

**THE IMPLEMENTATION OF ERROR CHECKING
AND OPTIONAL PARAMETER NEGOTIATION
FOR BGP-4 IN A NETWORK SIMULATOR**

by

Naomi F. Ko

B.A.Sc. (Honours), Simon Fraser University, 2001

A PROJECT SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ENGINEERING

In the School
of
Engineering Science

© Naomi F. Ko 2004

SIMON FRASER UNIVERSITY

Fall 2004

All rights reserved.

This work may not be reproduced in whole or in part, by photocopy
or other means, without permission of the author.

Approval

Name: Naomi F. Ko
Degree: Master of Engineering
Title of Project: The Implementation of Error Checking and Optional
Parameter Negotiation for BGP-4 in a Network Simulator

Examining Committee:

Chair: Dr. Glenn Chapman
Professor of the School of Engineering Science

Dr. Ljiljana Trajković
Senior Supervisor
Professor of the School of Engineering Science

Dr. Shahram Payandeh
Supervisor
Professor of the School of Engineering Science

Date Approved: _____

Abstract

Border Gateway Protocol (BGP) is the *de facto* inter-domain routing protocol currently used by the Internet. Its robustness and scalability have propelled BGP-4, the current version of BGP, to worldwide use. Therefore scalability and performance have become of particular interest to the communication networks community. In order to aid the research and development of communication networks, several simulation tools and network models have been developed. Among them are SSFNet and ns-2, each containing a BGP model to replicate the behaviour of the BGP network protocol.

This project expands on and adds more capabilities to the existing BGP model that was created in the ns-2 network simulator, to improve the model's accuracy according to the BGP specification. Among the changes are BGP header and message error checking, and the implementation and negotiation of the Authentication Information and Capabilities Advertisement Optional Parameters in the OPEN message. Simulation results show that the new error detection and optional parameter negotiation capabilities more closely echo the required behaviour in the BGP specification.

These modifications to ns-BGP will ensure that the model grows as the protocol does, with the ability to scale and handle new parameters and capabilities.

Dedication

*To my family,
for their love and support.*

*To Steve,
for always being there.*

Acknowledgements

As with any large project, the involvement of others plays a major role in its completion. Thus, this project would be incomplete without credit to those whose contributions have been much appreciated. I would like to thank:

Dr. Ljiljana Trajković, of the SFU School of Engineering Science – for her guidance and patience on this project;

Dr. Shahram Payandeh, of the SFU School of Engineering Science – for his feedback and comments on my project;

Tony Dongliang Feng – for implementing an excellent BGP-4 model in ns-2, serving as the foundation on which to build my project, and for being so very generous toward me with his time (even long after his graduation);

Nenad Lasković – for his endless patience in helping me learn C++ and all the nuances of object-oriented programming and ns-2;

Mary Wong and Jenny Koo – for spending several lunch hours imparting their insight and experience with me, and for providing many answers and clarifications;

The members of the Communication Networks Laboratory at SFU – for their help and for their company in the lab;

And several nameless others who supported me and encouraged me along the way, even when the state of the project was dispiriting.

I thank you.

Table of Contents

Approval	ii
Abstract	iii
Dedication	iv
Acknowledgements	v
Table of Contents	vi
Lists of Figures and Tables	viii
List of Acronyms and Technical Terms	x
Chapter 1 : Introduction	1
Chapter 2 : Background Knowledge	2
2.1 Border Gateway Protocol.....	2
2.2 SSFNet and ns-BGP Implementations of BGP.....	4
Chapter 3 : Proposed Project	5
3.1 BGP Header and Message Error Checking.....	5
3.1.1 BGP Header.....	5
3.1.2 OPEN Message.....	7
3.1.3 UPDATE Message.....	8
3.1.4 NOTIFICATION Message.....	10
3.1.5 KEEPALIVE Message.....	10
3.2 OPEN Process and Negotiation of Optional Parameters.....	10
3.2.1 OPEN Process.....	10
3.2.2 Optional Parameters.....	14
3.3 NOTIFICATION Error Codes and Subcodes.....	17
Chapter 4 : Project Implementation	19
4.1 BGP Header and Message Error Checking.....	19
4.2 NOTIFICATION Error Codes and Subcodes.....	21
4.3 Negotiation of Optional Parameters.....	21
4.3.1 Authentication Mechanism.....	21
4.3.2 Capabilities Advertisement.....	23

Chapter 5 : Simulation Results	27
5.1 Ideal Session Establishment.....	27
5.2 BGP Header and Message Error Checking.....	30
5.2.1 Unacceptable Hold Time Value	30
5.2.2 Unexpected Marker Field.....	30
5.2.3 Bad Path Attribute Flag.....	31
5.3 OPEN Process Negotiation of Optional Parameters	31
5.3.1 Scenario 1: Matching Authentication Mechanisms.....	31
5.3.2 Scenario 2: One-Sided Authentication Mechanism	34
5.3.3 Scenario 3: Different Authentication Mechanisms	37
5.3.4 Scenario 4: Mutually Exclusive Capability Sets	40
5.3.5 Scenario 5: Overlapping Capability Sets	43
5.3.6 Scenario 6: Both Optional Parameters – One Negotiation Required	47
5.3.7 Scenario 7: Both Optional Parameters – Two Negotiations Required	50
Chapter 6 : Future Enhancements.....	56
6.1 Completion of BGP Header and Message Error Checking.....	56
6.2 Restructuring and Addition of OPEN Message Optional Parameters	56
6.3 Support for Different Authentication Mechanisms in the Same BGP Session	57
6.4 Addition of New Capabilities Advertisements	57
6.5 Addition of New Path Attributes	57
6.6 Handling of NULL Characters.....	57
Chapter 7 : Conclusion	58
Appendix A : List of Modified Files.....	59
Appendix B : Simulation Results in Full	61
B.1 Ideal Negotiation Process	61
B.2 BGP Header and Message Error Checking.....	63
B.2.1 Unacceptable Hold Time Value	63
B.2.2 Unexpected Marker Field.....	64
B.2.3 Bad Path Attribute Flag.....	65
B.3 OPEN Process Negotiation of Optional Parameters	66
B.3.1 Scenario 1: Matching Authentication Mechanisms.....	66
B.3.2 Scenario 2: One-Sided Authentication Mechanism	68
B.3.3 Scenario 3: Different Authentication Mechanisms	71
B.3.4 Scenario 4: Mutually Exclusive Capability Sets	74
B.3.5 Scenario 5: Overlapping Capability Sets	77
B.3.6 Scenario 6: Both Optional Parameters – One Negotiation Required	81
B.3.7 Scenario 7: Both Optional Parameters – Two Negotiations Required	85
List of References	90

Lists of Figures and Tables

Figure 1: BGP message header format.....	6
Figure 2: OPEN message format.....	7
Figure 3: UPDATE message format.....	8
Figure 4: Path Attribute format.....	8
Figure 5: NOTIFICATION message format.....	10
Figure 6: Ideal OPEN process.....	11
Figure 7: Failed OPEN process.....	12
Figure 8: Failed OPEN process with reconnection attempt: one peer objects.....	13
Figure 9: Failed OPEN process with reconnection attempt: both peers object.....	14
Figure 10: Optional Parameter format.....	14
Figure 11: Authentication Information optional parameter format.....	15
Figure 12: Capabilities Advertisement optional parameter format.....	16
Figure 13: Authentication negotiation algorithm when an OPEN message is received.....	22
Figure 14: Authentication negotiation algorithm when an OPEN message is sent.....	23
Figure 15: Capability negotiation algorithm when an OPEN message is received.....	24
Figure 16: Capability negotiation algorithm when an OPEN message is sent.....	25
Figure 17: OPEN messages without optional parameters.....	27
Figure 18: Receipt and processing of OPEN messages.....	28
Figure 19: BGP sessions are established.....	29
Figure 20: Marker field is checked.....	29
Figure 21: Unacceptable Hold Time value.....	30
Figure 22: Unexpected Marker field.....	30
Figure 23: Bad UPDATE Path Attribute flag.....	31
Figure 24: Scenario 1: OPEN messages.....	32
Figure 25: Scenario 1: OPEN messages received.....	33
Figure 26: Scenario 1: Different Marker field used.....	34
Figure 27: Scenario 2: OPEN messages (first attempt).....	34
Figure 28: Scenario 2: Conflicting authentication mechanisms.....	35
Figure 29: Scenario 2: OPEN messages with no authentication (second attempt).....	35
Figure 30: Scenario 2: BGP session establishment with no authentication.....	36
Figure 31: Scenario 3: OPEN messages (first attempt).....	37
Figure 32: Scenario 3: Authentication mechanisms rejected.....	38
Figure 33: Scenario 3: OPEN messages with no authentication (second attempt).....	39
Figure 34: Scenario 3: BGP session establishment with no authentication.....	39
Figure 35: Scenario 4: OPEN messages (first attempt).....	40
Figure 36: Scenario 4: Capabilities accepted by n1, rejected by n0.....	41
Figure 37: Scenario 4: OPEN messages with no capabilities (second attempt).....	42
Figure 38: Scenario 4: BGP session establishment with no capabilities.....	42
Figure 39: Scenario 5: OPEN messages (first attempt).....	43
Figure 40: Scenario 5: Capabilities rejected by both peers.....	44
Figure 41: Scenario 5: OPEN messages with revised capabilities (second attempt).....	45
Figure 42: Scenario 5: BGP session establishment with one capability.....	46
Figure 43: Scenario 6: OPEN messages (first attempt).....	47
Figure 44: Authentication accepted, capability rejected.....	48

Figure 45: Scenario 6: OPEN messages with revised parameters (second attempt).	49
Figure 46: Scenario 6: Different Marker field used.	49
Figure 47: Scenario 7: OPEN messages (first attempt).	50
Figure 48: Scenario 7: Authentication mechanisms rejected (first attempt).	51
Figure 49: Scenario 7: OPEN messages (second attempt).	52
Figure 50: Scenario 7: Capabilities rejected (second attempt).	53
Figure 51: Scenario 7: OPEN messages (third attempt).	54
Figure 52: Scenario 7: BGP session establishment with no authentication, one capability.	55

Table 1: List of Error Codes and Subcodes for NOTIFICATION messages.	18
Table 2: Implemented error checks for BGP messages.	20
Table 3: Negotiated authentication mechanism.	22
Table 4: Negotiated capabilities.	24
Table 5: Negotiated authentication mechanism and capabilities.	26

List of Acronyms and Technical Terms

AS	Autonomous System
BGP	Border Gateway Protocol
EGP	External Gateway Protocol
IETF	Internet Engineering Task Force
IGP	Internal Gateway Protocol
MED	Multiple Exit Discriminator
NLRI	Network Layer Reachability Information
OSPF	Open Shortest Path First
RFC	Request for Comments
RIP	Routing Information Protocol
SSFNet	Scalable Simulation Framework Network
SSFNet.OS.BGP4	SSFNet's BGP model

Chapter 1: Introduction

Border Gateway Protocol (BGP) is the *de facto* inter-domain routing protocol currently used by the Internet. Its robustness and scalability have propelled BGP-4, the current version of BGP, to worldwide use. Therefore scalability and performance have become of particular interest to the communication networks community.

In order to aid the research and development of communication networks, several simulation tools and network models have been developed, including SSFNet and ns-2. Each network simulator contains a BGP model to replicate the behaviour of the BGP protocol.

As with all research tools, constant development and improvement are required to ensure that the simulation tools imitate real-life scenarios as closely as possible. The purpose of this project is to expand on and add more capabilities to an existing BGP model that was created in the ns-2 network simulator.

Chapter 2 provides background information on the Border Gateway Protocol necessary to understand the implications of this project. A brief description of the SSFNet BGP and ns-BGP models is given. Chapter 3 outlines the proposed project. Chapter 4 details the implementation, and annotated simulation results are presented in Chapter 5.

Chapter 6 provides future enhancements that may be implemented for accuracy, better scalability and code efficiency. A conclusion is given in Chapter 7, with references provided at the end of this report.

Chapter 2: Background Knowledge

A router is a hardware device that forwards data packets in a network [19]. It uses network information propagated by other routers to build a forwarding table that contain the best route to a specific network. Located at a junction of two or more networks [19], the router uses the table in conjunction with packet headers to determine how to direct the data traffic to its final destination.

The Internet consists of many autonomous systems (ASes), each an independent network of routers used to exchange routing information within an organisation. To exchange information among the routers, the network employs intra-domain routing protocols or internal gateway protocols (ISPs), such as Open Shortest Path First (OSPF) or Routing Information Protocol (RIP). Communication among ASes is facilitated by an inter-domain routing protocol or external gateway protocol (EGP), such as the Border Gateway Protocol (BGP).

2.1 Border Gateway Protocol

The Border Gateway Protocol (BGP) is the inter-domain routing protocol currently used by the Internet. Its robustness and scalability have propelled BGP-4, the current version of BGP, to worldwide use, making it the single *de facto* protocol for routing between ASes.

[18]

BGP's operation is based on the exchange of 4 types of messages: OPEN, UPDATE, NOTIFICATION, and KEEPALIVE.

The OPEN message contains handshaking information required for the BGP speakers to identify themselves and to establish a peer session using an agreed set of parameters. Any discrepancies are negotiated by exchanging OPEN and NOTIFICATION messages to arrive at a common set of parameters. This negotiation is discussed in greater detail in Section 3.2.

UPDATE messages contain routing information that needs to be distributed throughout the network, to inform all BGP speakers of networks that are no longer reachable (Unfeasible Routes) and networks that have become available (Network Layer Reachability Information, NLRI). The Unfeasible Routes, if any are included, are removed from the BGP speaker's forwarding table. Any NLRIs that are advertised in the UPDATE message are added to the forwarding table, once they have been checked for the absence of routing loops. Also included is a set of Path Attributes that pertains to all routes advertised in the UPDATE message, specifying certain properties of the routes. A decision-making process is executed as required to ensure that the route inserted is the best path for that network destination.

Each time a BGP message is received, a variety of error checks are performed to verify its integrity and the validity of the information contained within. If any one test fails, the BGP speaker alerts its peer by sending a NOTIFICATION message. The NOTIFICATION message contains error codes and error data that indicate the type of error that has occurred.

The fourth and last message type is the KEEPALIVE. Both peers send this short message periodically (at an agreed interval) to inform each other that they are still functioning and that the BGP session should remain active. This message is also used as an acknowledgement during the open process, to indicate that the proposed parameters are acceptable.

The portions of the BGP network communication protocol relevant to this project are: BGP header and message error checking, the OPEN negotiation process, and NOTIFICATION messages. These components are discussed in greater detail in the next chapter.

2.2 SSFNet and ns-BGP Implementations of BGP

SSFNet is a project that focusses on the research and development of scalable modelling and simulation tools. Using these tools, the dynamic behaviour of very large networks can be researched, particularly modelling scalability and performance scalability. The project is a Java-based network simulator with a BGP model SSF.OS.BGP4 written by Brian J. Premore [14].

The ns-2 network simulator is part of the Virtual InterNetwork Testbed (VINT) project, aimed at studying network protocol interactions and scalability. The DARPA-funded research project involves USC/ISI, Xerox PARC, LBNL and UC Berkeley. [12] The simulator, which is free and open source, is written in C++, with an OTcl command and configuration interface.

The ns-2 BGP model ns-BGP [9] was built by Tony Dongliang Feng by porting over SSFNet's BGP model, and thus accomplishes the same end. Some of the supporting models like TcpSocket, were also ported to ns-2 to maintain a hierarchy parallel to that of the SSFNet BGP model.

Chapter 3: Proposed Project

In order to simplify the BGP model, several assumptions were made during the implementation of SSF.OS.BGP4. These simplifications were thus also carried over to ns-BGP. This project removes some of those simplifications and assumptions, and expands on the capabilities of the ns-BGP model.

3.1 BGP Header and Message Error Checking

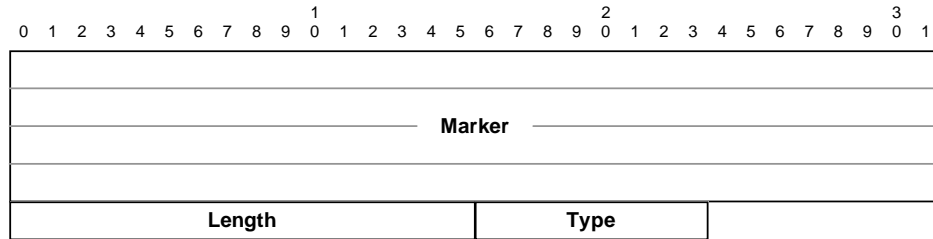
A major assumption made during the implementation of the BGP model was that no errors would occur, because the simulator is ideal. Thus, neither the SSFNet model nor the existing ns-BGP model provided any error checking, per Section 6 of [15], and error checking was completely omitted. [14] A large part of this project involves incorporating error checking into the BGP model, and ensuring that the message header and body fields comply with the recommended values in the BGP specification.

Upon receipt of a BGP message, the router checks the message header and body for errors. Any error in message format, in field value, in syntax, or in semantics in these fields results in a NOTIFICATION message being sent. In most cases, the BGP peer session and underlying TCP connection are terminated.

3.1.1 BGP Header

All BGP messages share a common 19-octet message header format. The format of the BGP message header is shown in Figure 1.

Figure 1: BGP message header format.



The 16-byte Marker field, used to verify the identity of a BGP speaker or to detect loss of synchronisation between peers, must hold an expected value. In the basic case, this field will contain all 1 bits (16 bytes of value 0xFF). When an authentication mechanism is used, the Marker may contain another value that must be in accordance with the specifications of the chosen authentication mechanism.

The second field indicates the length of the entire BGP message. The length of the message must lie in the acceptable range of 19 to 4096 bytes.

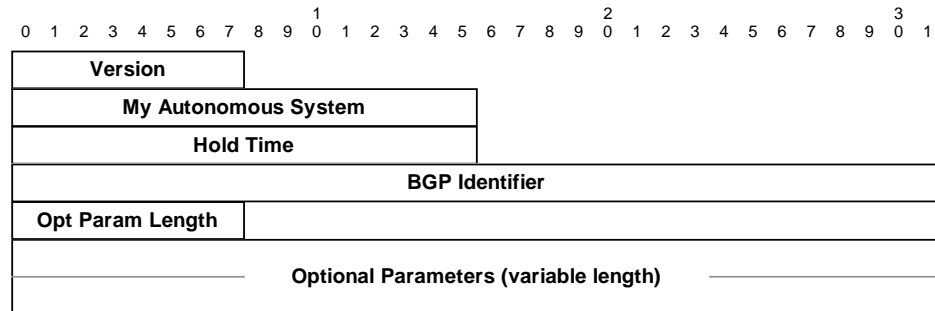
The last field of the header contains the Message Type, which designates the BGP message and thus the handling process. The valid message types are:

- Type 1: an OPEN message,
- Type 2: an UPDATE message,
- Type 3: a NOTIFICATION message, and,
- Type 4: a KEEPALIVE message.

3.1.2 OPEN Message

As soon as the underlying TCP connection is established, the BGP speaker initiates a peer session by sending an OPEN message. The OPEN message, shown in Figure 2, contains self-identification information and its proposed session parameters, adding a minimum of 10 octets to the BGP message length.

Figure 2: OPEN message format.



If the version of BGP sent by the peer is not supported by the speaker, a NOTIFICATION message is sent with the largest supported version number that is below the version received.

The Autonomous System in the OPEN message must be an acceptable AS number; however the determination of a valid AS number is not covered in the BGP specification.

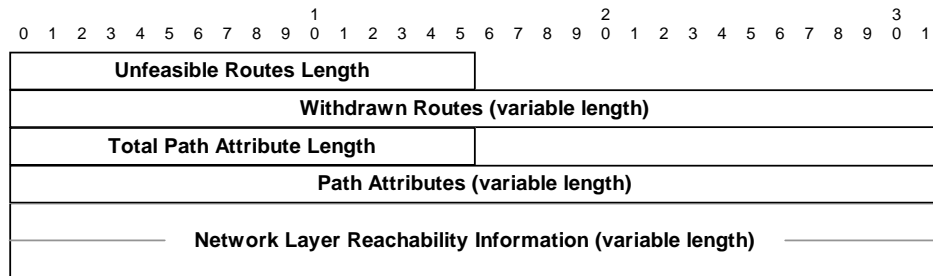
The Hold Time is the time interval within which a KEEPALIVE message must be dispatched to keep the BGP session active. If it is zero, the routers will not exchange KEEPALIVES. Otherwise, the Hold Time must be at least 3 seconds long. A proposed Hold Time of 1 or 2 seconds results in an error and termination of the BGP connection attempt. The smaller of the received and the speaker's Hold Times is selected as the session Hold Time.

The BGP Identifier, the return IP address of the peer, must be a valid IP address.

3.1.3 UPDATE Message

An UPDATE message contains 3 components: Unfeasible Routes, Path Attributes, and NLRI.

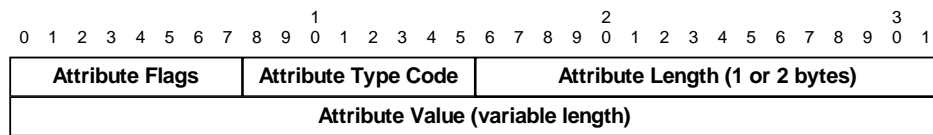
Figure 3: UPDATE message format.



The length of the Unfeasible Routes and all path attributes must not exceed the total length of the UPDATE message.

The format of a Path Attribute is shown in Figure 4. Associated with each Path Attribute are Attribute Flags, Attribute Type Code, Attribute Length, and Attribute Value fields.

Figure 4: Path Attribute format.



Each Attribute Type is tied to certain flag settings. These settings describe the attribute:

- Optional flag:
 - If set to 0, the attribute is a mandatory (or well-known) attribute and must be supported by the router.
 - If set to 1, the attribute is an optional one.

- Transitive flag:
 - If set to 0, the attribute may be dropped if the router does not support it.
 - If set to 1, the attribute must be forwarded when propagating routing information. The transitive flag must be set to 1 for well-known attributes.
- Partial flag:
 - If set to 0, the information for the attribute is complete: all routers along the path updated the attribute as necessary. The partial flag must be 0 for mandatory and non-transitive attributes.
 - If set to 1, the information for the associated optional transitive attribute is incomplete: a router along the path forwarded the attribute without updating it.
- Extended bit flag:
 - If set to 0, the Length of the Path Attribute is 1 octet long.
 - If set to 1, the length of the Path Attributes exceeds 255 bytes and thus the Path Attribute Length field occupies 2 octets.

The following Path Attributes are possible:

- Type 1: Origin
- Type 2: AS Path
- Type 3: Next Hop
- Type 4: Multi-Exit Discriminator
- Type 5: Local Preference
- Type 6: Atomic Aggregate
- Type 7: Aggregator
- Type 8: Community
- Type 9: Cluster List
- Type 10: Originator ID

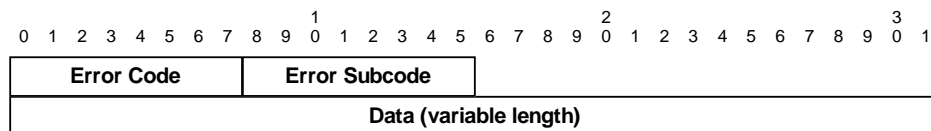
Each Path Attribute is verified for the correct flag settings, syntax, and semantics.

Path Attributes are continually being added and specified in Internet drafts to further describe routing information. The complete list of valid Attribute Types can be found in [3].

3.1.4 NOTIFICATION Message

The NOTIFICATION message contains 2 mandatory fields and an optional field containing the erroneous data. An error subcode and error data may provide further information about the nature of the error. Acceptable values for the Error Code and Error Subcode are specified in [15]. Any unrecognised values result in an error.

Figure 5: NOTIFICATION message format.



3.1.5 KEEPALIVE Message

BGP peers exchange these short KEEPALIVE messages as acknowledgements. The KEEPALIVE contains no further information other than its Message Type in the BGP header; thus, a received KEEPALIVE must be exactly 19 bytes long in total.

3.2 OPEN Process and Negotiation of Optional Parameters

3.2.1 OPEN Process

Not all BGP speakers may support the same parameters or feature set, thus negotiation of these parameters occurs during the OPEN process.

The OPEN process is a three-way handshaking procedure, depicted in Figure 6. Each BGP speaker sends an OPEN message with its desired settings and features. When the

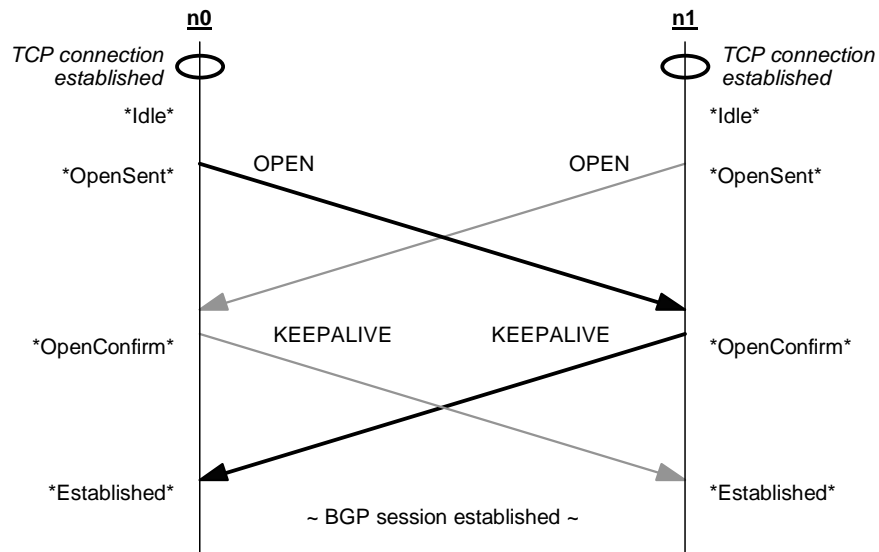
neighbour's OPEN message is received, the message fields are checked for errors, then for compatibility with the receiving speaker's.

Both BGP speakers must agree upon the set of parameters mentioned in Section 3.1.2 to configure the peer session. These parameters will dictate the set of laws that governs the peer session, and the optional parameters determine what capabilities the speakers can support.

If all parameters are acceptable, a KEEPALIVE is sent to the peer node to acknowledge them. Once a KEEPALIVE is received in return (the peer is also okay with its received parameters), the BGP connection is established.

Note that the vertical lines denote the lapse of time.

Figure 6: Ideal OPEN process.



If a parameter value is disagreeable, the BGP speaker sends a NOTIFICATION message and terminates the session. A BGP connection may be reattempted with a revised set of parameters.

Figure 7 shows a failed OPEN process in which BGP peer node 0 (n0) finds a value from its neighbour n1 unacceptable. No attempt is made to reestablish the BGP connection.

Figure 7: Failed OPEN process.

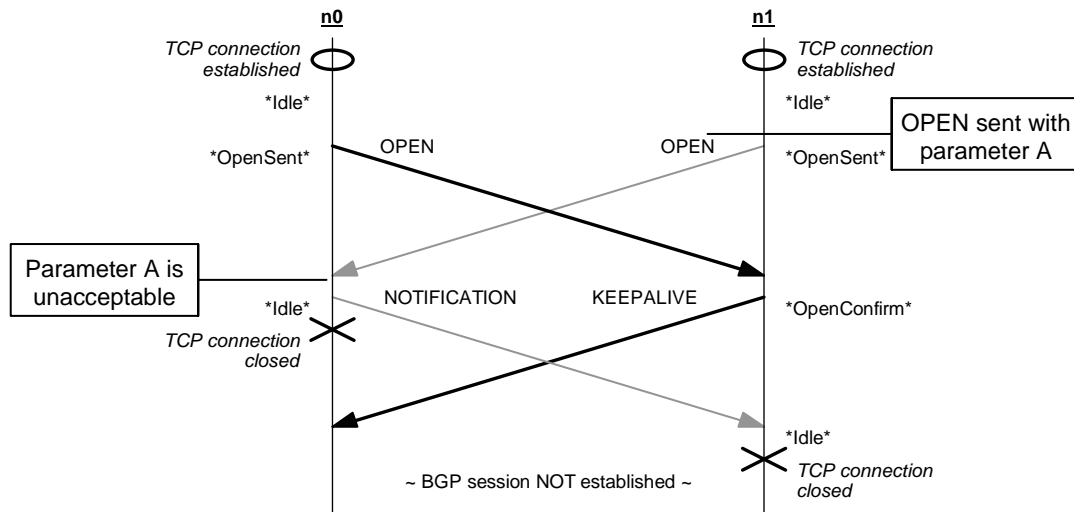


Figure 8 and Figure 9 show failed OPEN processes with reconnection attempts. In the first case, n0 alone finds a parameter unacceptable. In the second, n0 and n1 are presumed to send one parameter each (B and A, respectively), and both nodes object to the other's parameter. In both cases, after the failed initial attempt, a second attempt to set up the BGP session is made with a revised parameter set, and the ensuing message exchange results in a successfully established connection.

Figure 8: Failed OPEN process with reconnection attempt: one peer objects.

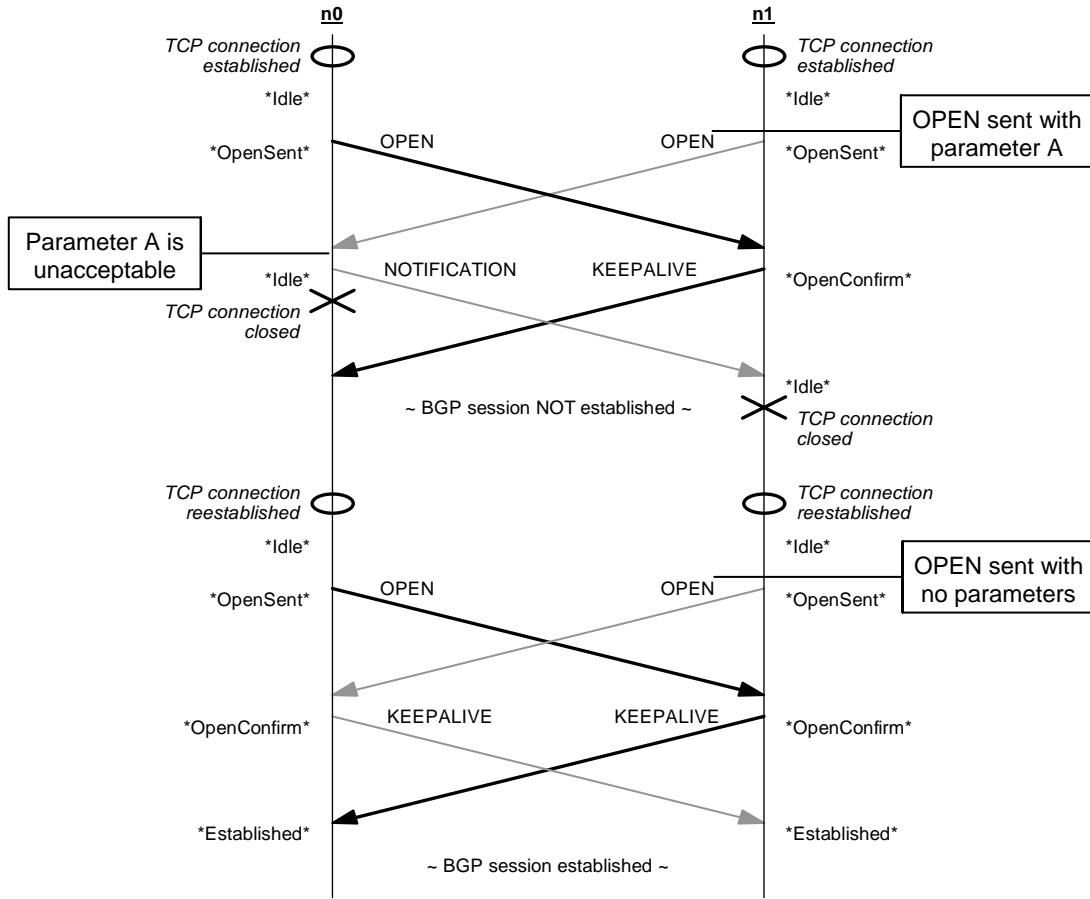
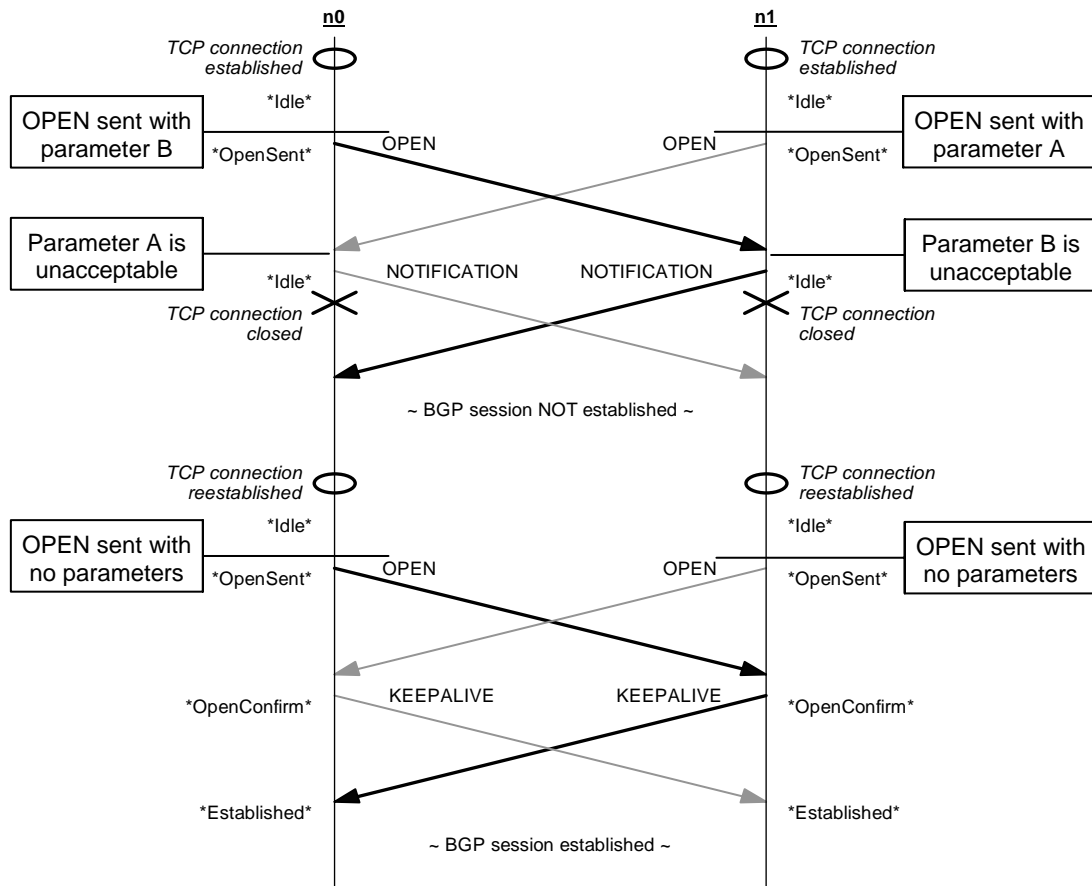


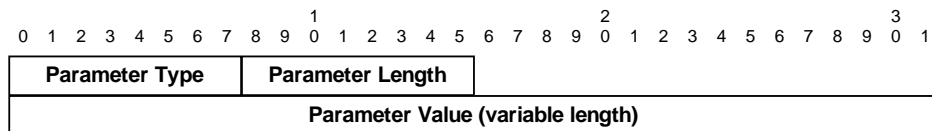
Figure 9: Failed OPEN process with reconnection attempt: both peers object.



3.2.2 Optional Parameters

The use of optional parameters is negotiated between the peering BGP speakers during the OPEN process. The format of an Optional Parameter is shown in Figure 10.

Figure 10: Optional Parameter format.



Currently, two types of optional parameters are recognised:

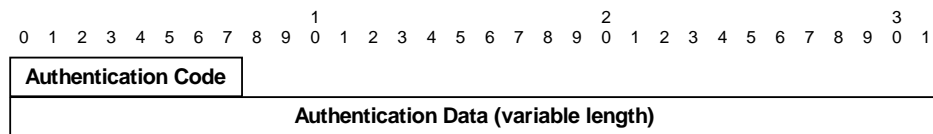
- Type 1: Authentication Information [15], which contains the data for a specific authentication mechanism; and,
- Type 2: Capabilities Advertisement [7], which communicates new features that a BGP speaker supports and may wish to employ during a session.

The existing ns-BGP model assumed that no optional parameters were included in the OPEN message, fixing the total message length to 29 bytes. Thus both the verification of and the negotiation of optional parameters were absent from this recently implemented BGP model. This project adds the capability to send, to receive, and to negotiate the Authentication Information and Capabilities Advertisement optional parameters described below.

3.2.2.1 Authentication Information

An authentication mechanism may be selected by the BGP peers. The optional parameter value, shown in Figure 10, will contain the Authentication Code, the semantics of the Authentication Data, and the algorithm for computing Marker field values. If the authentication mechanism is selected, the value of the Marker in the BGP header will be a predictable, computable value that can be verified by the receiving BGP peer.

Figure 11: Authentication Information optional parameter format.



Although the Authentication Information optional parameter has been specified in the BGP-4 RFC, this option has not yet been implemented by any network equipment vendors. [2]

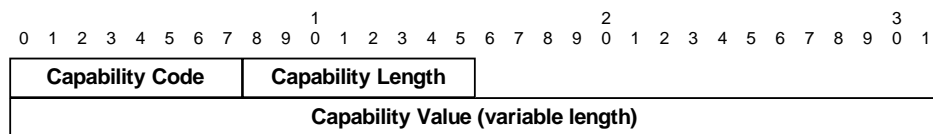
3.2.2.2 Capabilities Advertisement

As the Internet grows, so does the demand for additional features. The current implementation of BGP is such that, upon receipt of an unknown message type or a message with unfamiliar parameters, a NOTIFICATION message is sent and the BGP peer session terminated. Such a stipulation complicates the introduction of new features and capabilities into BGP.

The issue of new capabilities is resolved by the introduction of the Capabilities Advertisement optional parameter. This parameter enables a BGP speaker to advertise the features it supports and to identify those supported by its peer, allowing the two BGP speakers to negotiate which of these capabilities to employ during the session.

As shown in Figure 12 below, the Capabilities Advertisement optional parameter takes on a similar form to the Optional Parameter. A Capability Code identifies the type of capability being advertised, and a Capability Value provides the necessary data. Only one instance of a Capability Code may appear in the OPEN message.

Figure 12: Capabilities Advertisement optional parameter format.



Examples of capabilities that can be advertised are:

- Capability Code 1 – Multiprotocol Extensions,
- Capability Code 2 – Route Refresh Capability for BGP-4, and,
- Capability Code 65 – Support for 4-octet AS Numbers.

A current list of the assigned Capability Codes is available in [4].

3.3 NOTIFICATION Error Codes and Subcodes

Simplifications were made to the dispatched NOTIFICATION messages in both the SSF.OS.BGP4 and ns-BGP models. Both models sent Error Code and Subcode values of 0, regardless of the cause, and assumed no Error Data.

The Error Code indicates the area in which the error occurred, and the Error Subcode provides additional information about the nature of the error. This project alters the code so that the correct Error Code and Error Subcode are inserted and the sent NOTIFICATION message contains any requisite Error Data.

Table 1 on the next page lists the valid Error Codes and Subcodes.

Table 1: List of Error Codes and Subcodes for NOTIFICATION messages.

Error Code and Description	Error Subcodes and Descriptions
1 – Message Header Error	1 – Connection Not Synchronised 2 – Bad Message Length 3 – Bad Message Type
2 – OPEN Message Error	1 – Unsupported Version Number 2 – Bad Peer AS 3 – Bad BGP Identifier 4 – Unsupported Optional Parameter 5 – Authentication Failure 6 – Unacceptable Hold Time 7 – Unsupported Capability
3 – UPDATE Message Error	1 – Malformed Attribute List 2 – Unrecognised Well-known Attribute 3 – Missing Well-known Attribute 4 – Attribute Flags Error 5 – Attribute Length Error 6 – Invalid ORIGIN Attribute 7 – AS Routing Loop 8 – Invalid NEXT_HOP Attribute 9 – Optional Attribute Error 10 – Invalid Network Field 11 – Malformed AS_PATH
4 – Hold Timer Expired	(none)
5 – Finite State Machine Error	(none)
6 – Cease	(none)

Chapter 4: Project Implementation

The structure of the existing ns-BGP model was not modified as a result of this project. All implementations of error checking and message process were accomplished in pre-existing files. A list of the modified files is found in Appendix A.

This chapter outlines the implementations completed, along with assumptions, data formats, and test cases. The simulation results for these test cases are presented in Chapter 5.

4.1 BGP Header and Message Error Checking

Implementation of the BGP model in ns-2 entails coding in C++ and OTcl. The object-oriented nature of C++ confines all data to a particular data structure, as specified by the object class. Thus, any BGP message sent was created directly from an object of a specific construction, and any data received was previously assumed accurate and parsed into an object of the appropriate configuration.

The majority of the error checks, per Section 6 of [15], have been implemented. Unfortunately, since the network simulator is ideal, the verification of some tests could only be done locally: the error had to be hardcoded into the BGP message for it to occur!

Table 2 lists the error checks implemented for incoming BGP messages. The Bad Message Type (Error Code 1, Subcode 3) and AS Routing Loop (Error Code 3, Subcode 7) tests already existed in the BGP model. No method of determination was specified for a Bad Peer AS (Error Code 2, Subcode 2). Tests for Unsupported Optional Parameter (Error Code 2, Subcode 4) and Malformed AS_PATH (Error Code 3, Subcode 11) were not implemented due to time constraints.

Table 2: Implemented error checks for BGP messages.

Error Code and Description	Error Subcodes and Descriptions	Implemented
1 – Message Header Error	1 – Connection Not Synchronised 2 – Bad Message Length 3 – Bad Message Type	✓ ✓ ✗
2 – OPEN Message Error	1 – Unsupported Version Number 2 – Bad Peer AS 3 – Bad BGP Identifier 4 – Unsupported Optional Parameter 5 – Authentication Failure 6 – Unacceptable Hold Time 7 – Unsupported Capability	✓ ✗ ✓ ✗ ✓ ✓ ✓
3 – UPDATE Message Error	1 – Malformed Attribute List 2 – Unrecognised Well-known Attribute 3 – Missing Well-known Attribute 4 – Attribute Flags Error 5 – Attribute Length Error 6 – Invalid ORIGIN Attribute 7 – AS Routing Loop 8 – Invalid NEXT_HOP Attribute 9 – Optional Attribute Error 10 – Invalid Network Field 11 – Malformed AS_PATH	✓ ✓ ✓ ✓ ✓ ✓ ✗ ✓ ✓ ✓ ✗
4 – Hold Timer Expired	(none)	✓
5 – Finite State Machine Error	(none)	✓
6 – Cease	(none)	✓

4.2 NOTIFICATION Error Codes and Subcodes

When a verification test fails, a NOTIFICATION message is dispatched to the sender of the offending BGP message. Thus, Table 2 – the implementation of error checks – also pertains to the insertion of the corresponding Error Code, Error Subcode and any requisite Error Data into the resulting NOTIFICATION message.

4.3 Negotiation of Optional Parameters

The revised ns-BGP model has the ability to negotiate the use of the 2 Optional Parameters described in Section 3.2.2. Simulation results are presented in Chapter 5.

4.3.1 Authentication Mechanism

For the purpose of this document, an authentication mechanism is identified by the following syntax: [Authentication Code] / [Authentication Data], where [Authentication Code] is an integer code, and [Authentication Data] is the data as a string of characters.

Due to the lack of further specification, the following assumptions were made for the negotiation of an authentication mechanism:

1. Each BGP speaker can support at most one authentication mechanism.
2. The authentication information of both peers must match exactly in order for an authentication mechanism to be employed during the BGP session.
3. The same authentication mechanism, if any, is employed on both ends of the BGP session. That is, peer n0 cannot use mechanism A while peer n1 uses mechanism B.

Given these assumptions, the possible scenarios may be represented by 3 test cases, listed in Table 3. Scenarios are numbered for future reference.

Table 3: Negotiated authentication mechanism.

	Proposed Authentication Mechanism		Negotiated Authentication Mechanism
	n0	n1	
1	2/123	2/123	2/123
2	none	2/L	none
3	2/123	2/L	none

To arrive at the negotiated result, a BGP peer applies 2 algorithms, shown in the following figures. Figure 13 shows the procedure when an OPEN message is received, and Figure 14 presents the decision process for which Authentication Information to insert into an outgoing OPEN message. A Boolean variable `negotiated` is used to determine whether the authentication was previously negotiated. Variable `peer_authentication` keeps track of the proposed (received) authentication, and the authentication mechanism to use once the session is established.

Figure 13: Authentication negotiation algorithm when an OPEN message is received.

```
Set peer_authentication = received_authentication

If      negotiated == TRUE
  OR    received_authentication == own_authentication
  OR    no authentication proposed

Then
  Set negotiated = TRUE
  Continue rest of OPEN message processing without error

Otherwise
  Send NOTIFICATION message with received_authentication
```

Figure 14: Authentication negotiation algorithm when an OPEN message is sent.

```
If negotiated == TRUE
    Send OPEN message with peer_authentication

Else if peer_authentication still blank
    Send OPEN message with own_authentication

Else
    Send OPEN message without any Authentication Information
```

Once a mechanism is selected, a new Marker is used in the BGP message header when the BGP session becomes established. Since no vendor has yet implemented this option, legitimate authentication values are not yet available, thus an arbitrary value (of all 0x0Bs) was used so that the change in Marker value was immediately apparent.

The user also has the ability to set the Authentication Information for a BGP node using the following Tcl script commands (`$bgp_agent0` refers to BGP node 0):

- `$bgp_agent0 no-auth`
- `$bgp_agent0 set-auth [auth_code] [auth_data]`
where `[auth_data]` is a string such as "123" or a string variable:
`$bgp_agent0 set-auth 2 "123"`

4.3.2 Capabilities Advertisement

For the purpose of this document, a capability is identified by the following syntax: `[Capability Code] → [Capability Data]`, where `[Capability Code]` is an integer code, as assigned by the IANA [4], and `[Capability Data]` is the data as a string of characters.

As depicted in Table 4, the final negotiated set of capabilities is the intersection of both peers' capabilities. Therefore, scenarios fall under 2 categories: those in which the peers support overlapping capabilities, and those in which the capabilities are mutually exclusive.

Table 4: Negotiated capabilities.

	Proposed Capabilities		Negotiated Capabilities
	n0	n1	
4	none	2→mariners 3→angels	none
5	1→redsox 2→mariners	2→mariners 3→angels	2→mariners

The capabilities themselves are not implemented as part of this project. The effect of a negotiated capability is limited to its acceptance during the OPEN process.

The algorithms for negotiating supported capabilities, shown in Figure 15 and Figure 16, closely resemble the algorithms from Section 4.3.1. As with the authentication, a Boolean variable makes a note of whether the Optional Parameter was previously negotiated, and `peer_capabilities` record the capabilities proposed by the other peer and those that may be used once the session is established.

Figure 15: Capability negotiation algorithm when an OPEN message is received.

```
Set peer_capabilities = received_capabilities
If      negotiated == TRUE
  OR    received_capabilities ⊆ own_capabilities
Then
  Set negotiated = TRUE
  Continue rest of OPEN message processing without error
Otherwise
  Send NOTIFICATION message with received_authentication
```

Figure 16: Capability negotiation algorithm when an OPEN message is sent.

```
If negotiated == TRUE
    Send OPEN message with peer_capabilities

Else if peer_capabilities empty
    Send OPEN message with own_capabilities

Else
    Send OPEN message with (peer_capabilities  $\cap$  own_capabilities)
```

The user also has the ability to show, add, or remove capabilities for a BGP node using the following Tcl script commands (`$bgp_agent0` refers to BGP node 0):

- `$bgp_agent0 show-cap`
- `$bgp_agent0 add-cap [cap_code] [cap_data]`
where `[cap_data]` is a string such as "mariners" or a string variable:
`$bgp_agent0 add-cap 2 "mariners"`
- `$bgp_agent0 rmv-cap [cap_code] [cap_data]`

Per the Capabilities Advertisement specification [7], a BGP speaker should not include more than one instance of a Capability Code with identical Capability Length and Capability Value; but it may support multiple instances of a Capability Code, with different Capability Values.

Because a NOTIFICATION message carries only one Error Code and Subcode, if both Authentication Information and Capabilities Advertisement are included in an OPEN message, they are considered in that order: the authentication mechanism first, then the capabilities. The negotiation processes shown in Figure 8 and Figure 9 may be extended by a third attempt to establish the BGP connection if both the Authentication Information and Capabilities Advertisement parameters require negotiation.

Two scenarios (in Table 5) are carried out to verify the negotiation of an optional parameter combination. Scenario 6 provides a combination in which only one parameter is contest. Scenario 7 offers conflicting parameters for both Authentication Information and Capabilities Advertisements, and thus requires 2 further attempts to successfully establish a BGP connection: one to negotiate an authentication mechanism, and a second to negotiate capabilities.

Table 5: Negotiated authentication mechanism and capabilities.

	Proposed Optional Parameters		Negotiated Optional Parameters
	n0	n1	
6	2/123	2/123 2→mariners	2/123
7	2/123 1→redsox 2→mariners	2/L 2→mariners	2→mariners

Chapter 5: Simulation Results

Due to the length of the simulation output, only highlights of the results are presented below. Full simulation results are available in Appendix B.

5.1 Ideal Session Establishment

In an ideal OPEN process, as presented in Figure 6, a BGP connection is established on the first attempt. Neither BGP speaker objects to any parameters (mandatory or optional) proposed by the peer.

Figure 17 shows an excerpt of the simulation, where both peers send OPEN messages without any optional parameters. Therefore, only mandatory OPEN message fields are checked for validity).

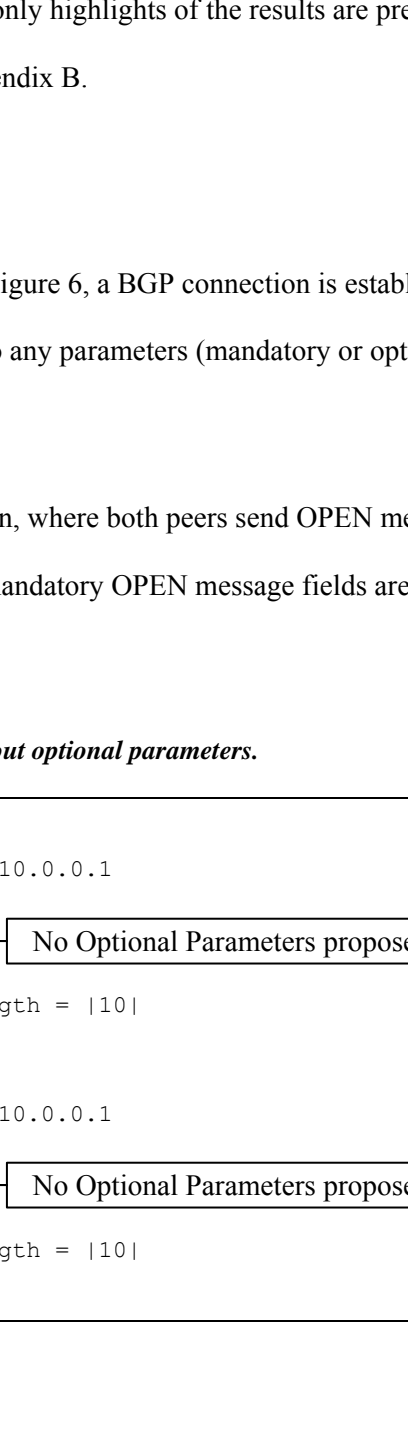
Figure 17: OPEN messages without optional parameters.

```
time: 0.0500096
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*

n1 creating OPEN MESG w/ length 10
Length of Opt Params = 10, OPEN Mesg Length = |10|
-> Total BGP Length is now |29|

time: 0.0500096
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*

n0 creating OPEN MESG w/ length 10
Length of Opt Params = 10, OPEN Mesg Length = |10|
-> Total BGP Length is now |29|
```



5.2 BGP Header and Message Error Checking

As mentioned in Section 4.1, the ideal simulator does not allow corruption to take place during message delivery, thus the errors in the BGP Header must be hardcoded to test their implementation and a full simulation (to an established BGP state) is not provided.

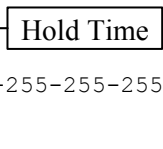
5.2.1 Unacceptable Hold Time Value

The Hold Time in an OPEN message must be 0 seconds or more than 3 seconds.

Figure 21 shows a connection attempt that is truncated as a result of a 2-second Hold Time.

Figure 21: Unacceptable Hold Time value.

```
n1 has received:
  Mesg type = 1, length = 29
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-29-1
  OpenMesg: 4-0-0-0-2-10-0-0-1-0
Checking if Marker is all 1's
non-zero Hold Timer value is less than the minimum recommended value 3s
(current_val = 2 s)
BGP session with peer 10.0.0.1 closed.
```

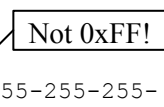


5.2.2 Unexpected Marker Field

If the Marker field does not hold the expected value – in this case, the default value of all 1-bits (0xFF bytes) – an error is generated and the BGP session is closed (Figure 22).

Figure 22: Unexpected Marker field.

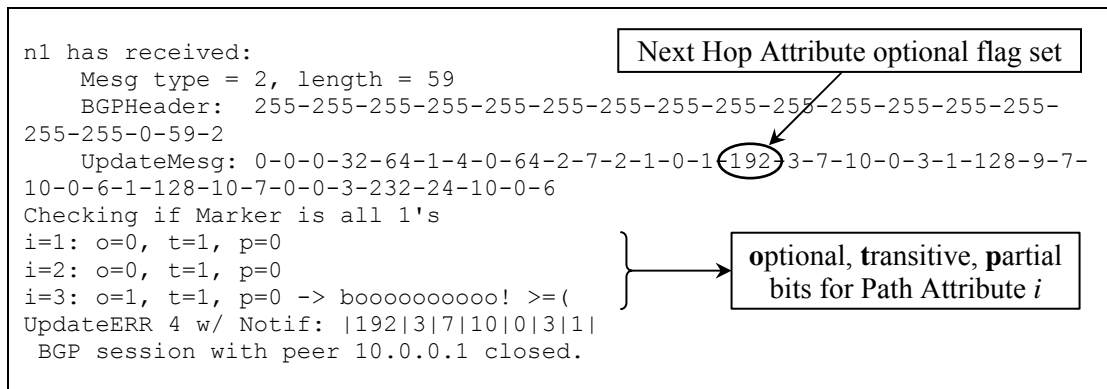
```
n1 has received:
  Mesg type = 1, length = 29
  BGPHeader: 255-255-255-255-255-255-255-255-255-10-255-255-255-
255-255-0-29-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-0
Checking if Marker is all 1's
The Marker field is not as expected.
BGP session with peer 10.0.0.1 closed.
```



5.2.3 Bad Path Attribute Flag

An UPDATE message with advertised routes (to propagate throughout the BGP network) includes Path Attributes that describe the routes. In Figure 23, Attribute Code 3 (Next Hop) has been flagged as an optional attribute, when it is in fact a mandatory one. Note the NOTIFICATION message generated with UpdateERR 4 (Error Code 3 for UPDATE Message Error; Subcode 4 for Attribute Flags Error).

Figure 23: Bad UPDATE Path Attribute flag.



5.3 OPEN Process Negotiation of Optional Parameters

The following sections confirm that the Authentication Information and Capabilities Advertisement Optional Parameters are properly negotiated during the OPEN process. Each scenario has different parameter settings, as described in Table 3, Table 4, and Table 5.

5.3.1 Scenario 1: Matching Authentication Mechanisms

The first order of business is to verify that an authentication mechanism is correctly applied, once negotiated. Thus, both BGP peers n0 and n1 insert the same Authentication Information 2/123 into their OPEN messages (Figure 24).

Figure 24: Scenario 1: OPEN messages.

```
time: 0.0500096
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

Authentication mechanism 2/123

```
n1 creating OPEN MESH w/ length 16
Inserting authentication into OPEN message:
  pType |1|, pLength |4|, aCode |2|, aData: 32:1 33:2 34:3
Length of Opt Params = |6|, OPEN Mesg Length = |16|
-> Total BGP Length is now |35|
```

```
time: 0.0500096
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

Authentication mechanism 2/123

```
n0 creating OPEN MESH w/ length 16
Inserting authentication into OPEN message:
  pType |1|, pLength |4|, aCode |2|, aData: 32:1 33:2 34:3
Length of Opt Params = |6|, OPEN Mesg Length = |16|
-> Total BGP Length is now |35|
```

Figure 25 shows that upon receipt of the OPEN messages, both peers agree to apply the proposed authentication mechanism. The BGP session is consequently established (not shown).

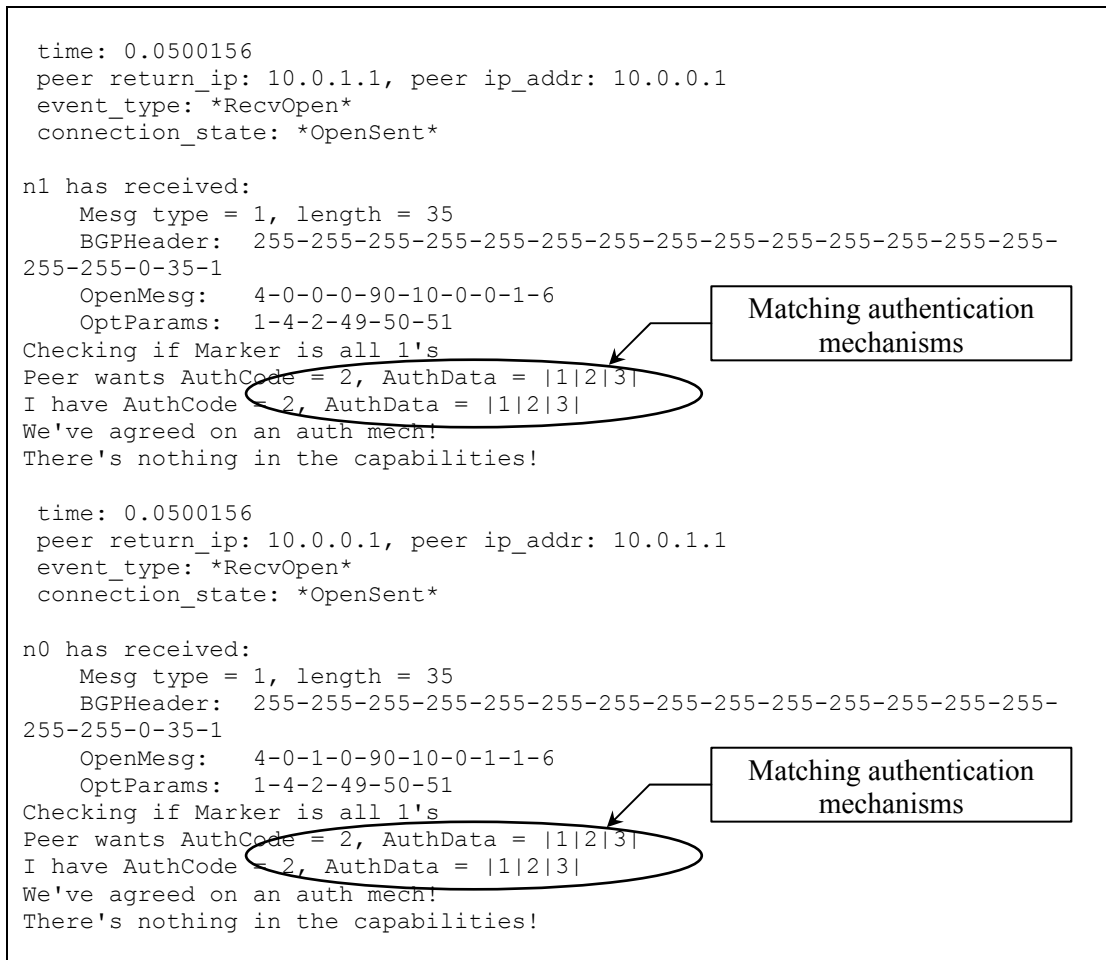
Figure 25: Scenario 1: OPEN messages received.

```
time: 0.0500156
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n1 has received:
  Mesg type = 1, length = 35
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-35-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-6
  OptParams: 1-4-2-49-50-51
Checking if Marker is all 1's
Peer wants AuthCode = 2, AuthData = |1|2|3|
I have AuthCode = 2, AuthData = |1|2|3|
We've agreed on an auth mech!
There's nothing in the capabilities!

time: 0.0500156
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n0 has received:
  Mesg type = 1, length = 35
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-35-1
  OpenMesg: 4-0-1-0-90-10-0-1-1-6
  OptParams: 1-4-2-49-50-51
Checking if Marker is all 1's
Peer wants AuthCode = 2, AuthData = |1|2|3|
I have AuthCode = 2, AuthData = |1|2|3|
We've agreed on an auth mech!
There's nothing in the capabilities!
```



Since both peers have agreed on an authentication mechanism, once the BGP session is established, all UPDATE, NOTIFICATION, and KEEPALIVE messages will be sent using a new Marker field value. To demonstrate that the proposed mechanism has indeed been engaged, the expected Marker field value for authentication mechanism 2/123 is set to all 0x0B bytes (decimal 11). Figure 26 confirms that the Marker field now contains the expected value and is checked against a new algorithm upon receipt.

While n1 does not object to the lack of an authentication mechanism, n0 does not agree with the one that n1 has sent and, closing the BGP connection (Figure 28), sending a NOTIFICATION message to n1.

Figure 28: Scenario 2: Conflicting authentication mechanisms.

```

time: 0.0500154
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n0 has received:
  Mesg type = 1, length = 33
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-33-1
  OpenMsg: 4-0-1-0-90-10-0-1-1-4
  OptParams: 1-2-2-76
Checking if Marker is all 1's
Peer wants AuthCode = 2, AuthData = |L|
I have AuthCode = 0, AuthData = |
  BGP session with peer 10.0.1.1 closed.
handle_open.auth: push to reconnect
  
```

When n1 receives n0's NOTIFICATION, the BGP connection attempt is halted and restarted using a revised parameter set, shown in Figure 29.

Figure 29: Scenario 2: OPEN messages with no authentication (second attempt).

```

time: 0.10003
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*

n1 creating OPEN MSG w/ length 10
Length of Opt Params = 10, OPEN Msg Length = |10|
-> Total BGP Length is now |29|

time: 0.100033
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*

n0 creating OPEN MSG w/ length 10
Length of Opt Params = 10, OPEN Msg Length = |10|
-> Total BGP Length is now |29|
  
```

On the second attempt, the session is successfully established. Figure 30 shows that the Marker field continues to hold the default value, confirming that a common authentication mechanism was not agreed upon and thus none is in effect.

Figure 30: Scenario 2: BGP session establishment with no authentication.

```
time: 0.100047
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvKeepAlive*
connection_state: *OpenConfirm*

n1 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-19-4
Checking if Marker is all 1's
BGP session with peer 10.0.0.1 established.

time: 0.100047
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvKeepAlive*
connection_state: *OpenConfirm*

n0 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-19-4
Checking if Marker is all 1's
BGP session with peer 10.0.1.1 established.

time: 29.1
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvKeepAlive*
connection_state: *Established*

n0 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-19-4
Checking if Marker is all 1's

time: 29.1
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvKeepAlive*
connection_state: *Established*

n1 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-19-4
Checking if Marker is all 1's
```

5.3.3 Scenario 3: Different Authentication Mechanisms

If both BGP speakers support authentication mechanisms that do not match, then no common mechanism can be negotiated. In Figure 31, peer n1 offers authentication mechanism 2/L, but on the other side, n0 inserts authentication mechanism 2/123.

Figure 31: Scenario 3: OPEN messages (first attempt).

```
time: 0.0500096
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*

n1 creating OPEN MESH w/ length 14
Inserting authentication into OPEN message:
  pType |1|, pLength |2|, aCode |2|, aData: 32:L
Length of Opt Params = |4|, OPEN Mesg Length = |14|
-> Total BGP Length is now |33|

time: 0.0500096
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*

n0 creating OPEN MESH w/ length 16
Inserting authentication into OPEN message:
  pType |1|, pLength |4|, aCode |2|, aData: 32:1 33:2 34:3
Length of Opt Params = |6|, OPEN Mesg Length = |16|
-> Total BGP Length is now |35|
```

Authentication mechanism 2/L

Authentication mechanism 2/123

The authentication mechanisms are rejected by both BGP peers (Figure 32), and a new BGP connection attempt is initiated.

Figure 32: Scenario 3: Authentication mechanisms rejected.

```
time: 0.0500154
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n0 has received:
  Mesg type = 1, length = 33
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-33-1
  OpenMesg: 4-0-1-0-90-10-0-1-1-4
  OptParams: 1-2-2-76
Checking if Marker is all 1's
Peer wants AuthCode = 2, AuthData = |L|
I have AuthCode = 2, AuthData = |1|2|3|
Auth data lengths don't match.
BGP session with peer 10.0.1.1 closed.
handle_open.auth: push to reconnect

time: 0.0500156
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n1 has received:
  Mesg type = 1, length = 35
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-35-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-6
  OptParams: 1-4-2-49-50-51
Checking if Marker is all 1's
Peer wants AuthCode = 2, AuthData = |1|2|3|
I have AuthCode = 2, AuthData = |L|
Auth data lengths don't match.
BGP session with peer 10.0.0.1 closed.
handle_open.auth: push to reconnect
```

Conflicting authentication mechanisms

Conflicting authentication mechanisms

As in Scenario 2, a new attempt to establish the BGP session is instigated (Figure 33).

Neither peer inserts Authentication Information into the revised OPEN messages.

Figure 33: Scenario 3: OPEN messages with no authentication (second attempt).

```
time: 0.100025
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

No Optional Parameters proposed

```
n1 creating OPEN MSG w/ length 10
Length of Opt Params = 10, OPEN Msg Length = |10|
-> Total BGP Length is now |29|

time: 0.100025
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

No Optional Parameters proposed

```
n0 creating OPEN MSG w/ length 10
Length of Opt Params = 10, OPEN Msg Length = |10|
-> Total BGP Length is now |29|
```

The second round of OPEN messages are processed, and as expected, the BGP session is established (Figure 34).

Figure 34: Scenario 3: BGP session establishment with no authentication.

```
time: 0.100039
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvKeepAlive*
connection_state: *OpenConfirm*
```

```
n1 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-19-4
Checking if Marker is all 1's
BGP session with peer 10.0.0.1 established.

time: 0.100039
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvKeepAlive*
connection_state: *OpenConfirm*
```

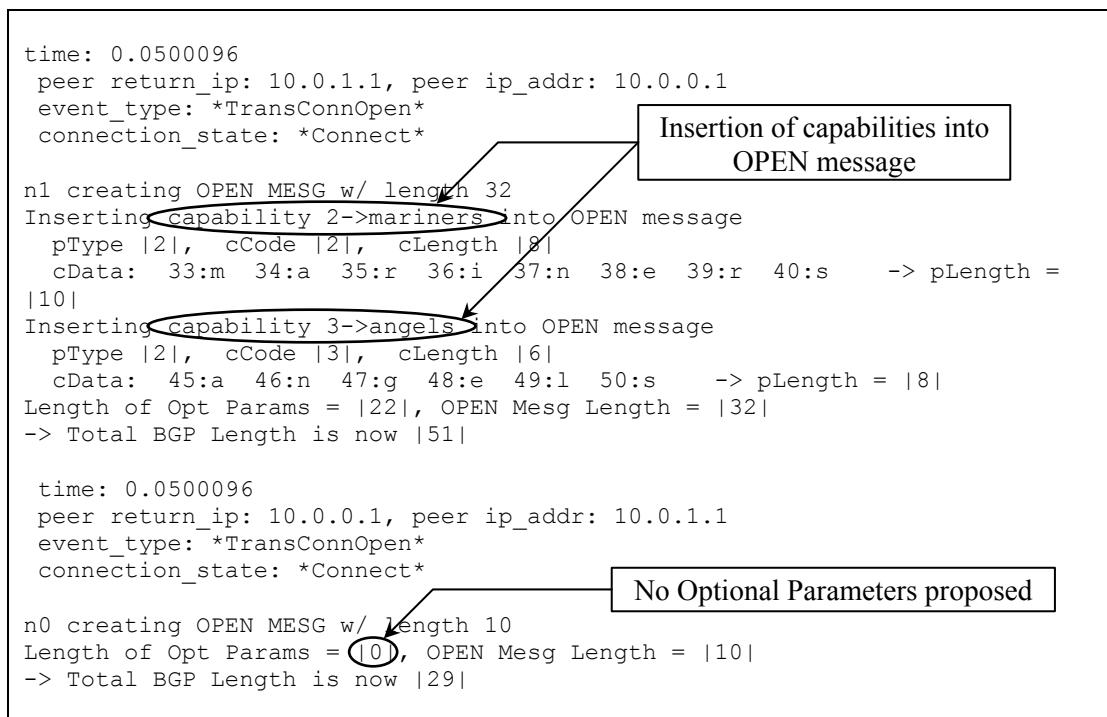
```
n0 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-19-4
Checking if Marker is all 1's
BGP session with peer 10.0.1.1 established.
```

5.3.4 Scenario 4: Mutually Exclusive Capability Sets

The negotiation process for Capabilities Advertisements is very similar to the Authentication Information, however, a set of parameters must be considered, rather than just one item.

This scenario considers the simple situation in which one peer supports multiple capabilities while the other supports none. Figure 35 illustrates the insertion of 2 capabilities (2→mariners and 3→angels) into n1's OPEN message. Peer n0 inserts none.

Figure 35: Scenario 4: OPEN messages (first attempt).



BGP peer n1 consents to using no capabilities, per n0's request (shown in Figure 36). However, peer n0 cannot support either of the capabilities that n1 has proposed, and therefore informs its peer by sending a NOTIFICATION with both Capabilities Advertisements in the Error Data field.

Figure 36: Scenario 4: Capabilities accepted by n1, rejected by n0.

```
time: 0.0500151
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n1 has received:
  Mesg type = 1, length = 29
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-29-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-0
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 0, AuthData = |
Nah, let's not use an auth mech.
There's nothing in the capabilities! ← No capabilities received

time: 0.0500169
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n0 has received:
  Mesg type = 1, length = 51
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-51-1
  OpenMesg: 4-0-1-0-90-10-0-1-1-22
  OptParams: 2-10-2-8-109-97-114-105-110-101-114-115-2-8-3-6-97-110-
103-101-108-115
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 0, AuthData = |
Nah, let's not use an auth mech.
2 caps included in the OPEN message: 2->mariners 3->angels
2 caps aren't in my caps: 2->mariners 3->angels
Inserting capability 2->mariners into NOTIF message, starting at nLen=0
  pType |2|, cCode |2|, cLength |8|
  cData: 4:m 5:a 6:r 7:i 8:n 9:e 10:r 11:s -> NotifData length
= |10|
Inserting capability 3->angels into NOTIF message, starting at nLen=12
  pType |2|, cCode |3|, cLength |6|
  cData: 16:a 17:n 18:g 19:e 20:l 21:s -> NotifData length = |8|
BGP session with peer 10.0.1.1 closed.
handle_open.caps: push to reconnect
```

As in the previous examples, the BGP connection is reattempted: both peers transmit OPEN messages without any Capabilities Advertisement (the intersection of both their sets of capabilities). Figure 37 shows the second attempt at a BGP connection.

Figure 37: Scenario 4: OPEN messages with no capabilities (second attempt).

```
time: 0.100033
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

No Optional Parameters proposed

```
n1 creating OPEN MSG w/ length 10
Length of Opt Params = 10, OPEN Msg Length = |10|
-> Total BGP Length is now |29|

time: 0.100036
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

No Optional Parameters proposed

```
n0 creating OPEN MSG w/ length 10
Length of Opt Params = 10, OPEN Msg Length = |10|
-> Total BGP Length is now |29|
```

With no capabilities to negotiate the second time around, the BGP session is established (shown in Figure 38), with no capabilities agreed upon.

Figure 38: Scenario 4: BGP session establishment with no capabilities.

```
time: 0.100053
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvKeepAlive*
connection_state: *OpenConfirm*
```

```
n0 has received:
  Msg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-19-4
Checking if Marker is all 1's
BGP session with peer 10.0.1.1 established.

time: 0.100053
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvKeepAlive*
connection_state: *OpenConfirm*
```

```
n1 has received:
  Msg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-19-4
Checking if Marker is all 1's
BGP session with peer 10.0.0.1 established.
```

5.3.5 Scenario 5: Overlapping Capability Sets

In this scenario, each peer supports 2 capabilities, one of them a common one. BGP speaker n0 supports 1→redsox and 2→mariners, and n1 supports 2→mariners and 3→angels. These capabilities are advertised in their OPEN messages (Figure 39).

Figure 39: Scenario 5: OPEN messages (first attempt).

```
time: 0.0500096
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*

n1 creating OPEN MSG w/ length 32
Inserting capability 2->mariners into OPEN message
  pType |2|, cCode |2|, cLength |8|
  cData: 33:m 34:a 35:r 36:i 37:n 38:e 39:r 40:s -> pLength =
|10|
Inserting capability 3->angels into OPEN message
  pType |2|, cCode |3|, cLength |6|
  cData: 45:a 46:n 47:g 48:e 49:l 50:s -> pLength = |8|
Length of Opt Params = |22|, OPEN Mesg Length = |32|
-> Total BGP Length is now |51|

time: 0.0500096
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*

n0 creating OPEN MSG w/ length 32
Inserting capability 1->redsox into OPEN message
  pType |2|, cCode |1|, cLength |6|
  cData: 33:r 34:e 35:d 36:s 37:o 38:x -> pLength = |8|
Inserting capability 2->mariners into OPEN message
  pType |2|, cCode |2|, cLength |8|
  cData: 43:m 44:a 45:r 46:i 47:n 48:e 49:r 50:s -> pLength =
|10|
Length of Opt Params = |22|, OPEN Mesg Length = |32|
-> Total BGP Length is now |51|
```

The diagram consists of two callout boxes, each containing the text "Insertion of capabilities into OPEN message". In the first log block (for peer n1), two arrows point from the callout box to the lines "Inserting capability 2->mariners into OPEN message" and "Inserting capability 3->angels into OPEN message". In the second log block (for peer n0), two arrows point from the callout box to the lines "Inserting capability 1->redsox into OPEN message" and "Inserting capability 2->mariners into OPEN message".

As the peers process the received OPEN messages (Figure 40), n1 encounters one capability that it does not support: 1→redsox. Meanwhile, n0 opposes the capability 3→angels. Both peers close their ends of the BGP session.

Figure 41 shows the ensuing second attempt to establish the connection: the common capability 2→mariners is inserted into the OPEN message on both sides, as before. However peer n0 omits capability 1→redsox, and n1 excludes capability 3→angels.

Figure 41: Scenario 5: OPEN messages with revised capabilities (second attempt).

```
time: 0.100026
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

n0 creating OPEN MESH w/ length 22
Inserting capability 2->mariners into OPEN message
pType |2|, cCode |2|, cLength |8|
cData: 33:m 34:a 35:r 36:i 37:n 38:e 39:r 40:s -> pLength =
|10|
Length of Opt Params = |12|, OPEN Mesg Length = |22|
-> Total BGP Length is now |41|

n1 creating OPEN MESH w/ length 22
Inserting capability 2->mariners into OPEN message
pType |2|, cCode |2|, cLength |8|
cData: 33:m 34:a 35:r 36:i 37:n 38:e 39:r 40:s -> pLength =
|10|
Length of Opt Params = |12|, OPEN Mesg Length = |22|
-> Total BGP Length is now |41|

The second attempt to set up a BGP session is successful, shown in Figure 42. The single common capability inserted into the OPEN messages is accepted by both peers, and the session proceeds to the Established state.

Figure 42: Scenario 5: BGP session establishment with one capability.

```
time: 0.100033
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n0 has received:
  Mesg type = 1, length = 41
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-41-1
  OpenMesg: 4-0-1-0-90-10-0-1-1-12
  OptParams: 2-10-2-8-109-97-114-105-110-101-114-115
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 0, AuthData = |
Nah, let's not use an auth mech.
1 caps included in the OPEN message: 2->mariners
0 caps aren't in my caps: ← No conflicting capabilities

time: 0.100033
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n1 has received:
  Mesg type = 1, length = 41
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-41-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-12
  OptParams: 2-10-2-8-109-97-114-105-110-101-114-115
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 0, AuthData = |
Nah, let's not use an auth mech.
1 caps included in the OPEN message: 2->mariners
0 caps aren't in my caps: ← No conflicting capabilities

time: 0.100041
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvKeepAlive*
connection_state: *OpenConfirm*

n0 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-19-4
Checking if Marker is all 1's
BGP session with peer 10.0.1.1 established.

time: 0.100041
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvKeepAlive*
connection_state: *OpenConfirm*

n1 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-19-4
Checking if Marker is all 1's
BGP session with peer 10.0.0.1 established.
```

5.3.6 Scenario 6: Both Optional Parameters – One Negotiation Required

This example considers the situation in which one type of Optional Parameter (the Authentication Information) is accepted by both peers, while the other Optional Parameter (Capabilities Advertisement) requires negotiation.

Figure 43 shows the construction of the OPEN messages, with n1 inserting 2/123 and 2→mariners into the Optional Parameters, and n0 inserting only 2/123.

Figure 43: Scenario 6: OPEN messages (first attempt).

```
time: 0.0500096
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*

n1 creating OPEN MSG w/ length 28
Inserting authentication into OPEN message:
  pType |1|, pLength |4|, aCode |2|, aData: 32:1 33:2 34:3
Inserting capability 2->mariners into OPEN message
  pType |2|, cCode |2|, cLength |8|
  cData: 39:m 40:a 41:r 42:i 43:n 44:e 45:r 46:s -> pLength =
|10|
Length of Opt Params = |18|, OPEN Msg Length = |28|
-> Total BGP Length is now |47|

time: 0.0500096
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*

n0 creating OPEN MSG w/ length 16
Inserting authentication into OPEN message:
  pType |1|, pLength |4|, aCode |2|, aData: 32:1 33:2 34:3
Length of Opt Params = |6|, OPEN Msg Length = |16|
-> Total BGP Length is now |35|
```

As expected, the Authentication Information is received without objection, but n0 protests the capability 2→mariners (Figure 44). The BGP connection attempt is terminated and restarted.

Figure 45: Scenario 6: OPEN messages with revised parameters (second attempt).

```

time: 0.100032
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*

n1 creating OPEN MSG w/ length 16
Inserting authentication into OPEN message:
  pType |1|, pLength |4|, aCode |2|, aData: 32:1 33:2 34:3
Length of Opt Params = |6|, OPEN Msg Length = |16|
-> Total BGP Length is now |35|

time: 0.100035
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*

n0 creating OPEN MSG w/ length 16
Inserting authentication into OPEN message:
  pType |1|, pLength |4|, aCode |2|, aData: 32:1 33:2 34:3
Length of Opt Params = |6|, OPEN Msg Length = |16|
-> Total BGP Length is now |35|

```

With an identical parameter set, the OPEN messages are processed without further objections, and the BGP session is established. Figure 46 shows that the new authentication mechanism is now in effect.

Figure 46: Scenario 6: Different Marker field used.

```

time: 29.1
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvKeepAlive*
connection_state: *Established*

n0 has received:
  Mesg type = 4, length = 19
  BGPHeader: 11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-0-19-4
Using Another_AuthCode to test the Marker

time: 29.1
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvKeepAlive*
connection_state: *Established*

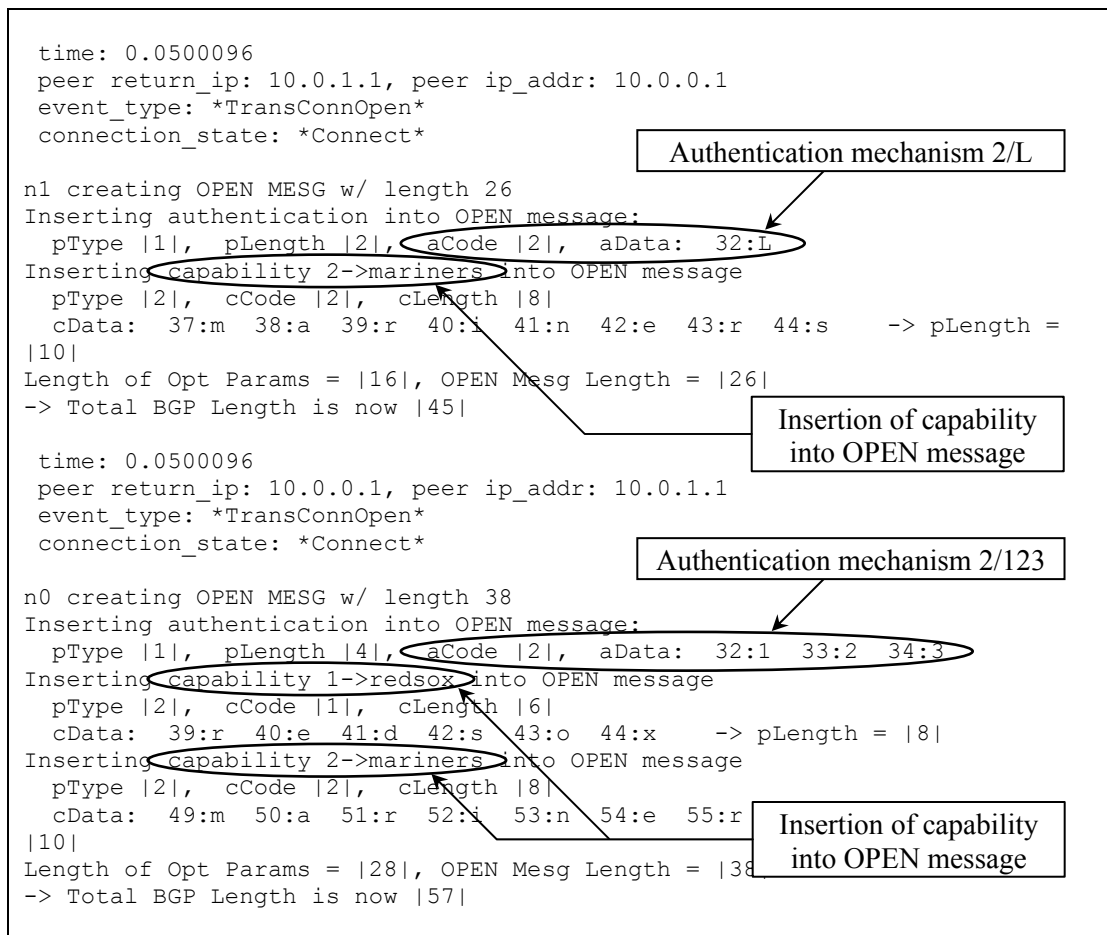
n1 has received:
  Mesg type = 4, length = 19
  BGPHeader: 11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-0-19-4
Using Another_AuthCode to test the Marker

```

5.3.7 Scenario 7: Both Optional Parameters – Two Negotiations Required

When both types of Optional Parameters require negotiation, the OPEN process of Figure 9 is extended to include a third attempt at session establishment. In the following scenario, both peers n0 and n1 send Authentication Information and Capabilities Advertisement Optional Parameters. Peer n0 inserts authentication mechanism 2/123 and capabilities 1→redsox and 2→mariners into its OPEN message (Figure 47); and n1 inserts authentication mechanism 2/L and capabilities 2→mariners and 3→angels into its proposed parameters.

Figure 47: Scenario 7: OPEN messages (first attempt).



When the OPEN messages are received, the conflicting authentication mechanisms are detected (Figure 48) on both ends of the connection, and the BGP session attempt is truncated, without first also processing the Capabilities Advertisement Optional Parameters.

Figure 48: Scenario 7: Authentication mechanisms rejected (first attempt).

```
time: 0.0500164
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n0 has received:
  Mesg type = 1, length = 45
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-45-1
  OpenMesg: 4-0-1-0-90-10-0-1-1-16
  OptParams: 1-2-2-76-2-10-2-8-109-97-114-105-110-101-114-115
Checking if Marker is all 1's
Peer wants AuthCode = 2, AuthData = |L|
I have AuthCode = 2, AuthData = |1|2|3|
Auth data lengths don't match.
BGP session with peer 10.0.1.1 closed.
handle_open.auth: push to reconnect

time: 0.0500174
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n1 has received:
  Mesg type = 1, length = 57
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-57-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-28
  OptParams: 1-4-2-49-50-51-2-8-1-6-114-101-1
109-97-114-105-110-101-114-115
Checking if Marker is all 1's
Peer wants AuthCode = 2, AuthData = |1|2|3|
I have AuthCode = 2, AuthData = |L|
Auth data lengths don't match.
BGP session with peer 10.0.0.1 closed.
handle_open.auth: push to reconnect
```

Conflicting authentication mechanisms

Conflicting authentication mechanisms

In Figure 49, a second attempt is initiated without the Authentication Information Optional Parameter. The Capabilities Advertisements sent in the first attempt's OPEN message are reinserted into the new OPEN messages, unrevised.

Figure 49: Scenario 7: OPEN messages (second attempt).

```
time: 0.100027
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*

n1 creating OPEN MESH w/ length 22
Inserting capability 2->mariners into OPEN message
  pType |2|, cCode |2|, cLength |8|
  cData: 33:m 34:a 35:r 36:i 37:n 38:e 39:r 40:s -> pLength =
|10|
Length of Opt Params = |12|, OPEN Mesg Length = |22|
-> Total BGP Length is now |41|

time: 0.100027
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*

n0 creating OPEN MESH w/ length 32
Inserting capability 1->redsox into OPEN message
  pType |2|, cCode |1|, cLength |6|
  cData: 33:r 34:e 35:d 36:s 37:o 38:x -> pLength = |8|
Inserting capability 2->mariners into OPEN message
  pType |2|, cCode |2|, cLength |8|
  cData: 43:m 44:a 45:r 46:i 47:n 48:e 49:r 50:s -> pLength =
|10|
Length of Opt Params = |22|, OPEN Mesg Length = |32|
-> Total BGP Length is now |51|
```

Insertion of capabilities into OPEN message

Insertion of capabilities into OPEN message

This time, the new Authentication Information Optional Parameter (none) is accepted, but the Capabilities Advertisements, which were previously not negotiated, encounter conflicts (Figure 50). Peer n1 supports only one of n0's proposed capabilities. The BGP session attempt is abandoned a second time, and a third attempt is initiated using revised capabilities set.

Figure 50: Scenario 7: Capabilities rejected (second attempt).

```
time: 0.100033
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n0 has received:
  Mesg type = 1, length = 41
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-41-1
  OpenMesg: 4-0-1-0-90-10-0-1-1-12
  OptParams: 2-10-2-8-109-97-114-105-110-101-114-115
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 2, AuthData = |1|2|3|
1 caps included in the OPEN message: 2->mariners
0 caps aren't in my caps: ← No conflicting capabilities

time: 0.100034
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n1 has received:
  Mesg type = 1, length = 51
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-
255-255-0-51-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-22
  OptParams: 2-8-1-6-114-101-100-115-111-120-2-10-2-8-109-97-114-105-
110-101-114-115
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 2, AuthData = |L|
2 caps included in the OPEN message: 1->redsox 2->mariners
1 caps aren't in my caps: 1->redsox
Inserting capability 1->redsox into NOTIF message, starting at nLen=0
pType |2|, cCode |1|, cLength |6|
cData: 4:r 5:e 6:d 7:s 8:o 9:x -> NotifData length = |8|
BGP session with peer 10.0.0.1 closed.
handle_open.caps: push to reconnect
```

The third attempt's OPEN messages now contain the previously negotiated Authentication Information (none) and the newly negotiated Capabilities Advertisements (the intersection of both peers' sets of capabilities: 2→mariners). Figure 51 shows the re-revised OPEN messages.

Figure 51: Scenario 7: OPEN messages (third attempt).

```
time: 0.15005
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*

n0 creating OPEN MSG w/ length 22
Inserting capability 2->mariners into OPEN message
  pType |2|, cCode |2|, cLength |8|
  cData: 33:m 34:a 35:r 36:i 37:n 38:e 39:r 40:s -> pLength =
|10|
Length of Opt Params = |12|, OPEN Mesg Length = |22|
-> Total BGP Length is now |41|

time: 0.150053
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*

n1 creating OPEN MSG w/ length 22
Inserting capability 2->mariners into OPEN message
  pType |2|, cCode |2|, cLength |8|
  cData: 33:m 34:a 35:r 36:i 37:n 38:e 39:r 40:s -> pLength =
|10|
Length of Opt Params = |12|, OPEN Mesg Length = |22|
-> Total BGP Length is now |41|
```

Insertion of common capability into OPEN message

Insertion of common capability into OPEN message

Finally, the BGP session is established. Figure 52 confirms that after 3 attempts to set up the BGP session, the Optional Parameters are properly negotiated and both peers arrive at a common set of parameters by which to govern the BGP connection.

Chapter 6: Future Enhancements

“Little by little does the trick.” – Aesop

In an ongoing project such as network protocol development, enhancements can always be made to improve code efficiency and scalability, and to increase the capabilities of a network model.

6.1 Completion of BGP Header and Message Error Checking

The majority of the verification tests listed in Table 2 have been implemented; however, a few errors checks still remain to be programmed. A simple enhancement would be to develop the remaining tests (marked with an X).

6.2 Restructuring and Addition of Optional Parameters

The Optional Parameters are correctly implemented and inserted into the OPEN message, and function according to specifications [15] and [7]. However, the current implementation is not the most efficient. Due to unfamiliarity with object-oriented programming, the Optional Parameters were written in a linear fashion, directly in the finite state machine program file `rtProtoBGP.cc`. While handling 2 Optional Parameters is still manageable, as BGP evolves, the number of Optional Parameters may grow very rapidly. Restructuring the Optional Parameter to take full advantage of the network simulator’s object-oriented nature would ensure that future expansions to the ns-BGP model could be made without cluttering the main program file.

6.3 Support for Different Authentication Mechanisms in the Same BGP Session

The project implementation of the Authentication Information negotiation made a few assumptions, mentioned in Section 4.3.1. However, there are no such assumptions made in the BGP specification. Thus, future work may include the removal of these assumptions to allow for more flexibility in the assignment and use of authentication mechanisms.

6.4 Addition of New Capabilities Advertisements

With the ever-growing list of capabilities for BGP, there exists the need to update the simulation model's feature set. The BGP model should not only recognise a Capability Code as being valid, but also process them and employ the capability itself. Many capabilities have already been developed, per [4]. The implementation of these new capabilities would enrich the networking community with a more complete tool with which to explore the capabilities and limitations of BGP.

6.5 Addition of New Path Attributes

As with the capabilities, new Path Attributes found in UPDATE messages are also being developed constantly. The performance of BGP will change with the availability of new information about advertised routes, making the implementation of new Path Attributes an important addition to the BGP model.

6.6 Handling of NULL Characters

The data for the Authentication Information and Capabilities Advertisement values are implemented as strings of characters, thus the NULL character (0x00) is used to delimit the data string, and cannot be used as a field value.

Chapter 7: Conclusion

Border Gateway Protocol (BGP) is the *de facto* inter-domain routing protocol, used by the Internet worldwide. Its performance and scalability have become of particular interest to the communication networks community, fuelling more investigation and research into BGP's design and improvement.

To advance the research and development of communication networks in general, simulation tools and network models have been developed, including SSFNet and ns-2. These network simulators contain a BGP model used to replicate the behaviour of a BGP router in a network environment.

The purpose of this project was to expand the capabilities of ns-BGP and to improve the model's accuracy according to the BGP specification. Among the changes were BGP header and message error checking, and the implementation and negotiation of the Authentication Information and Capabilities Advertisement Optional Parameters in the OPEN message.

These modifications to ns-BGP will ensure that the model grows as the protocol does, with the ability to scale and handle new parameters and capabilities.

Appendix A: List of Modified Files

Many of the existing ns-BGP files were modified to accomplish this project. The following files are all located in the ns-BGP directory bgp/.

- peer-entry.cc
- peer-entry.h
- rtProtoBGP.cc
- rtProtoBGP.h
- Comm/bgpmessage.cc
- Comm/bgpmessage.h
- Comm/keepalivemessage.cc
- Comm/keepalivemessage.h
- Comm/notificationmessage.cc
- Comm/notificationmessage.h
- Comm/openmessage.cc
- Comm/openmessage.h
- Path/aggregator.cc
- Path/aggregator.h
- Path/aspath.cc
- Path/aspath.h
- Path/atomicaggregate.cc
- Path/atomicaggregate.h
- Path/attribute.cc
- Path/attribute.h
- Path/clusterlist.cc
- Path/clusterlist.h
- Path/community.cc
- Path/community.h
- Path/localpref.cc

- Path/localpref.h
- Path/med.cc
- Path/med.h
- Path/nexthop.cc
- Path/nexthop.h
- Path/originatorid.cc
- Path/originatorid.h
- Path/origin.cc
- Path/origin.h

Appendix B: Simulation Results in Full

B.1 Ideal Negotiation Process

Simulation starts...

```
time: 0.05
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.05
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.0500096
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

n1 creating OPEN MESH w/ length 10
Length of Opt Params = |0|, OPEN Mesg Length = |10|
-> Total BGP Length is now |29|

```
time: 0.0500096
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

n0 creating OPEN MESH w/ length 10
Length of Opt Params = |0|, OPEN Mesg Length = |10|
-> Total BGP Length is now |29|

Figure 17

```
time: 0.0500151
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*
```

n1 has received:
 Mesg type = 1, length = 29
 BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
29-1
 OpenMesg: 4-0-0-0-90-10-0-0-1-0
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 0, AuthData = |
Nah, let's not use an auth mech.
There's nothing in the capabilities!

```
time: 0.0500151
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*
```

Figure 18

```
n0 has received:
  Mesg type = 1, length = 29
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
29-1
  OpenMesg: 4-0-1-0-90-10-0-1-1-0
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 0, AuthData = |
Nah, let's not use an auth mech.
There's nothing in the capabilities!
```

Figure 19

```
time: 0.050023
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvKeepAlive*
connection_state: *OpenConfirm*

n1 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
19-4
Checking if Marker is all 1's
BGP session with peer 10.0.0.1 established.

time: 0.050023
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvKeepAlive*
connection_state: *OpenConfirm*

n0 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
19-4
Checking if Marker is all 1's
BGP session with peer 10.0.1.1 established.
```

Figure 20

```
time: 29.05
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvKeepAlive*
connection_state: *Established*

n0 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
19-4
Checking if Marker is all 1's

time: 29.05
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvKeepAlive*
connection_state: *Established*

n1 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
19-4
Checking if Marker is all 1's
```

B.2 BGP Header and Message Error Checking

B.2.1 Unacceptable Hold Time Value

Simulation starts...

```
time: 0.05
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.05
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.0500096
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

```
n0 creating OPEN MESH w/ length 10
Length of Opt Params = |0|, OPEN Mesg Length = |10|
-> Total BGP Length is now |29|
```

```
time: 0.0500151
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*
```

```
n1 has received:
  Mesg type = 1, length = 29
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
29-1
  OpenMesg: 4-0-0-0-2-10-0-0-1-0
Checking if Marker is all 1's
non-zero Hold Timer value is less than the minimum recommended value 3s
(current_val = 2 s)
BGP session with peer 10.0.0.1 closed.
```

Figure 21

B.2.2 Unexpected Marker Field

Simulation starts...

```
time: 0.05
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.05
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.0500096
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

```
n1 creating OPEN MESH w/ length 10
Length of Opt Params = |0|, OPEN Mesg Length = |10|
-> Total BGP Length is now |29|
```

```
time: 0.0500096
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

```
n0 creating OPEN MESH w/ length 10
Length of Opt Params = |0|, OPEN Mesg Length = |10|
-> Total BGP Length is now |29|
```

```
time: 0.0500151
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *Idle*
```

```
n1 has received:
  Mesg type = 1, length = 29
  BGPHeader: 255-255-255-255-255-255-255-255-255-10-255-255-255-255-0-
29-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-0
Checking if Marker is all 1's
The Marker field is not as expected.
BGP session with peer 10.0.0.1 closed.
```

Figure 22

B.2.3 Bad Path Attribute Flag

```
time: 0.251736
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvUpdate*
connection_state: *Established*
```

<pre>n1 has received: Mesg type = 2, length = 59 BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0- 59-2 UpdateMesg: 0-0-0-32-64-1-4-0-64-2-7-2-1-0-1-192-3-7-10-0-3-1-128-9-7-10-0-6-1- 128-10-7-0-0-3-232-24-10-0-6 Checking if Marker is all 1's i=1: o=0, t=1, p=0 i=2: o=0, t=1, p=0 i=3: o=1, t=1, p=0 -> boooooooooo! >=(UpdateERR 4 w/ Notif: 192 3 7 10 0 3 1 BGP session with peer 10.0.0.1 closed.</pre>	Figure 23
---	-----------

```
time: 0.25328
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvNotification*
connection_state: *Established*
```

```
n0 has received:
  Mesg type = 3, length = 28
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
28-3
  NotifMesg: 3-4-192-3-7-10-0-5-1
Checking if Marker is all 1's
BGP session with peer 10.0.1.1 closed.
```

B.3 OPEN Process Negotiation of Optional Parameters

B.3.1 Scenario 1: Matching Authentication Mechanisms

Simulation starts...

```
time: 0.05
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.05
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.0500096
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*

n1 creating OPEN MESH w/ length 16
Inserting authentication into OPEN message:
  pType |1|, pLength |4|, aCode |2|, aData: 32:1 33:2 34:3
Length of Opt Params = |6|, OPEN Mesg Length = |16|
-> Total BGP Length is now |35|

time: 0.0500096
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*

n0 creating OPEN MESH w/ length 16
Inserting authentication into OPEN message:
  pType |1|, pLength |4|, aCode |2|, aData: 32:1 33:2 34:3
Length of Opt Params = |6|, OPEN Mesg Length = |16|
-> Total BGP Length is now |35|
```

Figure 24

```
time: 0.0500156
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n1 has received:
  Mesg type = 1, length = 35
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-0-
35-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-6
  OptParams: 1-4-2-49-50-51
Checking if Marker is all 1's
Peer wants AuthCode = 2, AuthData = |1|2|3|
I have AuthCode = 2, AuthData = |1|2|3|
We've agreed on an auth mech!
There's nothing in the capabilities!

time: 0.0500156
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*
```

Figure 25

```

n0 has received:
  Mesg type = 1, length = 35
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
35-1
  OpenMesg: 4-0-1-0-90-10-0-1-1-6
  OptParams: 1-4-2-49-50-51
Checking if Marker is all 1's
Peer wants AuthCode = 2, AuthData = |1|2|3|
I have AuthCode = 2, AuthData = |1|2|3|
We've agreed on an auth mech!
There's nothing in the capabilities!

```

```

time: 0.0500235
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvKeepAlive*
connection_state: *OpenConfirm*

```

```

n1 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
19-4
Checking if Marker is all 1's
BGP session with peer 10.0.0.1 established.

```

```

time: 0.0500235
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvKeepAlive*
connection_state: *OpenConfirm*

```

```

n0 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
19-4
Checking if Marker is all 1's
BGP session with peer 10.0.1.1 established.

```

```

time: 29.05
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvKeepAlive*
connection_state: *Established*

```

Figure 26

```

n0 has received:
  Mesg type = 4, length = 19
  BGPHeader: 11-11-11-11-11-11-11-11-11-11-11-11-11-11-0-19-4
Using Another_AuthCode to test the Marker

```

```

time: 29.05
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvKeepAlive*
connection_state: *Established*

```

```

n1 has received:
  Mesg type = 4, length = 19
  BGPHeader: 11-11-11-11-11-11-11-11-11-11-11-11-11-11-0-19-4
Using Another_AuthCode to test the Marker

```

B.3.2 Scenario 2: One-Sided Authentication Mechanism

Simulation starts...

```
time: 0.05
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.05
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.0500096
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

n1 creating OPEN MESSG w/ length 14
Inserting authentication into OPEN message:
 pType |1|, pLength |2|, aCode |2|, aData: 32:L
Length of Opt Params = |4|, OPEN Mesg Length = |14|
-> Total BGP Length is now |33|

```
time: 0.0500096
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

n0 creating OPEN MESSG w/ length 10
Length of Opt Params = |0|, OPEN Mesg Length = |10|
-> Total BGP Length is now |29|

Figure 27

```
time: 0.0500151
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*
```

n1 has received:
 Msg type = 1, length = 29
 BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
29-1
 OpenMesg: 4-0-0-0-90-10-0-0-1-0
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 2, AuthData = |L|
There's nothing in the capabilities!

```
time: 0.0500154
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*
```

n0 has received:
 Msg type = 1, length = 33
 BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
33-1
 OpenMesg: 4-0-1-0-90-10-0-1-1-4
 OptParams: 1-2-2-76
Checking if Marker is all 1's
Peer wants AuthCode = 2, AuthData = |L|
I have AuthCode = 0, AuthData = |
 BGP session with peer 10.0.1.1 closed.
handle_open.auth: push to reconnect

Figure 28

```

time: 0.0500238
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvNotification*
connection_state: *OpenConfirm*

n1 has received:
  Mesg type = 3, length = 24
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
24-3
  NotifMesg: 2-5-1-2-2
Checking if Marker is all 1's
BGP session with peer 10.0.0.1 closed.
OPENCONFIRM: push to reconnect

time: 0.100015
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *BGPstart*
connection_state: *Idle*

time: 0.100024
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *BGPstart*
connection_state: *Idle*

```

<pre> time: 0.10003 peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1 event_type: *TransConnOpen* connection_state: *Connect* n1 creating OPEN MESH w/ length 10 Length of Opt Params = 0 , OPEN Mesg Length = 10 -> Total BGP Length is now 29 time: 0.100033 peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1 event_type: *TransConnOpen* connection_state: *Connect* n0 creating OPEN MESH w/ length 10 Length of Opt Params = 0 , OPEN Mesg Length = 10 -> Total BGP Length is now 29 </pre>	<p>Figure 29</p>
--	-------------------------

```

time: 0.100039
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n1 has received:
  Mesg type = 1, length = 29
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
29-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-0
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 2, AuthData = |L|
There's nothing in the capabilities!

time: 0.100039
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*

```

```

n0 has received:
  Mesg type = 1, length = 29
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
29-1
  OpenMesg: 4-0-1-0-90-10-0-1-1-0
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 0, AuthData = |
Nah, let's not use an auth mech.
There's nothing in the capabilities!

```

<pre> time: 0.100047 peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1 event_type: *RecvKeepAlive* connection_state: *OpenConfirm* n1 has received: Mesg type = 4, length = 19 BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0- 19-4 Checking if Marker is all 1's BGP session with peer 10.0.0.1 established. time: 0.100047 peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1 event_type: *RecvKeepAlive* connection_state: *OpenConfirm* n0 has received: Mesg type = 4, length = 19 BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0- 19-4 Checking if Marker is all 1's BGP session with peer 10.0.1.1 established. time: 29.1 peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1 event_type: *RecvKeepAlive* connection_state: *Established* n0 has received: Mesg type = 4, length = 19 BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0- 19-4 Checking if Marker is all 1's time: 29.1 peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1 event_type: *RecvKeepAlive* connection_state: *Established* n1 has received: Mesg type = 4, length = 19 BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0- 19-4 Checking if Marker is all 1's </pre>	<p>Figure 30</p>
---	-------------------------

B.3.3 Scenario 3: Different Authentication Mechanisms

Simulation starts...

```
time: 0.05
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *BGPstart*
connection_state: *Idle*

time: 0.05
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.0500096
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*

n1 creating OPEN MESG w/ length 14
Inserting authentication into OPEN message:
  pType |1|, pLength |2|, aCode |2|, aData: 32:L
Length of Opt Params = |4|, OPEN Mesg Length = |14|
-> Total BGP Length is now |33|

time: 0.0500096
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*

n0 creating OPEN MESG w/ length 16
Inserting authentication into OPEN message:
  pType |1|, pLength |4|, aCode |2|, aData: 32:1 33:2 34:3
Length of Opt Params = |6|, OPEN Mesg Length = |16|
-> Total BGP Length is now |35|
```

Figure 31

```
time: 0.0500154
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n0 has received:
  Mesg type = 1, length = 33
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
33-1
  OpenMesg: 4-0-1-0-90-10-0-1-1-4
  OptParams: 1-2-2-76
Checking if Marker is all 1's
Peer wants AuthCode = 2, AuthData = |L|
I have AuthCode = 2, AuthData = |1|2|3|
Auth data lengths don't match.
BGP session with peer 10.0.1.1 closed.
handle_open.auth: push to reconnect

time: 0.0500156
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*
```

Figure 32

```

n1 has received:
  Msg type = 1, length = 35
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
35-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-6
  OptParams: 1-4-2-49-50-51
Checking if Marker is all 1's
Peer wants AuthCode = 2, AuthData = |1|2|3|
I have AuthCode = 2, AuthData = |L|
Auth data lengths don't match.
BGP session with peer 10.0.0.1 closed.
handle_open.auth: push to reconnect

```

```

time: 0.100015
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *BGPstart*
connection_state: *Idle*

```

```

time: 0.100016
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *BGPstart*
connection_state: *Idle*

```

```

time: 0.100025
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*

```

```

n1 creating OPEN MESH w/ length 10
Length of Opt Params = |0|, OPEN Mesg Length = |10|
-> Total BGP Length is now |29|

```

```

time: 0.100025
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*

```

```

n0 creating OPEN MESH w/ length 10
Length of Opt Params = |0|, OPEN Mesg Length = |10|
-> Total BGP Length is now |29|

```

```

time: 0.100031
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*

```

```

n1 has received:
  Msg type = 1, length = 29
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
29-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-0
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 2, AuthData = |L|
There's nothing in the capabilities!

```

```

time: 0.100031
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*

```

Figure 33

```

n0 has received:
  Mesg type = 1, length = 29
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
29-1
  OpenMesg: 4-0-1-0-90-10-0-1-1-0
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 2, AuthData = |1|2|3|
There's nothing in the capabilities!

```

<pre> time: 0.100039 peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1 event_type: *RecvKeepAlive* connection_state: *OpenConfirm* n1 has received: Mesg type = 4, length = 19 BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0- 19-4 Checking if Marker is all 1's BGP session with peer 10.0.0.1 established. time: 0.100039 peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1 event_type: *RecvKeepAlive* connection_state: *OpenConfirm* n0 has received: Mesg type = 4, length = 19 BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0- 19-4 Checking if Marker is all 1's BGP session with peer 10.0.1.1 established. </pre>	Figure 34
---	------------------

```

time: 29.1
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvKeepAlive*
connection_state: *Established*

n0 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
19-4
Checking if Marker is all 1's

time: 29.1
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvKeepAlive*
connection_state: *Established*

n1 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
19-4
Checking if Marker is all 1's

```

B.3.4 Scenario 4: Mutually Exclusive Capability Sets

Simulation starts...

```
time: 0.05
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.05
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.0500096
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

n1 creating OPEN MSG w/ length 32
Inserting capability 2->mariners into OPEN message
pType |2|, cCode |2|, cLength |8|
cData: 33:m 34:a 35:r 36:i 37:n 38:e 39:r 40:s -> pLength = |10|
Inserting capability 3->angels into OPEN message
pType |2|, cCode |3|, cLength |6|
cData: 45:a 46:n 47:g 48:e 49:l 50:s -> pLength = |8|
Length of Opt Params = |22|, OPEN Mesg Length = |32|
-> Total BGP Length is now |51|

```
time: 0.0500096
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

n0 creating OPEN MSG w/ length 10
Length of Opt Params = |0|, OPEN Mesg Length = |10|
-> Total BGP Length is now |29|

Figure 35

```
time: 0.0500151
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*
```

n1 has received:
Msg type = 1, length = 29
BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-255-0-
29-1
OpenMesg: 4-0-0-0-90-10-0-0-1-0
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 0, AuthData = |
Nah, let's not use an auth mech.
There's nothing in the capabilities!

```
time: 0.0500169
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*
```

Figure 36

```

n0 has received:
  Mesg type = 1, length = 51
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
51-1
  OpenMesg: 4-0-1-0-90-10-0-1-1-22
  OptParams: 2-10-2-8-109-97-114-105-110-101-114-115-2-8-3-6-97-110-103-101-108-
115
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 0, AuthData = |
Nah, let's not use an auth mech.
2 caps included in the OPEN message: 2->mariners 3->angels
2 caps aren't in my caps: 2->mariners 3->angels
Inserting capability 2->mariners into NOTIF message, starting at nLen=0
  pType |2|, cCode |2|, cLength |8|
  cData: 4:m 5:a 6:r 7:i 8:n 9:e 10:r 11:s -> NotifData length = |10|
Inserting capability 3->angels into NOTIF message, starting at nLen=12
  pType |2|, cCode |3|, cLength |6|
  cData: 16:a 17:n 18:g 19:e 20:l 21:s -> NotifData length = |8|
BGP session with peer 10.0.1.1 closed.
handle_open.caps: push to reconnect

```

```

time: 0.0500267
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvNotification*
connection_state: *OpenConfirm*

```

```

n1 has received:
  Mesg type = 3, length = 43
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
43-3
  NotifMesg: 2-7-2-10-2-8-109-97-114-105-110-101-114-115-2-8-3-6-97-110-103-101-
108-115
Checking if Marker is all 1's
BGP session with peer 10.0.0.1 closed.
OPENCONFIRM: push to reconnect

```

```

time: 0.100017
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *BGPstart*
connection_state: *Idle*

```

```

time: 0.100027
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *BGPstart*
connection_state: *Idle*

```

```

time: 0.100033
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*

n1 creating OPEN MSG w/ length 10
Length of Opt Params = |0|, OPEN Mesg Length = |10|
-> Total BGP Length is now |29|

time: 0.100036
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*

n0 creating OPEN MSG w/ length 10
Length of Opt Params = |0|, OPEN Mesg Length = |10|
-> Total BGP Length is now |29|

```

Figure 37

```

time: 0.100042
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n1 has received:
  Mesg type = 1, length = 29
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
29-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-0
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 0, AuthData = |
Nah, let's not use an auth mech.
There's nothing in the capabilities!

time: 0.100042
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n0 has received:
  Mesg type = 1, length = 29
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
29-1
  OpenMesg: 4-0-1-0-90-10-0-1-1-0
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 0, AuthData = |
Nah, let's not use an auth mech.
There's nothing in the capabilities!

```

<pre> time: 0.100053 peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1 event_type: *RecvKeepAlive* connection_state: *OpenConfirm* n0 has received: Mesg type = 4, length = 19 BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0- 19-4 Checking if Marker is all 1's BGP session with peer 10.0.1.1 established. time: 0.100053 peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1 event_type: *RecvKeepAlive* connection_state: *OpenConfirm* n1 has received: Mesg type = 4, length = 19 BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0- 19-4 Checking if Marker is all 1's BGP session with peer 10.0.0.1 established. </pre>	Figure 38
---	------------------

B.3.5 Scenario 5: Overlapping Capability Sets

Simulation starts...

```
time: 0.05
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.05
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.0500096
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

n1 creating OPEN MSG w/ length 32
Inserting capability 2->mariners into OPEN message
pType |2|, cCode |2|, cLength |8|
cData: 33:m 34:a 35:r 36:i 37:n 38:e 39:r 40:s -> pLength = |10|
Inserting capability 3->angels into OPEN message
pType |2|, cCode |3|, cLength |6|
cData: 45:a 46:n 47:g 48:e 49:l 50:s -> pLength = |8|
Length of Opt Params = |22|, OPEN Mesg Length = |32|
-> Total BGP Length is now |51|

```
time: 0.0500096
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

n0 creating OPEN MSG w/ length 32
Inserting capability 1->redsox into OPEN message
pType |2|, cCode |1|, cLength |6|
cData: 33:r 34:e 35:d 36:s 37:o 38:x -> pLength = |8|
Inserting capability 2->mariners into OPEN message
pType |2|, cCode |2|, cLength |8|
cData: 43:m 44:a 45:r 46:i 47:n 48:e 49:r 50:s -> pLength = |10|
Length of Opt Params = |22|, OPEN Mesg Length = |32|
-> Total BGP Length is now |51|

Figure 39

```
time: 0.0500169
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*
```

Figure 40

```

n1 has received:
  Mesg type = 1, length = 51
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
51-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-22
  OptParams: 2-8-1-6-114-101-100-115-111-120-2-10-2-8-109-97-114-105-110-101-
114-115
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 0, AuthData = |
Nah, let's not use an auth mech.
2 caps included in the OPEN message: 1->redsox 2->mariners
1 caps aren't in my caps: 1->redsox
Inserting capability 1->redsox into NOTIF message, starting at nLen=0
  pType |2|, cCode |1|, cLength |6|
  cData: 4:r 5:e 6:d 7:s 8:o 9:x -> NotifData length = |8|
BGP session with peer 10.0.0.1 closed.
handle_open.caps: push to reconnect

time: 0.0500169
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n0 has received:
  Mesg type = 1, length = 51
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
51-1
  OpenMesg: 4-0-1-0-90-10-0-1-1-22
  OptParams: 2-10-2-8-109-97-114-105-110-101-114-115-2-8-3-6-97-110-103-101-108-
115
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 0, AuthData = |
Nah, let's not use an auth mech.
2 caps included in the OPEN message: 2->mariners 3->angels
1 caps aren't in my caps: 3->angels
Inserting capability 3->angels into NOTIF message, starting at nLen=0
  pType |2|, cCode |3|, cLength |6|
  cData: 4:a 5:n 6:g 7:e 8:l 9:s -> NotifData length = |8|
BGP session with peer 10.0.1.1 closed.
handle_open.caps: push to reconnect

```

```

time: 0.100017
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *BGPstart*
connection_state: *Idle*

```

```

n0 just received ...
time: 0.100017
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *BGPstart*
connection_state: *Idle*

```

```

time: 0.100026
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*

n0 creating OPEN MSG w/ length 22
Inserting capability 2->mariners into OPEN message
  pType |2|, cCode |2|, cLength |8|
  cData: 33:m 34:a 35:r 36:i 37:n 38:e 39:r 40:s -> pLength = |10|
Length of Opt Params = |12|, OPEN Mesg Length = |22|
-> Total BGP Length is now |41|

```

Figure 41

```

time: 0.100026
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*

n1 creating OPEN MESSG w/ length 22
Inserting capability 2->mariners into OPEN message
  pType |2|, cCode |2|, cLength |8|
  cData: 33:m 34:a 35:r 36:i 37:n 38:e 39:r 40:s    -> pLength = |10|
Length of Opt Params = |12|, OPEN Mesg Length = |22|
-> Total BGP Length is now |41|

```

```

time: 0.100033
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n0 has received:
  Mesg type = 1, length = 41
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
41-1
  OpenMesg: 4-0-1-0-90-10-0-1-1-12
  OptParams: 2-10-2-8-109-97-114-105-110-101-114-115
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 0, AuthData = |
Nah, let's not use an auth mech.
1 caps included in the OPEN message: 2->mariners
0 caps aren't in my caps:

time: 0.100033
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n1 has received:
  Mesg type = 1, length = 41
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
41-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-12
  OptParams: 2-10-2-8-109-97-114-105-110-101-114-115
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 0, AuthData = |
Nah, let's not use an auth mech.
1 caps included in the OPEN message: 2->mariners
0 caps aren't in my caps:

time: 0.100041
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvKeepAlive*
connection_state: *OpenConfirm*

n0 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
19-4
Checking if Marker is all 1's
BGP session with peer 10.0.1.1 established.

time: 0.100041
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvKeepAlive*
connection_state: *OpenConfirm*

```

Figure 42

```
n1 has received:  
  Mesg type = 4, length = 19  
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-  
19-4  
Checking if Marker is all 1's  
BGP session with peer 10.0.0.1 established.
```

B.3.6 Scenario 6: Both Optional Parameters – One Negotiation Required

Simulation starts...

```
time: 0.05
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.05
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.0500096
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

n1 creating OPEN MSG w/ length 28
Inserting authentication into OPEN message:
 pType |1|, pLength |4|, aCode |2|, aData: 32:1 33:2 34:3
Inserting capability 2->mariners into OPEN message
 pType |2|, cCode |2|, cLength |8|
 cData: 39:m 40:a 41:r 42:i 43:n 44:e 45:r 46:s -> pLength = |10|
Length of Opt Params = |18|, OPEN Mesg Length = |28|
-> Total BGP Length is now |47|

```
time: 0.0500096
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

n0 creating OPEN MSG w/ length 16
Inserting authentication into OPEN message:
 pType |1|, pLength |4|, aCode |2|, aData: 32:1 33:2 34:3
Length of Opt Params = |6|, OPEN Mesg Length = |16|
-> Total BGP Length is now |35|

Figure 43

```
time: 0.0500156
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*
```

n1 has received:
 Mesg type = 1, length = 35
 BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-0-
35-1
 OpenMesg: 4-0-0-0-90-10-0-0-1-6
 OptParams: 1-4-2-49-50-51
Checking if Marker is all 1's
Peer wants AuthCode = 2, AuthData = |1|2|3|
I have AuthCode = 2, AuthData = |1|2|3|
We've agreed on an auth mech!
There's nothing in the capabilities!

```
time: 0.0500166
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*
```

Figure 44

```

n0 has received:
  Mesg type = 1, length = 47
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
47-1
  OpenMesg: 4-0-1-0-90-10-0-1-1-18
  OptParams: 1-4-2-49-50-51-2-10-2-8-109-97-114-105-110-101-114-115
Checking if Marker is all 1's
Peer wants AuthCode = 2, AuthData = |1|2|3|
I have AuthCode = 2, AuthData = |1|2|3|
We've agreed on an auth mech!
1 caps included in the OPEN message: 2->mariners
1 caps aren't in my caps: 2->mariners
Inserting capability 2->mariners into NOTIF message, starting at nLen=0
  pType |2|, cCode |2|, cLength |8|
  cData: 4:m 5:a 6:r 7:i 8:n 9:e 10:r 11:s -> NotifData length = |10|
BGP session with peer 10.0.1.1 closed.
handle_open.caps: push to reconnect

```

```

time: 0.0500256
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvNotification*
connection_state: *OpenConfirm*

```

```

n1 has received:
  Mesg type = 3, length = 33
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
33-3
  NotifMesg: 2-7-2-10-2-8-109-97-114-105-110-101-114-115
Checking if Marker is all 1's
BGP session with peer 10.0.0.1 closed.
OPENCONFIRM: push to reconnect

```

```

time: 0.100017
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *BGPstart*
connection_state: *Idle*

```

```

time: 0.100026
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *BGPstart*
connection_state: *Idle*

```

```

time: 0.100032
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*

```

```

n1 creating OPEN MESSG w/ length 16
Inserting authentication into OPEN message:
  pType |1|, pLength |4|, aCode |2|, aData: 32:1 33:2 34:3
Length of Opt Params = |6|, OPEN Mesg Length = |16|
-> Total BGP Length is now |35|

```

```

time: 0.100035
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*

```

```

n0 creating OPEN MESSG w/ length 16
Inserting authentication into OPEN message:
  pType |1|, pLength |4|, aCode |2|, aData: 32:1 33:2 34:3
Length of Opt Params = |6|, OPEN Mesg Length = |16|
-> Total BGP Length is now |35|

```

```

time: 0.100041
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*

```

Figure 45

```

n1 has received:
  Msg type = 1, length = 35
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
35-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-6
  OptParams: 1-4-2-49-50-51
Checking if Marker is all 1's
Peer wants AuthCode = 2, AuthData = |1|2|3|
I have AuthCode = 2, AuthData = |1|2|3|
We've agreed on an auth mech!
There's nothing in the capabilities!

time: 0.100041
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n0 has received:
  Msg type = 1, length = 35
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
35-1
  OpenMesg: 4-0-1-0-90-10-0-1-1-6
  OptParams: 1-4-2-49-50-51
Checking if Marker is all 1's
Peer wants AuthCode = 2, AuthData = |1|2|3|
I have AuthCode = 2, AuthData = |1|2|3|
We've agreed on an auth mech!
There's nothing in the capabilities!

time: 0.100049
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvKeepAlive*
connection_state: *OpenConfirm*

n1 has received:
  Msg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
19-4
Checking if Marker is all 1's
BGP session with peer 10.0.0.1 established.

time: 0.100049
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvKeepAlive*
connection_state: *OpenConfirm*

n0 has received:
  Msg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
19-4
Checking if Marker is all 1's
BGP session with peer 10.0.1.1 established.

```

<pre> time: 29.1 peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1 event_type: *RecvKeepAlive* connection_state: *Established* n0 has received: Msg type = 4, length = 19 BGPHeader: 11-11-11-11-11-11-11-11-11-11-11-11-11-11-0-19-4 Using Another_AuthCode to test the Marker time: 29.1 peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1 event_type: *RecvKeepAlive* connection_state: *Established* </pre>	Figure 46
---	------------------

```
n1 has received:  
  Mesg type = 4, length = 19  
  BGPHeader: 11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-0-19-4  
Using Another_AuthCode to test the Marker
```

B.3.7 Scenario 7: Both Optional Parameters – Two Negotiations Required

Simulation starts...

```
time: 0.05
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.05
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *BGPstart*
connection_state: *Idle*
```

```
time: 0.0500096
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

n1 creating OPEN MESH w/ length 26
Inserting authentication into OPEN message:
 pType |1|, pLength |2|, aCode |2|, aData: 32:L
Inserting capability 2->mariners into OPEN message
 pType |2|, cCode |2|, cLength |8|
 cData: 37:m 38:a 39:r 40:i 41:n 42:e 43:r 44:s -> pLength = |10|
Length of Opt Params = |16|, OPEN Mesg Length = |26|
-> Total BGP Length is now |45|

```
time: 0.0500096
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*
```

n0 creating OPEN MESH w/ length 38
Inserting authentication into OPEN message:
 pType |1|, pLength |4|, aCode |2|, aData: 32:1 33:2 34:3
Inserting capability 1->redsox into OPEN message
 pType |2|, cCode |1|, cLength |6|
 cData: 39:r 40:e 41:d 42:s 43:o 44:x -> pLength = |8|
Inserting capability 2->mariners into OPEN message
 pType |2|, cCode |2|, cLength |8|
 cData: 49:m 50:a 51:r 52:i 53:n 54:e 55:r 56:s -> pLength = |10|
Length of Opt Params = |28|, OPEN Mesg Length = |38|
-> Total BGP Length is now |57|

Figure 47

```
time: 0.0500164
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*
```

n0 has received:
 Msg type = 1, length = 45
 BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
45-1
 OpenMesg: 4-0-1-0-90-10-0-1-1-16
 OptParams: 1-2-2-76-2-10-2-8-109-97-114-105-110-101-114-115
Checking if Marker is all 1's
Peer wants AuthCode = 2, AuthData = |L|
I have AuthCode = 2, AuthData = |1|2|3|
Auth data lengths don't match.
BGP session with peer 10.0.1.1 closed.
handle_open.auth: push to reconnect

Figure 48

```

time: 0.0500174
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n1 has received:
  Msg type = 1, length = 57
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
57-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-28
  OptParams: 1-4-2-49-50-51-2-8-1-6-114-101-100-115-111-120-2-10-2-8-109-97-114-
105-110-101-114-115
Checking if Marker is all 1's
Peer wants AuthCode = 2, AuthData = |1|2|3|
I have AuthCode = 2, AuthData = |L|
Auth data lengths don't match.
BGP session with peer 10.0.0.1 closed.
handle_open.auth: push to reconnect

```

```

time: 0.100016
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *BGPstart*
connection_state: *Idle*

```

```

time: 0.100017
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *BGPstart*
connection_state: *Idle*

```

```

time: 0.100027
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*

n1 creating OPEN MSG w/ length 22
Inserting capability 2->mariners into OPEN message
  pType |2|, cCode |2|, cLength |8|
  cData: 33:m 34:a 35:r 36:i 37:n 38:e 39:r 40:s -> pLength = |10|
Length of Opt Params = |12|, OPEN Mesg Length = |22|
-> Total BGP Length is now |41|

time: 0.100027
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*

n0 creating OPEN MSG w/ length 32
Inserting capability 1->redsox into OPEN message
  pType |2|, cCode |1|, cLength |6|
  cData: 33:r 34:e 35:d 36:s 37:o 38:x -> pLength = |8|
Inserting capability 2->mariners into OPEN message
  pType |2|, cCode |2|, cLength |8|
  cData: 43:m 44:a 45:r 46:i 47:n 48:e 49:r 50:s -> pLength = |10|
Length of Opt Params = |22|, OPEN Mesg Length = |32|
-> Total BGP Length is now |51|

```

Figure 49

```

time: 0.100033
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*

```

Figure 50

```

n0 has received:
  Mesg type = 1, length = 41
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
41-1
  OpenMesg: 4-0-1-0-90-10-0-1-1-12
  OptParams: 2-10-2-8-109-97-114-105-110-101-114-115
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 2, AuthData = |1|2|3|
1 caps included in the OPEN message: 2->mariners
0 caps aren't in my caps:

time: 0.100034
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n1 has received:
  Mesg type = 1, length = 51
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
51-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-22
  OptParams: 2-8-1-6-114-101-100-115-111-120-2-10-2-8-109-97-114-105-110-101-
114-115
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 2, AuthData = |L|
2 caps included in the OPEN message: 1->redsox 2->mariners
1 caps aren't in my caps: 1->redsox
Inserting capability 1->redsox into NOTIF message, starting at nLen=0
  pType |2|, cCode |1|, cLength |6|
  cData: 4:r 5:e 6:d 7:s 8:o 9:x -> NotifData length = |8|
BGP session with peer 10.0.0.1 closed.
handle_open.caps: push to reconnect

```

```

time: 0.100043
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvNotification*
connection_state: *OpenConfirm*

```

```

n0 has received:
  Mesg type = 3, length = 31
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
31-3
  NotifMesg: 2-7-2-8-1-6-114-101-100-115-111-120
Checking if Marker is all 1's
BGP session with peer 10.0.1.1 closed.
OPENCONFIRM: push to reconnect

```

```

time: 0.150034
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *BGPstart*
connection_state: *Idle*

```

```

time: 0.150043
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *BGPstart*
connection_state: *Idle*

```

```

time: 0.15005
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *TransConnOpen*
connection_state: *Connect*

```

Figure 51

```

n0 creating OPEN MESH w/ length 22
Inserting capability 2->mariners into OPEN message
  pType |2|, cCode |2|, cLength |8|
  cData: 33:m 34:a 35:r 36:i 37:n 38:e 39:r 40:s    -> pLength = |10|
Length of Opt Params = |12|, OPEN Mesg Length = |22|
-> Total BGP Length is now |41|

time: 0.150053
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *TransConnOpen*
connection_state: *Connect*

n1 creating OPEN MESH w/ length 22
Inserting capability 2->mariners into OPEN message
  pType |2|, cCode |2|, cLength |8|
  cData: 33:m 34:a 35:r 36:i 37:n 38:e 39:r 40:s    -> pLength = |10|
Length of Opt Params = |12|, OPEN Mesg Length = |22|
-> Total BGP Length is now |41|

```

Figure 52

```

time: 0.150059
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n0 has received:
  Msg type = 1, length = 41
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
41-1
  OpenMesg: 4-0-1-0-90-10-0-1-12
  OptParams: 2-10-2-8-109-97-114-105-110-101-114-115
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 2, AuthData = |1|2|3|
1 caps included in the OPEN message: 2->mariners
0 caps aren't in my caps:

time: 0.150059
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvOpen*
connection_state: *OpenSent*

n1 has received:
  Msg type = 1, length = 41
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
41-1
  OpenMesg: 4-0-0-0-90-10-0-0-1-12
  OptParams: 2-10-2-8-109-97-114-105-110-101-114-115
Checking if Marker is all 1's
Peer wants AuthCode = 0, AuthData = |
I have AuthCode = 2, AuthData = |L|
1 caps included in the OPEN message: 2->mariners
0 caps aren't in my caps:

time: 0.150067
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvKeepAlive*
connection_state: *OpenConfirm*

n0 has received:
  Msg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
19-4
Checking if Marker is all 1's
BGP session with peer 10.0.1.1 established.

```

```
time: 0.150067
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvKeepAlive*
connection_state: *OpenConfirm*

n1 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
19-4
Checking if Marker is all 1's
BGP session with peer 10.0.0.1 established.
```

```
time: 29.1501
peer return_ip: 10.0.1.1, peer ip_addr: 10.0.0.1
event_type: *RecvKeepAlive*
connection_state: *Established*

n1 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
19-4
Checking if Marker is all 1's

time: 29.1501
peer return_ip: 10.0.0.1, peer ip_addr: 10.0.1.1
event_type: *RecvKeepAlive*
connection_state: *Established*

n0 has received:
  Mesg type = 4, length = 19
  BGPHeader: 255-255-255-255-255-255-255-255-255-255-255-255-255-0-
19-4
Checking if Marker is all 1's
```

List of References

- [1] Bates, T., et al. "Multiprotocol Extensions for BGP-4," IETF RFC 2858, June 2000.
- [2] "BGP Fundamentals." [webpage] Accessed August 2004. Available from <<http://www.riverstonenet.com/support/bgp/fundamentals/>>. Internet.
- [3] "Border Gateway Protocol." [webpage] Accessed August 2004. Available from <<http://www.iana.org/assignments/bgp-parameters>>. Internet.
- [4] "Capability Codes – Per RFC3392." [webpage]. Accessed August 2004. Available from <<http://www.iana.org/assignments/capability-codes>>. Internet.
- [5] Cisco Systems. "Internetworking Technologies Handbook, Third Edition." [book online]. Cisco Press, 2000. Accessed June 2004. Available from <http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/bgp.pdf>. Internet.
- [6] Chandra, R., and Scudder, J. "Capabilities Advertisement with BGP-4," IETF RFC 2842, May 2000.
- [7] Chandra, R., and Scudder, J. "Capabilities Advertisement with BGP-4," IETF RFC 3392, November 2002.
- [8] Fall, K., and Varadhan, K. "The ns Manual." [webpage]. Accessed June 2004. Available from <http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf>. Internet.
- [9] Feng, T.D. "Implementation of BGP in a Network Simulator." Master's thesis, Simon Fraser University, 2004.
- [10] Feng, T.D., Ballantyne, R., Trajković, Lj. "Implementation of BGP in a Network Simulator." Paper presented at the *Applied Telecommunication Symposium 2004*, Arlington, Virginia, April 2004.
- [11] Greis, M. "Tutorial for the Network Simulator 'ns'." [webpage] Accessed June 2004. Available from <<http://www.isi.edu/nsnam/ns/tutorial/index.html>>. Internet.
- [12] "The Network Simulator – ns-2." [webpage]. Accessed July 2004. Available from <<http://www.isi.edu/nsnam/ns/>>. Internet.
- [13] Pepelnjak, I., and Guichard, J. *MPLS and VPN Architectures*. Indianapolis, IN: Cisco Press, 2001.
- [14] Premore, B.J. "SSF Implementation of BGP-4 v1.5.0." [webpage]. Accessed June 2004. Available from <<http://www.ssfnet.org/bgp/doc/>>. Internet.

- [15] Rekhter, Y., and Li, T. "A Border Gateway Protocol 4 (BGP-4)", IETF RFC 1771, March 1995.
- [16] Reynolds, J., and Postel, J. "Assigned Numbers," IETF RFC 1700, October 1994.
- [17] Rosen, E.C., and Rekhter, Y. "BGP/MPLS IP VPNs," Internet-Draft draft-ietf-13vpn-rfc2547bis-01.txt, September 2003.
- [18] Stewart III, J.W. *BGP4: Inter-Domain Routing in the Internet*. Boston, MA: Addison-Wesley, 1999.
- [19] "Webopedia: Online Computer Dictionary for Computer and Internet Terms and Definitions." [webpage]. Accessed July 2004. Available from <<http://www.webopedia.com/>>. Internet.
- [20] Winston, P.H. *On to C++*. Reading, MA: Addison-Wesley, 1994.