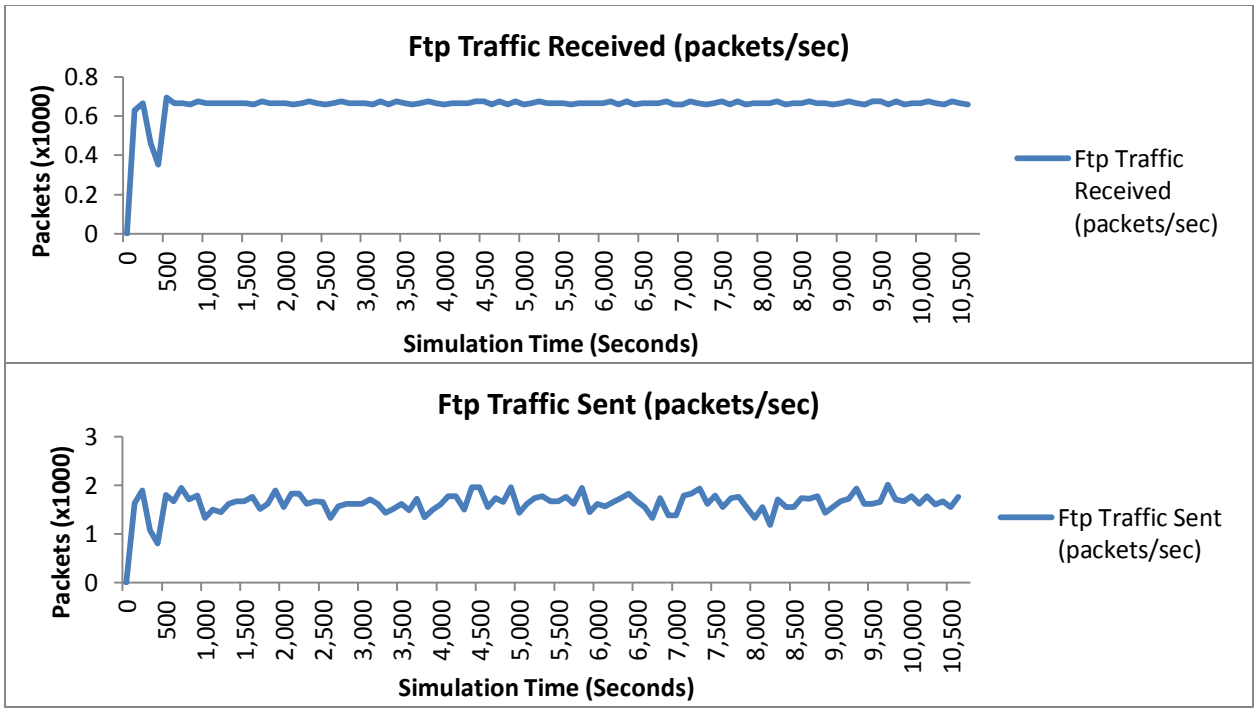


Figure 83 FTP Traffic Sent/Received (Simulation 5)

A.4.2 HTTP

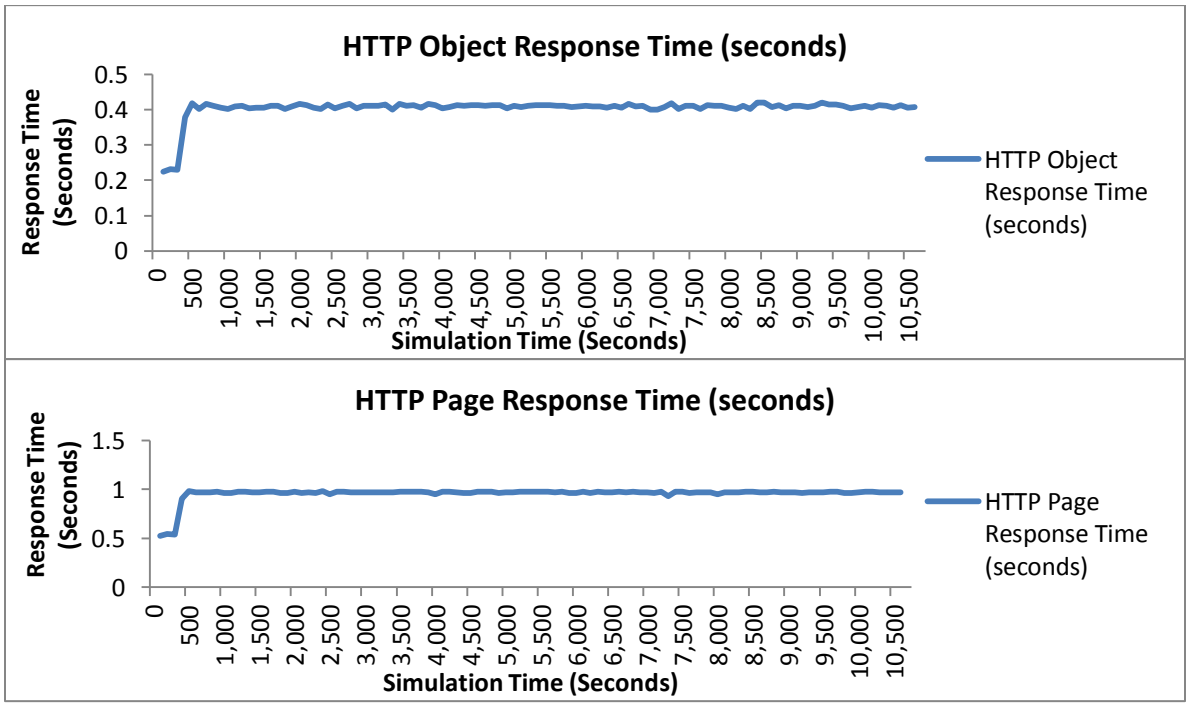


Figure 84 HTTP Object Response/Page Response Time (Simulation 5)

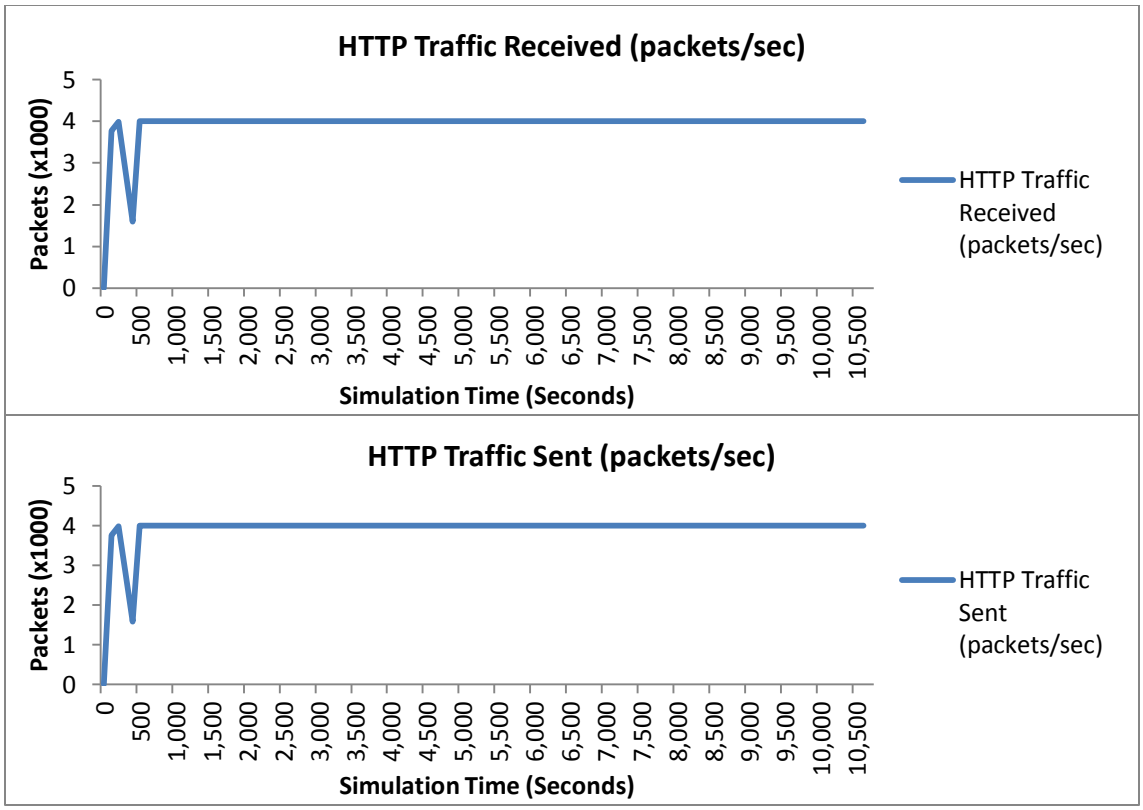


Figure 85 HTTP Traffic Sent/Received (Simulation 5)

A.4.3 Link Utilizations

We can see that the fluctuation of link utilization is now limited to the period 400-500 seconds. The link AS7-AS8 is the path utilized after the 500th second where our algorithm has enforced using it due to its stability.

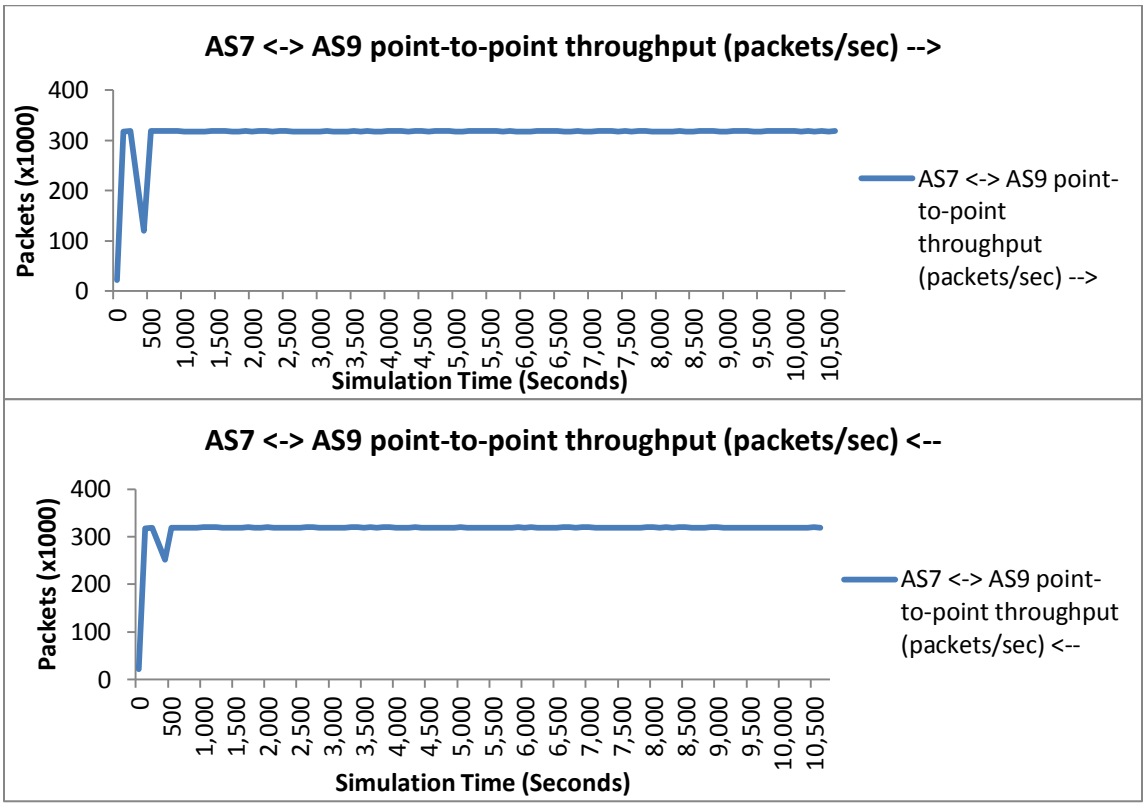


Figure 86 Link AS7-AS9 Utilization (Simulation 5)

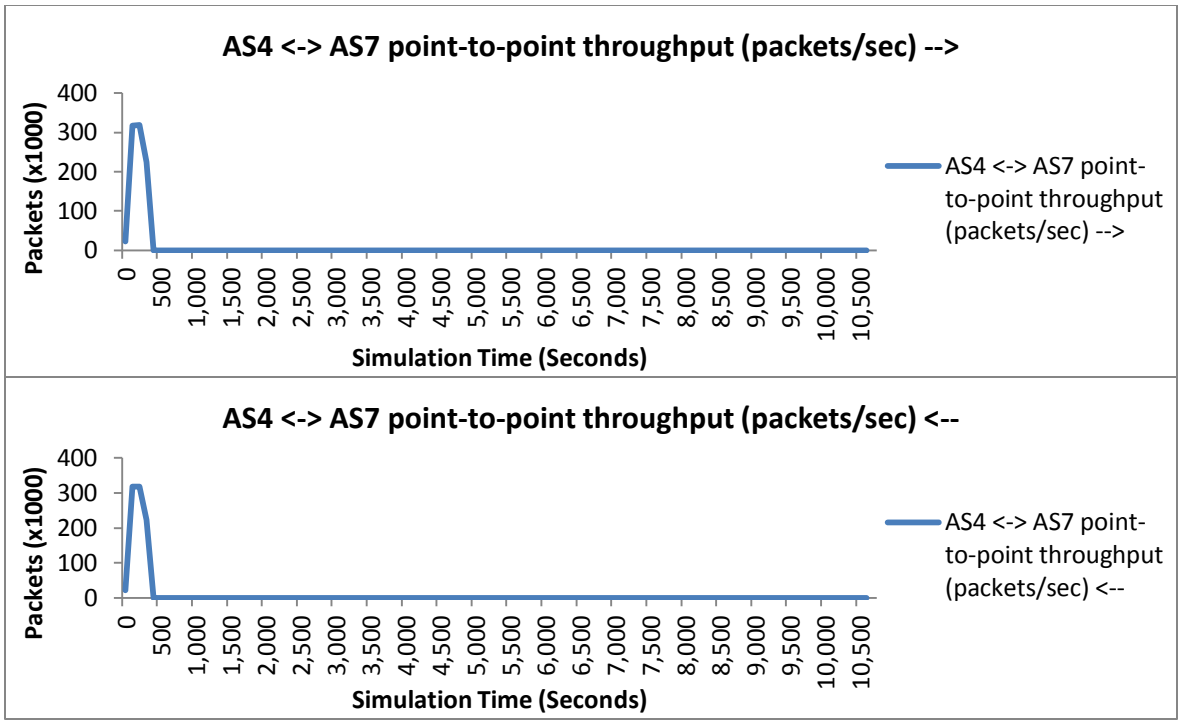


Figure 87 Link AS7-AS4 Utilization (Simulation 5)

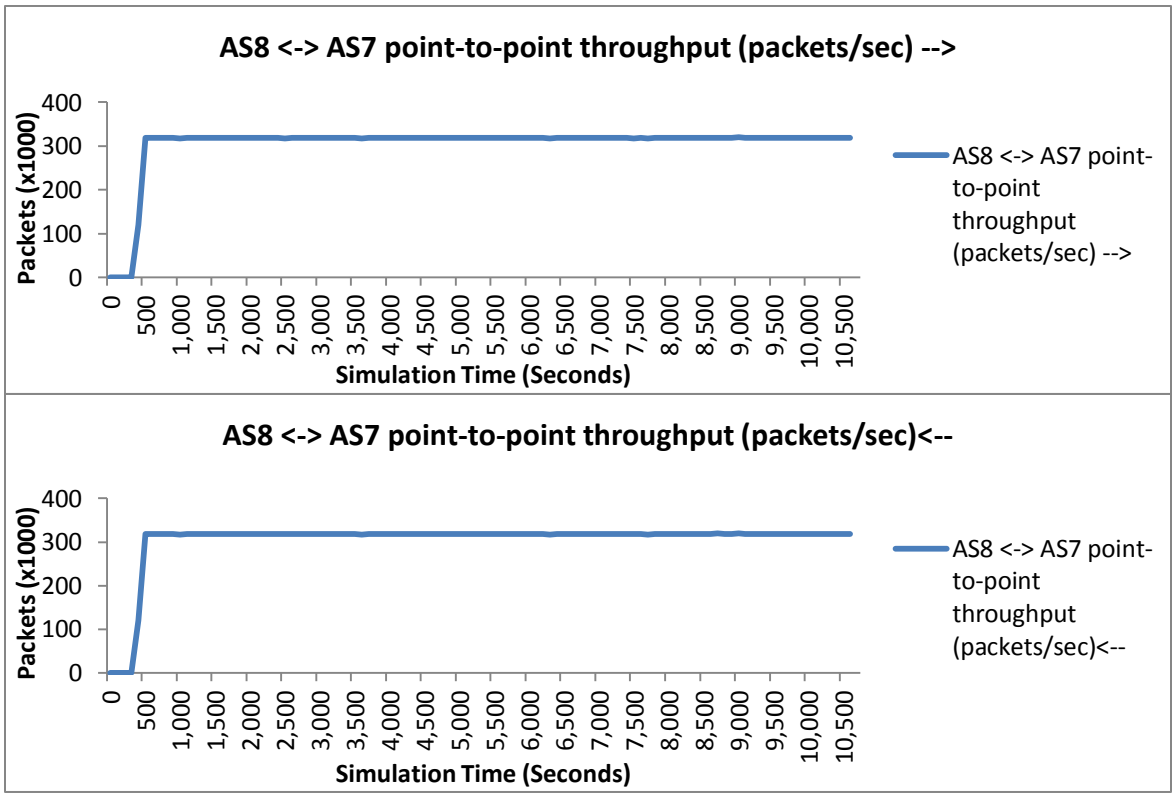


Figure 88 Link AS7-AS8 Utilization (Simulation 5)

Appendix B

C Code Implementation

B.1 Modified bgp_conn_update_message_handle()code

```
static void
bgp_conn_update_message_handle (Packet* data_pkptr)
        {
        List*                           reachability_info_ptr = OPC_NIL;
        List*                           withdrawn_routes_ptr = OPC_NIL;
        BgpT_Path_Attrs*                path_attributes_ptr = OPC_NIL;
        int                             i, num_prefixes;
        BgpT_Mp_Prefix*                 ith_prefix_ptr;

        FIN (bgp_conn_update_message_handle (data_pkptr))

        /*********************************************************/
        /* Multiprotocol NLRI are actually present inside the    */
        /* path attributes, since they cannot be carried in  the */
        /* NLRI field in the real world. Move all the reachable  */
        /* and unreachable routes into a single list for easy    */
        /* processing.                                           */
        */
        /*********************************************************/

bgp_support_get_info_from_update (data_pkptr, &withdrawn_routes_ptr, &reachability_info_ptr,
&path_attributes_ptr);


        /* creat a new history list for the first time.*/

if ( node_id == 8) {
if (First_time && !bgp_conn_update_loop_check (path_attributes_ptr)){
if (OPC_NIL != withdrawn_routes_ptr && OPC_NIL != reachability_info_ptr){
AS_mp_withdrawn_list_copy (MY_LIST, withdrawn_routes_ptr);
AS_mp_reachability_list_copy (MY_LIST, reachability_info_ptr);
}
else
  if (OPC_NIL != withdrawn_routes_ptr && OPC_NIL == reachability_info_ptr) {
  AS_mp_withdrawn_list_copy (MY_LIST, withdrawn_routes_ptr);
  }
```

```
  else
  if (OPC_NIL == withdrawn_routes_ptr && OPC_NIL != reachability_info_ptr){
  AS_mp_reachability_list_copy (MY_LIST, reachability_info_ptr);
  }
   }
}
        /* TODO: change to fd from nfd                                  */

        /* Step 1: Process unreachable destinations.                    */
        if (OPC_NIL != withdrawn_routes_ptr)
                {
                /* If peers advertise/withdraw host routes for locally connected        */
                /* interfaces, we must ignore the the advertisement. This is done       */
                /* in the real world.
                                */
                bgp_conn_remove_local_intf_updates (withdrawn_routes_ptr);

                if ( node_id == 8)
                        bgp_conn_remove_local_intf_updates(MY_LIST);

                /* If no valid withdrawn routes exist after the above cleanup,          */
                /* there is nothing else to do.
                        */
                if (op_prg_list_size (withdrawn_routes_ptr) == 0)
                        {
                        op_prg_mem_free (withdrawn_routes_ptr);
                        conn_info_ptr->unreachable_rte_exists = OPC_FALSE;
                        }
                else
                        {

                        if (node_id == 8 ){
                        if(!First_time)
                                AS_update_mylist_withdraw(MY_LIST, withdrawn_routes_ptr);
                        }

                        bgp_support_unreachable_routes_withdraw (bgp_my_adj_rib_in_ptr,
withdrawn_routes_ptr);

                        /* Place the list in connection info so that parent can update  */
                        /* the local BGP routing table (the local RIB).                 */
                        conn_info_ptr->unreachable_rte_list_ptr = withdrawn_routes_ptr;
                        conn_info_ptr->unreachable_rte_exists = OPC_TRUE;
                        }
                }
        else
                {
                conn_info_ptr->unreachable_rte_exists = OPC_FALSE;
```

```
        }

/* STEP 2: Process the reachability information, update  */
/* the Adj-RIB-In and run the decision process if reqd.   */
/* Read in the Network Layer reachability Information     */
/* and the Path Attributes.                               */
if (OPC_NIL != reachability_info_ptr)
        {
        /* If peers advertise/withdraw host routes for locally connected        */
        /* interfaces, we must ignore the the advertisement. This is done       */
        /* in the real world.
                        */
        bgp_conn_remove_local_intf_updates (reachability_info_ptr);
        if ( node_id==8 )
                bgp_conn_remove_local_intf_updates(MY_LIST);
        /* Check if any routes remain after the check for updates relating      */
        /* to local interfaces.
                        */
        if (op_prg_list_size (reachability_info_ptr) == 0)
                {
                op_prg_mem_free (reachability_info_ptr);
                bgp_support_path_attrs_destroy (path_attributes_ptr);
                if ( node_id == 8)
                        First_time = OPC_FALSE;
                FOUT;
                }

        if (node_id==8){
                if(!First_time && !bgp_conn_update_loop_check (path_attributes_ptr))
                AS_update_mylist_reachability(MY_LIST, reachability_info_ptr);
        }
        /* Updates coming from EBGP connections may not have*/
        /* some attributes like MED and LocPref. This is        */
        /* indicated by a special unset value. These values     */
        /* must be changed to default for use in the route      */
        /* selection process. The unset value is used only to   */
        /* determine the size of the update.                    */
        if (BGPC_LOCAL_PREF_UNSET == path_attributes_ptr->local_pref)
                path_attributes_ptr->local_pref = BGPC_DEFAULT_LOCAL_PREF;

        if (BGPC_MULTI_EXIT_DISC_UNSET == path_attributes_ptr->multi_exit_disc)
                path_attributes_ptr->multi_exit_disc = BGPC_DEFAULT_MULTI_EXIT_DISC;

        /* First check if we should ignore the advertised */
        /* routes. Reasons why we might ignore the routes:*/
        /* 1) The local AS number is in the as path.    */
        /* 2) This router is part of a confederation and the*/
        /*    confederation ID is in the as path.       */
```

```
            /* 3) This router is a route reflector and the       */
            /*    originator_id in the path attributes is its    */
            /*    own router ID.                                 */
            /* 4) This router is a route reflector and the       */
            /*    cluster list contains the local cluster ID.    */
            if (bgp_conn_update_loop_check (path_attributes_ptr))
                    {
                    /* We are ignoring these routes because of one */
                    /* or more of the reasons listed above.        */
                    /* But if any of the advertised routes were    */
                    /* previously advertised by this neighbor we   */
                    /* must consider them to have been withdrawn.*/
                    /* Check for this condition. Such routes will  */
                    /* be added to the list of withdrawn routes.   */
                    num_prefixes = op_prg_list_size (reachability_info_ptr);

                    for (i = 0; i < num_prefixes; i++)
                            {
                            /* Get the ith entry in the list.                */
            ith_prefix_ptr = (BgpT_Mp_Prefix *) op_prg_list_access (reachability_info_ptr, i);
                            /* Call the function that would add this  */
                            /* prefix to the list of withdrawn routes */
                            bgp_conn_previously_advertised_route_check (ith_prefix_ptr);
                            }

                    /* Free up the memory used for the NLRI and its*/
                    /* path attributes.                            */
                    bgp_support_rte_list_destroy (reachability_info_ptr);
                    bgp_support_path_attrs_destroy (path_attributes_ptr);

                    FOUT;
                    }
            /* If it is an EBGP connection, reset the LocPref.     */
            /* Local preference must not cross AS boundaries.      */
            if (BgpC_Conn_Type_Ebgp == conn_info_ptr->bgp_connection_type)
                    path_attributes_ptr->local_pref = BGPC_DEFAULT_LOCAL_PREF;

            /* Handle the information appropriately.                   */
            bgp_conn_reachability_info_process (reachability_info_ptr, path_attributes_ptr);

            /* Free up the memory used for the NLRI. Do not free path attributes     */
            /* as they are being used in the local route table.                      */
            bgp_support_rte_list_destroy (reachability_info_ptr);
            }
    else
            {
            conn_info_ptr->adj_rib_in_ptr->num_new_routes = 0;
            }
```

```
        if ( node_id == 8)
                First_time = OPC_FALSE;
        FOUT;
        }
```

```
/* The following  function to create history list for the first time*/

static void AS_mp_withdrawn_list_copy (List* to_list_ptr, List* from_list_ptr)
        {
        BgpT_Mp_Prefix                  *orig_prefix_ptr, *copy_prefix_ptr;
        int                              num_prefixes, ith_prefix;

        FIN (AS_mp_withdrawn_list_copy ());

        num_prefixes = op_prg_list_size (from_list_ptr);
        for (ith_prefix = 0; ith_prefix < num_prefixes; ith_prefix++)
                {
                orig_prefix_ptr = (BgpT_Mp_Prefix *) op_prg_list_access (from_list_ptr, ith_prefix);
                copy_prefix_ptr = bgp_support_mp_address_copy (orig_prefix_ptr);
                copy_prefix_ptr->penalty = orig_prefix_ptr ->penalty + 500;
                op_prg_list_insert (to_list_ptr, copy_prefix_ptr, OPC_LISTPOS_TAIL);
                }
        FOUT;
        }
```

```
/* The following  function to create history list for the first time*/

static void AS_mp_reachability_list_copy (List* to_list_ptr, List* from_list_ptr)
        {
        BgpT_Mp_Prefix                  *orig_prefix_ptr, *copy_prefix_ptr;
        int                              num_prefixes, ith_prefix;

        FIN (AS_mp_reachability_list_copy ());

        num_prefixes = op_prg_list_size (from_list_ptr);
        for (ith_prefix = 0; ith_prefix < num_prefixes; ith_prefix++)
                {
                orig_prefix_ptr = (BgpT_Mp_Prefix *) op_prg_list_access (from_list_ptr, ith_prefix);
                copy_prefix_ptr = bgp_support_mp_address_copy (orig_prefix_ptr);
                copy_prefix_ptr->penalty = orig_prefix_ptr ->penalty + 1000;
                op_prg_list_insert (to_list_ptr, copy_prefix_ptr, OPC_LISTPOS_TAIL);
                }
        FOUT;
        }
```

```
/* This function updates history list with withdrawn routes and penalties */

static void AS_update_mylist_withdraw (List* to_list_ptr, List* from_list_ptr){

int                                 num_withdrawn_routes, j_count;
```

```
int                                 num_penalty_routes, i_count;
BgpT_Mp_Prefix                      *jth_withdrawn_rte_ptr, *ith_penalty_rte_ptr;

FIN (AS_update_mylist_ withdraw ());

          num_withdrawn_routes = op_prg_list_size (from_list_ptr);
   num_penalty_routes = op_prg_list_size (to_list_ptr);
          for (j_count = 0; j_count < num_withdrawn_routes; j_count ++)            {
               jth_withdrawn_rte_ptr = (BgpT_Mp_Prefix *) op_prg_list_access (from_list_ptr, j_count);
          for (i_count = 0; i_count < num_penalty_routes; i_count ++){
               ith_penalty_rte_ptr = (BgpT_Mp_Prefix *) op_prg_list_access (to_list_ptr, i_count);
                 if(bgp_support_mp_prefix_equal(jth_withdrawn_rte_ptr, ith_penalty_rte_ptr)){
                                    ith_penalty_rte_ptr->penalty = ith_penalty_rte_ptr->penalty + 500;
                                     if (ith_penalty_rte_ptr->penalty >= 2000){
                                                         ith_penalty_rte_ptr->cutoff = OPC_TRUE;
                                                         jth_withdrawn_rte_ptr ->cutoff = OPC_TRUE;

                                                                  }
                               }
                              }
 }
FOUT;
}
```

```
/* Function to update history lists with reachability routes and penalties*/

static void AS_update_mylist_reachability  (List* to_list_ptr, List* from_list_ptr){

int                                      num_reachability_routes, j_count;
int            num_penalty_routes, i_count;
BgpT_Mp_Prefix                      *jth_reachability_rte_ptr, *ith_penalty_rte_ptr;

FIN (AS_update_mylist_reachability());
   num_reachability_routes = op_prg_list_size (from_list_ptr);
   num_penalty_routes = op_prg_list_size (to_list_ptr);
          for (j_count = 0; j_count < num_reachability_routes; j_count ++)            {
                jth_reachability_rte_ptr = (BgpT_Mp_Prefix *) op_prg_list_access (from_list_ptr,
j_count);
          for (i_count = 0; i_count < num_penalty_routes; i_count ++){
                ith_penalty_rte_ptr = (BgpT_Mp_Prefix *) op_prg_list_access (to_list_ptr, i_count);
                  if(bgp_support_mp_prefix_equal(jth_reachability_rte_ptr, ith_penalty_rte_ptr)){
                        ith_penalty_rte_ptr->penalty = ith_penalty_rte_ptr->penalty + 1000;
                                     if (ith_penalty_rte_ptr->penalty >= 2000){
                                                         ith_penalty_rte_ptr->cutoff = OPC_TRUE;
                                                         jth_reachability_rte_ptr->cutoff = OPC_TRUE;
                                                                  }
                               }
}
                               }
```

```
FOUT;
}
```

```
static void
bgp_conn_reachability_info_process (List* nlri_route_ptr, BgpT_Path_Attrs* path_attrs_ptr)
	{
	int				i_count;
	int				j_count;
	int				num_penalty_routes;
	int				num_prefixes;
	Boolean				is_attrs_changed;
#ifndef OPD_NO_DEBUG
	Boolean				trace_bgp_active = OPC_FALSE;
	Boolean				trace_bgp_send_active = OPC_FALSE;
	Boolean				trace_bgp_receive_active = OPC_FALSE;
	char				msg_str [128];
	char				mp_prefix_str [128];
#endif
	BgpT_Mp_Prefix*			ith_prefix_ptr;
	BgpT_Mp_Prefix*			copy_prefix_ptr;
	BgpT_Rte_Entry*			new_rte_entry_ptr;
	BgpT_Addr_Family	addr_family;
	BgpT_Mp_Prefix		*jth_penalty_rte_ptr ;

	/** A new route will be formed out of the newly received  **/
	/** reachability information. This route will be passed	**/
	/** through routing policies and if selected will be	**/
	/** added to Adj-RIB-In of this connection process.	**/
	FIN (bgp_conn_reachability_info_process (nlri_route_ptr, path_attrs_ptr))

#ifndef OPD_NO_DEBUG
	/* Find out if the simulation is being run with a trace	*/
	/* on BGP.
		*/
	if (debug_active)
		{
		trace_bgp_active = op_prg_odb_ltrace_active ("bgp");
		trace_bgp_send_active = op_prg_odb_ltrace_active ("bgp_send");
		trace_bgp_receive_active = op_prg_odb_ltrace_active ("bgp_receive");
		}
#endif

	/* Maintain a separate temporary list of the new routes */
	/* added to Adj-RIB-In to be used and destroyed by the  */
	/* parent process.
	*/
	bgp_my_adj_rib_in_ptr->new_routes_lptr = op_prg_list_create ();

	/* Network reachability information is a list of		*/
```

```
        /* Prefixes. Run through this list and read the              */
        /* individual addresses.                                     */
        num_prefixes = op_prg_list_size (nlri_route_ptr);

        /* Access each prefix and pass it to the function that       */
        /* processes the route information.                          */
        for (i_count = 0; i_count < num_prefixes; i_count++)
                {
                ith_prefix_ptr = (BgpT_Mp_Prefix *)op_prg_list_access (nlri_route_ptr, i_count);

                /* Make a copy of the Prefix for use in the routing       */
                /* information bases.                                     */
                copy_prefix_ptr = (BgpT_Mp_Prefix*) bgp_support_mp_prefix_copy (ith_prefix_ptr);

#ifndef OPD_NO_DEBUG
                /* Print out the reachability information received        */
                /* as a trace message.                                                          */
                if (trace_bgp_active || trace_bgp_receive_active || bgp_support_prefix_trace_active
(bgp_support_ip_prefix_from_mp_prefix (copy_prefix_ptr)))
                        {
                        bgp_support_mp_prefix_print (mp_prefix_str, copy_prefix_ptr);
                        sprintf (msg_str, "Received reachability information for destination [%s].",

                mp_prefix_str);
                        op_prg_odb_print_major (proc_info_str, msg_str, OPC_NIL);
                        }
#endif

                /* Create a new route so that it can be added to          */
                /* the Routing Information Base. If this list has    */
                /* more than one prefix, note that the same path          */
                /* attrs ptr will be passed. The entry create             */
                /* function will automatically increment the no: of       */
                /* references to this path attrs.                         */
                /* For now set the admin weight on all routes to 0.       */
                /* It will be set appropriately in the Update_mesg        */
                /* enter execs of the bgp_process model.                  */
                new_rte_entry_ptr = (BgpT_Rte_Entry*) bgp_support_rte_entry_create
(copy_prefix_ptr, path_attrs_ptr, my_peer_id, 0);

                /* Check to see if the route has reached Cutoff Threshold, then we have to penalize
the RTE entry, We are using the Penalty list for this  */

if ( node_id == 8){
        num_penalty_routes = op_prg_list_size (MY_LIST);
        for (j_count = 0; j_count < num_penalty_routes; j_count ++){
            jth_penalty_rte_ptr = (BgpT_Mp_Prefix *) op_prg_list_access (MY_LIST , j_count);
            if(bgp_support_mp_prefix_equal(jth_penalty_rte_ptr, copy_prefix_ptr)){
```

```
                if (jth_penalty_rte_ptr -> cutoff)
                    {
                     new_rte_entry_ptr->Penalized =OPC_TRUE;
                     break;
                     }
                }
          }
}
                    /* Do not accept route if addr family is not supported     */
                    addr_family = copy_prefix_ptr->address_family_id;

                    if (OPC_NIL == conn_info_ptr->af_info_array [addr_family])
                              {
        bgp_support_rte_entry_destroy (new_rte_entry_ptr);

#ifndef OPD_NO_DEBUG
                              if (trace_bgp_active || trace_bgp_receive_active)
                                        {
                                        op_prg_odb_print_major ("Ignoring reachability information due to
unsupported address family.",
                                                                                  OPC_NIL);
                                        }
#endif
        continue;
                              }

                    /* Set the weight of the new route.                                  */
                    new_rte_entry_ptr->weight = conn_info_ptr->af_info_array [addr_family]->weight;

                    /* Apply the filters and routing policies applicable to the          */
                    /* receiving peer process. Set the connection type in the call       */
                    /* to None to indicate that we do not need to worry about the        */
                    /* well  known communities as this is an incoming route.                                */
          if (bgp_support_rte_filter_policy_apply (&new_rte_entry_ptr, conn_info_ptr->af_info_array
[addr_family]->incoming_update_pib,
                              conn_info_ptr->af_info_array [addr_family]->filter_opts_ptr->pre_filter_in,
conn_info_ptr->af_info_array [addr_family]->filter_opts_ptr->path_filter_in, OPC_TRUE,
&is_attrs_changed, BgpC_Conn_Type_None,
                              conn_info_ptr->local_info_ptr) == OPC_FALSE)
                              {
        /* This route should not be included. move on   */
        bgp_support_rte_entry_destroy (new_rte_entry_ptr);

                              /* We are ignoring this route because one of the*/
                              /* policies rejected it. But if was previously        */
                              /* advertised by this neighbor we must consider */
                              /* it to have been withdrawn. Check for this       */
                              /* condition. Such routes will be added to the     */
```

```
                        /* list of withdrawn routes.                    */
                        bgp_conn_previously_advertised_route_check (ith_prefix_ptr);

          /* Update the statistics that indicate the number   */
        /* routes that were dropped due to route policies.  */
        op_stat_write
(conn_info_ptr>local_info_ptr>bgp_local_stats.num_policy_discards_local_stat_hndl, 1.0);
      op_stat_write
 (conn_info_ptr->local_info_ptr->bgp_local_stats.num_policy_discards_local_stat_hndl, 0.0);

#ifndef OPD_NO_DEBUG
                        if (trace_bgp_active || trace_bgp_receive_active)
                                {
                                op_prg_odb_print_major ("Ignoring reachability information due to
local policies.",OPC_NIL);
                                }
#endif
        continue;
        }

                /* Set the AS number and BGP ID of the neighbor         */
                /* in the route entry. These values are used to    */
                /* break ties among routes of equal preference.         */
          new_rte_entry_ptr->advertising_rtr_bgp_id       = conn_info_ptr->neighbor_bgp_id;
          new_rte_entry_ptr->advertising_rtr_as_number = conn_info_ptr->neighbor_as_number;

                if (bgp_node_is_route_reflector (conn_info_ptr->local_info_ptr))
                        {
                if (bgp_conn_nbr_is_rrc (conn_info_ptr, addr_family))
                                {
                new_rte_entry_ptr->advertising_neighbor_type = BgpC_Neighbor_Type_Client;
                                }
                else
                new_rte_entry_ptr->advertising_neighbor_type = BgpC_Neighbor_Type_Non_Client;
                        }
                else
                new_rte_entry_ptr->advertising_neighbor_type = BgpC_Neighbor_Type_None;

                /* Pass this newly created route information to the      */
                /* the function that processes the route and makes      */
                /* required changes to the Adj-RIB-In.                  */
                bgp_conn_route_entry_process (new_rte_entry_ptr);

                /* Do this for all the entries in the reachability   */
                /* Information list.                                 */
                }

          /* Free the temporary list if there is no new routes in     */
```

```
        /* Adj-RIB-In.
        */
        if (bgp_my_adj_rib_in_ptr->num_new_routes == 0)
                op_prg_mem_free (bgp_my_adj_rib_in_ptr->new_routes_lptr);

        FOUT;
        }
```

```
static void
bgp_conn_route_entry_process (BgpT_Rte_Entry* new_rte_entry_ptr)
        {
        int                     route_position;
        FIN (bgp_conn_route_entry_process (new_rte_entry_ptr));

        /* STEP 1: Check if there already exists an entry in          */
        /* the Adj-RIB-In with the newly received dest addr.       */
        if (bgp_support_rte_entry_find (bgp_my_adj_rib_in_ptr, new_rte_entry_ptr-
>dest_prefix_ptr, &route_position) != OPC_NIL)
                {
                /* Delete the old entry.                                                */
                bgp_support_ith_rte_entry_delete (bgp_my_adj_rib_in_ptr, route_position);
                }

        /* When support for address aggregation is added the    */
        /* search should go on to see if there are any                */
        /* overlapping routes that exist.                                */

        /* Insert the new route entry into the RIB in and also      */
        /* into temporary new route list for the parent process.*/
        bgp_support_rte_entry_ith_place_insert (bgp_my_adj_rib_in_ptr, new_rte_entry_ptr,
route_position);
        /* Pass a copy to the new routes list because we need different routes in the local RIB and */
        /* RIB-Ins. In VRF scenarios, the RIB-In route (from CE) will be IPv4, whereas the local RIB*/
        /* will contain a VPNv4 route.                                                */
        op_prg_list_insert (bgp_my_adj_rib_in_ptr->new_routes_lptr,
                bgp_support_rte_entry_copy (new_rte_entry_ptr), OPC_LISTPOS_TAIL);
        /* Increment the number of new routes in the table.       */
        bgp_my_adj_rib_in_ptr->num_new_routes++;
        FOUT;
        }
```

```
BgpT_Rte_Entry*
bgp_support_rte_entry_copy (BgpT_Rte_Entry* orig_rte_entry)
        {
        BgpT_Rte_Entry*                 new_rte_entry_ptr;
        /** Makes a copy of the original route entry.                **/
        FIN (bgp_support_rte_entry_copy (orig_rte_entry))

        /* Allocate memory for the new route.                                */
```

```
        new_rte_entry_ptr = (BgpT_Rte_Entry*) op_prg_pmo_alloc (bgp_rte_entry_pmh);

        /* Set the number of references to the particular route */
       /* enrty to be 0.                                        */
  new_rte_entry_ptr->num_references = 0;

        /* I added this part to copy penalty information for the shared list of new entries*/

        new_rte_entry_ptr->Penalized = orig_rte_entry->Penalized;
        /* Copy the id of the peer that received this update    */
        new_rte_entry_ptr->local_adv_receive_peer_id = orig_rte_entry-
>local_adv_receive_peer_id;

        /* Copy the BGP ID of the neighbor originating this update    */
        new_rte_entry_ptr->advertising_rtr_bgp_id  = orig_rte_entry->advertising_rtr_bgp_id;

        /* Copy the AS number of the neighbor originating this update    */
        new_rte_entry_ptr->advertising_rtr_as_number = orig_rte_entry-
>advertising_rtr_as_number;

        /* Copy the neighbor_type.                                                    */
        new_rte_entry_ptr->advertising_neighbor_type = orig_rte_entry-
>advertising_neighbor_type;

        /* Copy the degree of preference for this route which     */
        /* be changed if required by other routers.                    */
        new_rte_entry_ptr->degree_of_preference = orig_rte_entry->degree_of_preference;

        /* Set the input prefix as the destination prefix ptr   */
       new_rte_entry_ptr->dest_prefix_ptr      = bgp_support_mp_prefix_copy ( orig_rte_entry-
>dest_prefix_ptr);

        /* Set the path attribtues                                                    */
        new_rte_entry_ptr->path_attrs_ptr       = bgp_support_path_attrs_copy (orig_rte_entry-
>path_attrs_ptr);

        new_rte_entry_ptr->weight = orig_rte_entry->weight;
        FRET (new_rte_entry_ptr);
        }
```

B.2 bgp_update_message_State_enter_exicutives code and Modified Selection Code

```
static void
bgp_unfeasible_routes_process (List* withdrawn_route_list_ptr, int withdrawing_peer_id)
        {
        int                     num_withdrwn_rte;
        int                     count_j;
        int                     route_position;
  BgpT_Mp_Prefix*               jth_withdrwn_rte_ptr;
  BgpT_Rte_Entry*               found_rte_entry_ptr;
  BgpT_Rte_Entry*               replacement_rte_ptr;
        Boolean                 delete_route = OPC_FALSE;
        Boolean                 add_route = OPC_FALSE;
        Boolean                 update_route = OPC_FALSE;
        MplsT_NHLFE*            top_nhlfe_ptr;
        MplsT_FEC               fec_name;
        Boolean                 jth_withdrwn_cutoff;

        /** This function removes all unfeasible routes from the Loc-RIB **/
        FIN (bgp_unfeasible_routes_process (withdrawn_route_list_ptr, withdrawing_peer_id))

        /* Get the number of withdrawn (unfeasible) routes.             */
  num_withdrwn_rte = op_prg_list_size (withdrawn_route_list_ptr);

  /* Loop through the Local Route Information Base and remove the      */
        /* unfeasible routes.
                        */
  for (count_j = 0; count_j < num_withdrwn_rte; count_j++)
    {
    /* Get the jth withdrawn route.                                    */
                /* This route will be removed from the list and added on only     */
                /* if the route existed in the local table. This ensures that     */
                /* unneccesary routes do not keep moving around.                  */
    jth_withdrwn_rte_ptr = (BgpT_Mp_Prefix*) op_prg_list_access (withdrawn_route_list_ptr,
count_j);
                jth_withdrwn_cutoff = jth_withdrwn_rte_ptr->cutoff;
                /* We have to check if node 8 has a withdrawn route with cutoff reached, then we
will do the same procedure just without the replacement route being advertised*/
    /* Search the routes in the routing info base to find a route  */
                /* that matches the withdrawn route.
        */
        found_rte_entry_ptr = bgp_support_rte_entry_find (bgp_adj_local_rib_ptr,
jth_withdrwn_rte_ptr, &route_position);
                if (found_rte_entry_ptr != OPC_NIL && found_rte_entry_ptr-
>local_adv_receive_peer_id == withdrawing_peer_id)
       {
```

```
                        /* We have a match.                                  */
                        /* Next step is to find a replacement route from */
                        /* RIB-INs of other connections. The IP common rte       */
                        /* table is passed to this function to break ties    */
                        /* if the address family is IPv4. It will NOT be     */
                        /* used for VPNv4 prefixes.                          */
                        replacement_rte_ptr = bgp_replacement_route_find (found_rte_entry_ptr,
ip_rte_table_ptr);

        /* Re-initialize the route operation flags                    */
                        delete_route    = OPC_FALSE;
                        add_route               = OPC_FALSE;
                        update_route    = OPC_FALSE;

                        if (replacement_rte_ptr == OPC_NIL)
                                {
                                /* No replacement was found. Route must be deleted.   */
                                delete_route = OPC_TRUE;
                                }
                        else if (replacement_rte_ptr->admin == found_rte_entry_ptr->admin)
                                {
                                /* Since the replace has the same admin weight as the   */
                                /* original, an update can be performed.                */
                                update_route = OPC_TRUE;
                                }
                        else
                                {
                                /* Original and replacement have different admin weights.       */
                                /* We need a delete followed by an add.                         */
                                delete_route = add_route = OPC_TRUE;
                                }

                        /* If this is a VPNv4 route that is being updated, then we must   */
                        /* find an NHLFE to the next BGP hop, since the packets will      */
                        /* have to be label-switched in the core. If such an LSP is not      */
                        /* present, then we should not be adding the route.                */
                        if (add_route || update_route)
                                {
                                /* New route is being added.     */
                                if (replacement_rte_ptr->dest_prefix_ptr->address_family_id ==
BgpC_Vpnv4_Address)
                                        {
                                        /* The destination is a VPN site.                                */
                                        /* We must have an LSP to the next hop of this route.    */
                                        if (OPC_FALSE == bgp_support_lsp_to_remote_pe_available
(replacement_rte_ptr->path_attrs_ptr->next_hop, &top_nhlfe_ptr, &fec_name,
ip_module_data_ptr))
                                                {
```

```
                                                /* No LSP to the destination PE. This route is no good.
        */
                                                add_route = update_route = OPC_FALSE;
                                                }
                                        }
                                }

                        /* Update all the IP routing tables that are interested in this       */
                        /* information. Could be just the common routing table or some*/
                        /* VRF tables (based on import target configuration).                 */
        bgp_update_ip_tables (found_rte_entry_ptr, replacement_rte_ptr, ip_module_data_ptr,
delete_route, update_route, add_route, ibgp_unique_proc_id, bgp_unique_proc_id,
                ibgp_admin_distance, top_nhlfe_ptr, fec_name, bgp_adj_local_rib_ptr);

                        /* Update the BGP Local RIB with this information.        */
                        if (delete_route)
                        bgp_support_ith_rte_entry_delete (bgp_adj_local_rib_ptr, route_position);

                        if (update_route)
                        bgp_support_ith_rte_entry_replace (bgp_adj_local_rib_ptr, route_position,
replacement_rte_ptr);

                        if (add_route)
                        bgp_support_rte_entry_ith_place_insert (bgp_adj_local_rib_ptr,
replacement_rte_ptr, route_position);
                        /* Add the replacement route to a list so that it is propagated to */
                        /* other BGP peers.                */

                        /* only add the replacement route if it is node 8 and the cutoff is not set*/
                        if (replacement_rte_ptr != OPC_NIL){
                        if (node_id==8){
                                if (jth_withdrwn_cutoff==OPC_FALSE)
                                        op_prg_list_insert (new_rte_list_ptr, replacement_rte_ptr,
OPC_LISTPOS_TAIL);
                                else
                                        op_prg_list_remove (withdrawn_route_list_ptr, count_j);
                                }

                        else
                        op_prg_list_insert (new_rte_list_ptr, replacement_rte_ptr,
OPC_LISTPOS_TAIL);
                        }
                        }
                } /* End for all withdrawn routes.          */

        FOUT;
        }
```

```
static int
bgp_reachability_info_process (BgpT_Conn_Info* bgp_conn_info_ptr)
        {
        int                                     count_i;
        int                                     num_new_routes;
        BgpT_Rte_Entry                         *ith_rte_entry_ptr;
        Boolean                                 remote_vpn_site;

        /** The new routes will be processed and the entries that turn out      **/
        /** to be the best routes to the destination that they advertise   **/
        /** will entered into the Local Routing Information Base and the        **/
        /** others will be ignored, but not destroyed.                          **/
        FIN (bgp_reachability_info_process (<args>))
        /* Access the route entries one and by one and pass them to the        */
        /* route process select function.
                */
        num_new_routes = op_prg_list_size (new_rte_list_ptr);

        /* Process the list of reachablity information.                        */
        for (count_i = 0; count_i < num_new_routes; count_i++)
                {
                /* Remove the route from the list....                          */
                ith_rte_entry_ptr = (BgpT_Rte_Entry *)op_prg_list_remove (new_rte_list_ptr,
OPC_LISTPOS_HEAD);

                remote_vpn_site = (ip_node_is_pe (ip_module_data_ptr) && (bgp_conn_info_ptr-
>neighbor_site_vrf_name == OPC_NIL));

                bgp_reachable_rte_process (ith_rte_entry_ptr, remote_vpn_site);

                /* Continue till all the routes are procesed.                                  */
                }

        FRET (op_prg_list_size (new_rte_list_ptr));
        }
```

```
static void
bgp_reachable_rte_process (BgpT_Rte_Entry* rte_entry_ptr, Boolean remote_vpn_site)
        {
        Boolean                         new_route_selected;
        Boolean                         update_route, add_route, delete_route;
        int                             old_entry_position;
```

```
BgpT_Rte_Entry*                 old_rte_entry_ptr = OPC_NIL;
MplsT_NHLFE*                    top_nhlfe_ptr = OPC_NIL;
MplsT_FEC                       fec_name = OPC_NIL;
BgpT_Rte_Entry*                 CAGG_Route = OPC_NIL;
Int num_routes=0;
BgpT_Rte_Entry*                  ith _rte_ptr=OPC_NIL;
Boolean   Found == OPC_FALSE;
/** This function adds an entry to the IP routing table or modifies an existing one if present.
**/
FIN (bgp_reachable_rte_process ());

new_route_selected = bgp_support_route_select (ip_rte_table_ptr,
        bgp_adj_local_rib_ptr, rte_entry_ptr, remote_vpn_site, &old_rte_entry_ptr,
         &old_entry_position, &top_nhlfe_ptr, &fec_name);

/* If the new route was selected insert it into the Loc-RIB          */
/* and also update the Forwarding Information Base.                  */
/* else do nothing.
                */
if (new_route_selected == OPC_FALSE)
        FOUT;
/* Insert this route into the new route list, so that it will */
/* be propagated to other peers.*/


/* if it is the CAGG node, then we have to create a new CAGG route */

if (node_id == 8 ) {

/* Insert this route into the new route list, so that it will */
/* be propagated to other peers.
*/
If (rte_entry_ptr->penalized == OPC_FALSE)
        op_prg_list_insert (new_rte_list_ptr, rte_entry_ptr, OPC_LISTPOS_TAIL);

Else
{
CAGG_Route = bgp_support_rte_entry_copy (rte_entry_ptr);
CAGG_Route-> path_attrs_ptr= bgp_CAGG_Create (rte_entry_ptr-> path_attrs_ptr);
new_route_selected = bgp_support_route_select_modified (ip_rte_table_ptr,
        bgp_adj_local_rib_ptr, rte_entry_ptr, remote_vpn_site, &old_rte_entry_ptr,
```

```
                    &old_entry_position, &top_nhlfe_ptr, &fec_name);

num _routes =op_prg_list_size (last_advertised_iminus1  );
If (last_advertised_iminus1  ==  OPC_NIL || num_routes==0 ){
op_prg_list_insert (new_rte_list_ptr ,CAGG_Route, OPC_LISTPOS_TAIL);
op_prg_list_insert (last_advertised_i, bgp_support_mp_address_copy (CAGG_Route->
dest_prefix_ptr), OPC_LISTPOS_TAIL);
}
Else{

/* compare if the entry already advertised last time
   for (i_count = 0; i_count < num _routes; i_count ++){
ith _rte_ptr = (BgpT_Mp_Prefix *) op_prg_list_access (last_advertised_iminus1  , i_count);

      if(bgp_support_mp_prefix_equal(ith _rte_ptr -> dest_prefix_ptr , rte_entry_ptr->
dest_prefix_ptr)){
op_prg_list_insert (last_advertised_i, bgp_support_mp_address_copy (CAGG_Route->
dest_prefix_ptr), OPC_LISTPOS_TAIL);
Found==OPC_TRUE;
break;
}

If (!Found)
{
op_prg_list_insert (new_rte_list_ptr ,CAGG_Route, OPC_LISTPOS_TAIL);
op_prg_list_insert (last_advertised_i, bgp_support_mp_address_copy (CAGG_Route->
dest_prefix_ptr), OPC_LISTPOS_TAIL);
}
}
/* The new route must make its way to the IP route table(s).      */

        /* If there is no old route for this destination, then the only       */
        /* operation that we need to perform is an ADD.                       */
        if (old_rte_entry_ptr == OPC_NIL)
                {
                update_route = delete_route = OPC_FALSE;
                add_route = OPC_TRUE;
                }

        /* If the old route has a different admin distance or a different   */
        /* next hop, we cannot perform an update operation on the existing      */
```

```
        /* IP route. We must do a DELETE followed by an ADD.                    */
        else if ((old_rte_entry_ptr->admin != rte_entry_ptr->admin) ||
                        !inet_address_equal (old_rte_entry_ptr->path_attrs_ptr->next_hop,
                                rte_entry_ptr->path_attrs_ptr->next_hop))
                {
                add_route = delete_route = OPC_TRUE;
                update_route = OPC_FALSE;
                }
        /* If the admin distance and the next hop is the same, then it is  */
        /* only a metric change. UPDATE function can be used.              */
        else
                {
                add_route = delete_route = OPC_FALSE;
                update_route = OPC_TRUE;
                }
        bgp_update_ip_tables (old_rte_entry_ptr, rte_entry_ptr, ip_module_data_ptr,
                delete_route, update_route, add_route, ibgp_unique_proc_id, bgp_unique_proc_id,
                ibgp_admin_distance, top_nhlfe_ptr, fec_name, bgp_adj_local_rib_ptr);

        /* The IP table has been updated. The BGP table can be updated now.    */
        /* If an old route exists, replace it.          */
        if (old_rte_entry_ptr != OPC_NIL)
                {
                bgp_support_ith_rte_entry_replace (bgp_adj_local_rib_ptr, old_entry_position,
rte_entry_ptr);
                }
        /* If there is no old route for this destination, just add the new route.      */
        else
                {
                bgp_support_rte_entry_ith_place_insert (bgp_adj_local_rib_ptr, rte_entry_ptr,
old_entry_position);
                }

        }
        else
                {
        op_prg_list_insert (new_rte_list_ptr, rte_entry_ptr, OPC_LISTPOS_TAIL);

        /* The new route must make its way to the IP route table(s).       */

        /* If there is no old route for this destination, then the only        */
```

```
        /* operation that we need to perform is an ADD.                                    */

        if (old_rte_entry_ptr == OPC_NIL)
                {
                update_route = delete_route = OPC_FALSE;
                add_route = OPC_TRUE;
                }
        /* If the old route has a different admin distance or a different   */
        /* next hop, we cannot perform an update operation on the existing      */
        /* IP route. We must do a DELETE followed by an ADD.                       */
        else if ((old_rte_entry_ptr->admin != rte_entry_ptr->admin) ||
                        !inet_address_equal (old_rte_entry_ptr->path_attrs_ptr->next_hop,
                                rte_entry_ptr->path_attrs_ptr->next_hop))
                {
                add_route = delete_route = OPC_TRUE;
                update_route = OPC_FALSE;
                }

        /* If the admin distance and the next hop is the same, then it is  */
        /* only a metric change. UPDATE function can be used.                      */
        else
                {
                add_route = delete_route = OPC_FALSE;
                update_route = OPC_TRUE;
                }
        bgp_update_ip_tables (old_rte_entry_ptr, rte_entry_ptr, ip_module_data_ptr,
                delete_route, update_route, add_route, ibgp_unique_proc_id, bgp_unique_proc_id,
                ibgp_admin_distance, top_nhlfe_ptr, fec_name, bgp_adj_local_rib_ptr);

        /* The IP table has been updated. The BGP table can be updated now.    */

        /* If an old route exists, replace it.          */
        if (old_rte_entry_ptr != OPC_NIL)
                {
                bgp_support_ith_rte_entry_replace (bgp_adj_local_rib_ptr, old_entry_position,
rte_entry_ptr);
                }
        /* If there is no old route for this destination, just add the new route.     */
        else{
                bgp_support_rte_entry_ith_place_insert (bgp_adj_local_rib_ptr, rte_entry_ptr,
old_entry_position);
```

```
            }
        }
        FOUT;
        }
```

```
Boolean
bgp_support_route_select_modified (IpT_Cmn_Rte_Table* ip_cmn_rte_table_ptr,
        BgpT_Rte_Table* local_rte_tbl_ptr, BgpT_Rte_Entry* new_entry_ptr,
        Boolean remote_vpn_site, BgpT_Rte_Entry **old_entry_pptr,
        int* old_entry_pos_ptr, MplsT_NHLFE** top_nhlfe_pptr, MplsT_FEC* fec_ptr)
        {
        BgpT_Rte_Entry*                         found_rte_entry_ptr;
        int                                     best_rte_index = 0;
        Boolean                                 new_route_selected = OPC_FALSE;
        IpT_Cmn_Rte_Table_Entry*    ip_rte_entry_ptr;
        double                                  first_route_update;
        double                                  second_route_update;
        /** The decision process will compare the new route     **/
        /** with all the existing routes in its local RIB.      **/
        /** These routes in this Routing Information Base are   **/
        /** the best routes known this far to the respective    **/
        /** destinations. If the new route will replace the     **/
        /** the installed route if it found to be better.       **/
        FIN (bgp_support_route_select ());

        /* First make sure that the next hop is a valid address     */
        /* by checking it against the default IP address. This      */
        /* will be the case for routes injected by IGPs.            */
        /* If Next hop is same as the advertising neighbor's        */
        /* IP address, the next hop is reachable, otherwise call*/
        /* the common rte table lookup to see if the IGPs know  */
        /* of any route the address marked as Next Hop.                     */

        /* This next hop check must not be made for VPNv4 routes        */
        /* because the next hop may be present in the common route      */
        /* table in some cases (remote CE), and in the VRF table in     */
        /* other cases (local CE).
        */
        /* Do not make check for 6PE next hops also.                        */
        if ((new_entry_ptr->dest_prefix_ptr->address_family_id != BgpC_Vpnv4_Address) &&
                (!inet_address_equal                    (new_entry_ptr->path_attrs_ptr->next_hop,
InetI_Default_v4_Addr)) &&
                (!inet_address_is_ipv4_mapped (&(new_entry_ptr->path_attrs_ptr->next_hop))))
                {
                if (inet_cmn_rte_table_lookup (ip_cmn_rte_table_ptr,
                                new_entry_ptr->path_attrs_ptr->next_hop,
                                 OPC_NIL,       OPC_NIL,       OPC_NIL,       &ip_rte_entry_ptr)       ==
OPC_COMPCODE_FAILURE)
                        {
                        /* There is no known route to the next hop.             */
                        /* Reject the route.                                                    */
                        FRET (OPC_FALSE);
                        }
```

```
                }

        /* Get the LSP to remote PE even for 6PE (if next hop is a mapped address).       */
        if (inet_address_is_ipv4_mapped (&(new_entry_ptr->path_attrs_ptr->next_hop)))
                {
                if                                              (bgp_support_lsp_to_remote_pe_available
(inet_ipv4_addr_from_ipv4_mapped_addr_get (&(new_entry_ptr->path_attrs_ptr->next_hop)),
                        top_nhlfe_pptr, fec_ptr, ip_cmn_rte_table_ptr->iprmd_ptr) == OPC_FALSE)
                        {
                        /* No label switched path. Route cannot be used.          */
                        FRET (OPC_FALSE);
                        }
                }
        /* For VPN routes coming from other PEs, there must be           */
        /* an LSP to the remote PE.                                              */
        else if (remote_vpn_site)
                {
                if      (bgp_support_lsp_to_remote_pe_available      (new_entry_ptr->path_attrs_ptr-
>next_hop, top_nhlfe_pptr, fec_ptr,
                        ip_cmn_rte_table_ptr->iprmd_ptr) == OPC_FALSE)
                        {
                        /* No label switched path (TE or LDP) available to the remote PE.          */
                        /* This route cannot be used.
                                */

                        /*  VRF  lite  support:  MPLS  LSPs  will  not  be  available  in  VRF  lite
implementations. */
                        /* FRET (OPC_FALSE); */
                        }
                }
        /* Search the route list and look for a route to    */
        /* the destination advertised by the new route.            */
        found_rte_entry_ptr  =  bgp_support_rte_entry_find  (local_rte_tbl_ptr,  new_entry_ptr-
>dest_prefix_ptr, old_entry_pos_ptr);
        if (found_rte_entry_ptr != OPC_NIL)
                {
                /* There is a previous route for this network.      */
                /* The next step is to select the better of the */
                /* two routes. The function will return the         */
                /* index of the best route. A return value of 1     */
                /* indicates that the route passed in as the        */
                /* first argument is the best route or a value      */
                /* of 2 indicates otherwise.                                     */
                /* If the current advertisement was received by */
                /* the same peer as the old route, replace the    */
                /* the old route with the new one without even */
                /* running the selection procedure.                              */
                if      (found_rte_entry_ptr->local_adv_receive_peer_id      ==      new_entry_ptr-
```

```
>local_adv_receive_peer_id)
                        {
                        /* Set the best route index to be 2.                */
                        best_rte_index = 2;
                        }
                else
                        {
                        /* Run the routes through the decision              */
                        /* process. */
        first_route_update=
search_my_update_list(found_rte_entry_ptr,my_updates_list,my_updates_time);
        second_route_update=
search_my_update_list(new_entry_ptr,my_updates_list,my_updates_time);
                                                                        */
                        best_rte_index = bgp_support_best_rte_find (found_rte_entry_ptr,
                                new_entry_ptr,local_rte_tbl_ptr->owner_as_number,
ip_cmn_rte_table_ptr,local_rte_tbl_ptr->routeproc_id, local_rte_tbl_ptr->ibgp_routeproc_id);

if (first_route_update > DELAY && best_rte_index == 1)
best_rte_index = 1;
else
if(second_route_update > DELAY && best_rte_index == 2)
best_rte_index = 2;
else
if (first_route_update> second_route_update)
best_rte_index = 1;
else
if (second_route_update > first_route_update>)
best_rte_index = 2;
                }

        if (best_rte_index == 2)
                        {
                        *old_entry_pptr    = found_rte_entry_ptr;

                        /* Set the value to the be returned.        */
                        new_route_selected = OPC_TRUE;
                        }
                }
        else
                {
                /* If we didn't find an already installed entry,*/
                /* then this route may be the first only known    */
                /* route therefore install it.                    */
                new_route_selected = OPC_TRUE;
                }
        FRET (new_route_selected);
        }
```

```
BgpT_Path_Attrs*
bgp _CAGG_Create (BgpT_Path_Attrs* orig_path_attrs_ptr)
        {
        BgpT_Path_Attrs*        copy_path_attrs_ptr;
        BgpT_Path_Segment*    new_path_seg_ptr1;
        BgpT_Path_Segment*   new_path_seg_ptr2;
        BgpT_Path_Segment*   new_path_seg_ptr3;

        List*                                   new_as_path_list_ptr;
        int                                             num_path_segments;
        int                                             ith_entry;
        int                                             seg_length;
        int                                             seg_length2;
        int                                             count_i;

        /** This function creates a copy of the    path attributes data      **/
        /** structure that is passed in as the argument and returns       **/
        /** a pointer to the copy.
                **/
        FIN (bgp_support_path_attrs_copy (orig_path_attrs_ptr))

        /* Allocate a new block of memory for the copy of the path                 */
        /* attributes.
                        */
        copy_path_attrs_ptr = bgp_support_path_attrs_mem_alloc ();

        /* Transfer the contents from original to the copy area.             */
        copy_path_attrs_ptr->num_references = 0;
        copy_path_attrs_ptr->origin                = orig_path_attrs_ptr->origin;
        copy_path_attrs_ptr->next_hop                   = inet_address_copy (orig_path_attrs_ptr-
>next_hop);
        copy_path_attrs_ptr->multi_exit_disc= orig_path_attrs_ptr->multi_exit_disc;
        copy_path_attrs_ptr->local_pref                 = orig_path_attrs_ptr->local_pref;
        copy_path_attrs_ptr->as_path_length    = 3;

        /* The member that needs special care is the as_path list.                 */
        new_as_path_list_ptr = op_prg_list_create ();

        /* This list would contain a set of path segments.  Need to          */
        /* scan through each segment and create a copy a copy                      */
        num_path_segments = 3;

        /* Create a new Path segment memory  block.              */
        new_path_seg_ptr 1= bgp_support_path_seg_mem_alloc ();
        new_path_seg_ptr 2= bgp_support_path_seg_mem_alloc ();
        new_path_seg_ptr 3= bgp_support_path_seg_mem_alloc ();
```

```
        /* Copy the members that are of standard data type.                    */
        new_path_seg_ptr1->segment_type  = BgpC_Path_Seg_Type_As_Sequence;
new_path_seg_ptr2->segment_type  = BgpC_Path_Seg_Type_As_Set;
new_path_seg_ptr3->segment_type  = BgpC_Path_Seg_Type_As_Sequence;

seg_length = new_path_seg_ptr1->segment_length = 1;
new_path_seg_ptr3->segment_length = 1;
seg_length2 = new_path_seg_ptr2->segment_length = 6;

        /* The memeber segment value is an array of AS numbers. Copy */
        /* that into a new array.
        */

new_path_seg_ptr1->segment_value_array = (int*) prg_cmo_alloc (bgp_as_path_list_cmh,
(seg_length)*sizeof (int));
new_path_seg_ptr2->segment_value_array = (int*) prg_cmo_alloc (bgp_as_path_list_cmh,
(seg_length2)*sizeof (int));
new_path_seg_ptr3->segment_value_array = (int*) prg_cmo_alloc (bgp_as_path_list_cmh,
(seg_length)*sizeof (int));

                /* Add the first AS number as the first entry in the first as path    */
new_path_seg_ptr1->segment_value_array [0] = 1;

new_path_seg_ptr2->segment_value_array [0] = 2;
new_path_seg_ptr2->segment_value_array [1] = 3;
new_path_seg_ptr2->segment_value_array [2] = 4;
new_path_seg_ptr2->segment_value_array [3] = 5;
new_path_seg_ptr2->segment_value_array [4] = 6;
new_path_seg_ptr2->segment_value_array [5] = 8;

new_path_seg_ptr3->segment_value_array [0] = 7;

for(int k=0;k==(3  - Best_AS_Path_L);k++)
new_path_seg_ptr4->segment_value_array [0] = 7;




                /* Insert this path segment into the as_path list pointer  */
op_prg_list_insert (new_as_path_list_ptr, new_path_seg_ptr1, OPC_LISTPOS_TAIL);
op_prg_list_insert (new_as_path_list_ptr, new_path_seg_ptr2, OPC_LISTPOS_TAIL);
op_prg_list_insert (new_as_path_list_ptr, new_path_seg_ptr3, OPC_LISTPOS_TAIL);
op_prg_list_insert (new_as_path_list_ptr, new_path_seg_ptr4, OPC_LISTPOS_TAIL);
```

```
                /* Continue processing the next path segment in the list */
                }

        /* All the as_path segments have been added and the as list has */
        /* been copied succefully. Assign the new list pointer to the       */
        /* copy of the path attributes.
        */
        copy_path_attrs_ptr->as_path_list_ptr = new_as_path_list_ptr;

        /* Now copy the originator_ID and the cluster list.                                    */
        copy_path_attrs_ptr->originator_id                  = orig_path_attrs_ptr->originator_id;
        if (OPC_NIL != orig_path_attrs_ptr->cluster_list_ptr)
                {
                int       num_elems, elem_index;
                copy_path_attrs_ptr->cluster_list_ptr    = op_prg_list_create ();
                /* Perform a shallow copy of the cluster IDs. These are state variables    */
                /* and must not be destroyed.
                                */
                num_elems = op_prg_list_size (orig_path_attrs_ptr->cluster_list_ptr);
                for (elem_index = 0; elem_index < num_elems; elem_index++)
                        {
                        op_prg_list_insert (copy_path_attrs_ptr->cluster_list_ptr,
                                op_prg_list_access (orig_path_attrs_ptr->cluster_list_ptr,
elem_index),
                                OPC_LISTPOS_TAIL);
                        }
                }

        /* Now copy the community set.
                */
        /* Remember that we need to allocate a new block of memory for        */
        /* community set and copy the communities one by one.                                  */
        if (OPC_NIL != orig_path_attrs_ptr->community_set_ptr)
                {
                /* Allocate enough memory for the community set.                               */
                copy_path_attrs_ptr->community_set_ptr        =
bgp_support_community_set_mem_alloc ();
                /* copy the number of communities.
        */
                copy_path_attrs_ptr->community_set_ptr->num_communities =
                        orig_path_attrs_ptr->community_set_ptr->num_communities;
                /* If the community set is non empty, allocate enough memory*/
                /* to store the communities and copy them one by one.          */
                if (0 != orig_path_attrs_ptr->community_set_ptr->num_communities)
                        {
                        copy_path_attrs_ptr->community_set_ptr->communities_array =
(BgpT_Community_Number*)
```

```
                                    prg_cmo_alloc (bgp_comm_number_cmh, copy_path_attrs_ptr-
>community_set_ptr->num_communities *
                                                                        sizeof (BgpT_Community_Number));

                            for (count_i=0; count_i<copy_path_attrs_ptr->community_set_ptr-
>num_communities; count_i++)
                                    {
                                    copy_path_attrs_ptr->community_set_ptr-
>communities_array[count_i] =
                                            orig_path_attrs_ptr->community_set_ptr-
>communities_array[count_i];
                                    }
                            }
                    else
                            {
                            /* The community set is empty, set the pointer to nil.     */
                            orig_path_attrs_ptr->community_set_ptr->communities_array = OPC_NIL;
                            }
                    }
            /* Done with the copying. Return the newly created value.                  */
            FRET (copy_path_attrs_ptr);
            }
```