BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

MODELING AND SIMULATION OF EIGRP AND BGP

MODELOVÁNÍ A SIMULACE EIGRP A BGP

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. JAN ZAVŘEL**
AUTOR PRÁCE

SUPERVISOR **Ing. VLADIMÍR VESELÝ, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2022

Brno University of Technology
Faculty of Information Technology

Department of Information Systems (DIFS) Academic year 2021/2022

Master's Thesis Specification

Student: **Zavřel Jan, Bc.**
Programme: Information Technology and Artificial Intelligence
Specialization: Computer Networks
Title: **Modeling and Simulation of EIGRP and BGP**
Category: Networking
Assignment:

1. Analyze EIGRP and BGPv4 routing protocols and study their behavior on Cisco devices.
2. Find out the status of EIGRP and BGP implementation in OMNeT ++ and Cisco.
3. According to the supervisor's recommendation, implement EIGRP and BGP support into the (ANSA) INET framework of the OMNeT ++ environment.
4. Verify the behavior of the implemented simulation models against the corresponding real topology and discuss it.
5. Create tutorials to demonstrate the operation of your models.

Recommended literature:

- Wehrle, K., Mesut, G., & Gross, J., eds. *Modeling and tools for network simulation*. Springer Science & Business Media; 2010.
- Rekhter, Y., Li, T., & Hares, S. *A border gateway protocol 4 (BGP-4)*, RFC 4271; 2005.
- Bates, T., Chandra, R., Katz, D., & Rekhter, Y. *Multiprotocol extensions for BGP-4*, RFC 4760, 2007.
- Savage, D., Ng, J., Moore, S., Slice, D., Paluch, P., & White, R. *Cisco's enhanced interior gateway routing protocol* (EIGRP). RFC 7868; 2016.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Veselý Vladimír, Ing., Ph.D.**
Head of Department: Kolář Dušan, doc. Dr. Ing.
Beginning of work: November 1, 2021
Submission deadline: May 18, 2022
Approval date: October 13, 2021

Abstract

Over the last few decades, the Internet has become one of the most important tools for interpersonal communication. Billions of people use it every day for entertainment, for work, for education, or for the satisfaction of human contact. It must be acknowledged that a huge proportion of the population is existentially dependent on its proper functioning. This is reflected in the ever-increasing demands for higher speeds, lower latency and greater coverage and stability. Network engineers and architects must keep these aspects in mind during the design and deployment processes. One way in which the designed topologies can be tested is through simulation. Simulation uses simulation models that, if they accurately reflect reality, can provide key information about topologies in a risk-free environment at a very affordable cost. This paper deals with the analysis and subsequent improvement of two simulation models of dynamic routing protocols, EIGRP and BGP. These models can be used to create complex simulation topologies and scenarios in the discrete simulator OMNeT++.

Abstrakt

Internet se za posledních několik desítek let stal jedním z nejdůležitějších nástrojů pro mezilidskou komunikaci. Miliardy lidí ho denně používají na zábavu, na práci, na vzdělání či uspokojení lidského kontaktu. Je nutné si připustit, že na jeho správném fungování je obrovská část populace existenčně závislá. Toto se odráží na stále se zvětšujících požadavcích na vyšší rychlost, nižší zpoždění a větší pokrytí a stabilitu. Síťoví inženýři a architekti musí při návrhu a nasazení dbát právě na tyto aspekty. Jedním ze způsobů, kterým je možné navržené topologie otestovat, je simulace. Simulace využívá simulační modely, které, pokud přesně odráží realitu, mohou poskytnout klíčové informace o topologiích v bezpečném prostředí a to za velice přívětivou cenu. Tato práce se zabývá analýzou a následným vylepšením dvou simulačních modelů dynamických směrovacích protokolů EIGRP a BGP. Tyto modely mohou být použity k vytvoření komplexních simulačních topologií a scénářů v diskrétním simulátoru OMNeT++.

Keywords

Dynamic routing protocols, BGP, EIGRP, simulation, OMNeT++, INET

Klíčová slova

Dynamické směrovací protokoly, BGP, EIGRP, simulace, OMNeT++, INET

Reference

ZAVŘEL, Jan. *Modeling and simulation of EIGRP and BGP*. Brno, 2022. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Vladimír Veselý, Ph.D.

Rozšířený abstrakt

Simulace je jeden z populárních způsobů zkoumání síťových protokolů či testování navržených topologií. Simulace využívají simulační modely daných protokolů. Je však klíčové, aby tyto modely byly přesné. A právě tímto tématem se zabývá následující diplomová práce. Hlavním cílem je analyzovat stav simulačních modelů protokolů EIGRP a BGP pro diskrétní simulační prostředí OMNeT++, vylepšit a rozšířit jejich rozsah, otestovat a popsat jejich funkcionalitu a v neposlední řadě tyto modely připravit na sloučení s populární síťovou simulační knihovnou pro OMNeT++ nazývanou INET.

OMNeT++ je open-source simulační knihovna s modulární architekturou, která dovoluje vytvářet jednoduché moduly a ty poté skládat do komplexnějších celků. Chování těchto modulů je definováno jejich implementací v jazyce C++ a jedním z primárních způsobů komunikace mezi jednoduchými nebo složenými moduly je zasílání zpráv. Síťová simulační knihovna INET pak ve zmíněném simulátoru implementuje celou řadu nejpoužívanějších síťových protokolů jako Ethernet, IP, TCP, UDP, či sadu různých aplikačních protokolů. Tyto simulační modely je pak možné využít k modelování libovolného vlastního nadstavbového protokolu. Zatímco podstatně omezená verze simulačního modelu směrovacího protokolu BGP je již zahrnuta v samotné knihovně INET, simulační model EIGRP je implementovaný v knihovně ANSAINET. Prestože je tato knihovna založena na knihovně INET, jedná se o starší verzi a tudíž bylo nutné EIGRP model upravit na nové INET API a zároveň s tím odstranit závislosti na funcionalitách, které byly poskytovány samotnou knihovnou ANSAINET a v INET nemají dvojníka.

Oba zmiňované aplikační protokoly, EIGRP i BGP, se řadí mezi dynamické směrovací protokoly. Tato rodina protokolů slouží k dynamické synchronizaci směrovacích tabulek mezi vícero směrovači. Díky použití směrovacích protokolů se případné změny v topologii dynamicky projevují do obsahů směrovacích tabulek, což výrazně zlehčuje práci síťovým administrátorům a zlepšuje fungování celé sítě jako celku. Přestože se EIGRP a BGP řadí mezi tyto protokoly, v reálném prostředí slouží každý k jinému účelu. Zatímco protokol EIGRP je jeden z protokolů, který zajišťuje směrování uvnitř autonomního systému (AS), kde jsou všechny směrovače typicky spravovány jedinou organizací, BGP protokol se používá jako společný protokol pro propojení různých autonomních systémů. Samotné autonomní systémy jsou typicky pod správou velkých organizací; v rámci České republiky to jsou například O2 Czech Republic, Seznam, VUT, T-Mobile, Nordic Telecom a dalších přibližně 700 organizací.

EIGRP je dynamický směrovací protokol patřící do rodiny „Interior Gateway Protocol“ (IGP), které zajišťují výměnu směrovacích informací v rámci jednoho AS. Narozdíl od jiných populárních protokolů patřících do této rodiny (jako jsou OSPF či IS-IS, které jsou označovány jako typ „link-state“), EIGRP je typu „distance-vector“, čili je spíše podobný směrovacím protokolům jako RIP či IGRP. Směrování protokolů typu „distance-vector“ se často nazývá „routing by rumor“, tedy směrování na základě dohad. Směrovače si individuálně počítají vlastní nejkratší cestu do cílových sítí bez znalosti celé topologie a informaci o nejkratší cestě, co znají, sdílí s ostatními. A právě tento koncept je srdcem EIGRP. To si navíc propůjčuje některé podpůrné koncepty prvně implementované v „link-state“ protokolech jako například proces ustanovení a udržování sousedů. EIGRP je obecně vnímáno jako moderní verze „distance-vector“ protokolů. Jeho velkou nevýhodou je velice omezená podpora mezi jinými výrobci síťových zařízení než je jeho vynálezce *Cisco Systems* a to i přestože bylo EIGRP standardizováno.

Dynamický směrovací protokol BGP patří do rodiny „Exterior Gateway Protocol“ (EGP), které zajišťují výměnu směrovacích informací mezi autonomními systémy. Primárním účelem

je oznámit připojeným autonomním systémům informace o sítích, které jsou dostupné v nebo skrze lokální autonomní systém. BGP je jediný aktuálně používaný protokol s tímto účelem. Vytváří lehkou vrstvu abstrakce, jelikož libovolný AS se ostatním jeví jako entita s jednotným směrovacím plánem, bez ohledu na to, jak je v daném AS přesně implementované směrování. BGP, na rozdíl od například EIGRP, směruje na základě sady atributů a nikoliv vzdálenosti. Nejkratší cest k cíli, tj. s nejmenším počtem AS k cíli, tedy obecně nemusí být nejvhodnější. Tyto atributy jsou oznámovány společně s cílovými sítěmi a slouží hlavně k výběru mezi více cestami do stejné cílové sítě. Použití atributů ovšem závisí na každém AS a jeho směrovací politice a tudíž je na každém z nich, jakou cestu do cílové sítě zvolí a propaguje sousedním AS.

Oba tyto protokoly už jsou v určité formě dostupné v simulátoru OMNeT++. Při analýze EIGRP modelu bylo nalezeno několik drobných problémů, které způsobovaly mírné odchylky ve výsledcích simulace. Jedná se například o chybný výpočet metriky, nekonzistetní stav interní databáze cest, či vyžadované přesné pořadí simulačních signálů. Sloučení s knihovnou INET bylo navíc komplikované faktem, že EIGRP model využíval jednak ANSAINET funkcionality, které nejsou v knihovně INET dostupné, a druhak byl založen na staré verzi samotné knihovny INET, jejíž API se od té doby dočkalo markantních změn. Simulační model protokolu BGP je sice součástí knihovny INET už poměrně dlouho, avšak rozsah tohoto modelu je poměrně omezený. V modelu kompletně chyběla podpora pro protokol IPv6, dále chyběla například funkcionalita oznamování nově nedostupných sítí a model byl celkově neinteraktivní, takže bylo obtížné experimentovat s dynamicky se měnícími topologiemi. Některé tyto aspekty byly řešeny v diplomové práci Ing. Adriana Nováka v roce 2019, avšak ne dokonale. Jeho změny nebyly sloučeny s knihovnou INET a jeho verze byla navržena ke kompletnímu přepsání.

Jako součást této práce bylo opraveno několik nalezených problémů se simulačním modelem protokolu EIGRP, funkcionalita poskytovaná knihovnou ANSAINET byla přesunuta do samotného modelu a využití knihovny INET bylo uzpůsobeno novému API. Takto upravený model byl předložen ke sloučení s knihovnou INET, následně přijat a vydán jako její pevná součást od verze 4.3. Naproti tomuto úspěchu, simulační model BGP vyžadoval daleko rozsáhlejší změny. Konfigurační proces byl mírně upraven, aby více odrážel konfiguraci reálných zařízení. Model byl dále rozšířen o podporu protokolu IPv6 s použitím generických struktur, některé typy zpráv modelu byly přidány či kompletně přepsány, populární typy atributů cest byly taktéž přidány a operační smyčka směrovače, která řeší reakci BGP procesu na vnější i vnitřní události, byla úplně přepsána. V tomto modelu ovšem stále chybí některé prvky, které musí být implementovány, než bude model schopen nahradit aktuálního verzi modelu BGP v knihovně INET.

K demonstraci funkcionality obou modelů byly vytvořeny „směrovací tutoriály“, které popisují a demonstrují základní charakteristiky chování modelovaných protokolů. Chování modelů v těchto „tutoriálech“ bylo ověřeno proti implementaci na Cisco zařízeních. Každý „tutoriál“ je navíc opatřen všemi potřebnými konfiguračními a simulačními soubory k jejich rekonstrukci. Rozšířenou verzi těchto „tutoriálů“ je plánováno publikovat na stránkách knihovny INET, kde mohou uživatelé simulátoru OMNeT++ a knihovny INET proniknout do tajů těchto protokolů a zároveň si s jejich pomocí postavit vlastní testovací topologie a navrhnout vlastní scénáře k experimentování.

Výstupem práce jsou vylepšené simulační modely protokolů EIGRP a BGP a sada „směrovacích tutoriálů“, která demonstruje jejich funkčnost a vlastnosti. EIGRP model byl sloučen s knihovnou INET a je nyní věřejně dostupný k experimentování všemi jejími uživateli. Simulační model protokolu BGP byl přepsán a výsledný model daleko lépe re-

flektuje reálné chování modelovaného protokolu. Protože se i interaktivita modelu značně zlepšila, je nyní možné na modelu testovat nové scénáře, jako pády linek, pády celých směrovačů či celkově změny v topologii. V budoucnu je plánováno zveřejnění „směrovacích tutoriálů“ na stránkách knihovny INET spolu se sloučením takto vylepšeného BGP modelu.

Modeling and simulation of EIGRP and BGP

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Mr. Ing. Vladimír Veselý Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Jan Zavřel
May 18, 2022

Acknowledgements

I would like to thank my supervisor Vladimír for his undeniably significant role in my education and for his cultivation of my love of science in general. I also want to thank my two best friends: František Fiala for his willingness to pilot my WoW character(s) on numerous occasions and for keeping my overall sanity at manageable levels, and Jakub Víšek for being the perfect role model for what a good engineer should look like. Last but not least, I would like to thank my sister Petra for occasionally cutting my hair and Lukáš Fürle for motivating me to work out more.

I would like to share a recipe for a dish I eat almost every day: **Red Lentil Dhal**. Place washed red lentils in a pot, add 2.5 times the weight of the lentils of water, bring to a boil, reduce the heat to medium, and cook for 15 minutes. Meanwhile, add vegan butter to a pan and saute some onions. Add copious amounts of garlic and chilli, add lemon juice, garam masala and season to taste with salt&pepper. When the lentils are cooked, add them to the pan and stir vigorously. Optionally, vegan milk can also be added to create a more creamier texture. Serve hot with a Naan or similar flatbread.

Contents

Chapter 1

Introduction

With the number of people using the Internet quadrupling since 2005 [30], it is now more important than ever to push for high availability and stability. These properties can be partially achieved with optimal route selection by network devices on the path to the desired destination. A combination of static routes with dynamic routing protocols is usually used. Such an approach enables the router to select optimal paths to various destinations even after the topology layout changes while having control over specific cases with static routes. Usage of these protocols allows routers to dynamically exchange routing information and update their routing tables accordingly when necessary.

One way of getting close with the complex routing protocols is through simulation model experimentation. If such a model reliably reflects reality, behaviour patterns during critical scenarios can be observed and analysed within a safe and risk-free environment. The results can be taken into account during the next iteration of the topology design, leading to a more stable and optimal solution.

This thesis deals with the implementation of simulation models of two routing protocols. These simulation models can be used to conduct experiments on topologies in a simulated environment. Individual experiments can involve various events (e.g., link failures, node failures, etc.). All observed effects of these events on the topology can be thoroughly investigated and analysed. It is preferred to simulate such events inside a safe environment rather than on active devices, as it is cheaper, faster, and more convenient. However, this requires that the simulation models to be as accurate as possible.

OMNeT++ is an open-source, discrete, and modular C++ simulation library and framework. Its functionality can be extended with additional frameworks (e.g., INET, ANSAINET) that provide simulation models for a variety of protocols. This thesis deals with the implementation of EIGRP and BGP simulation models within OMNeT++.

EIGRP belongs to the family of Interior Gateway Protocols. These protocols are designed to dynamically distribute routing information throughout an autonomous system (AS) as an alternative to fully static routing. This protocol was designed and developed by Cisco Systems, Inc. and is described in RFC 7868. EIGRP simulation model was originally implemented in the ANSAINET framework.

BGP is an Exterior Gateway Protocol and a widely used standard. It is mostly used to connect ASes that are under different administration. It creates the global routing system of the Internet and links tens of thousands of ASes together. The current version is BGP-4 and is described in RFC 4271. A basic BGP simulation model is already implemented in the INET framework.

The goal of this thesis is to improve, verify, and integrate the EIGRP and BGP simulation models into OMNeT++'s INET framework. The quality of the models is to be ensured by a close comparison of all aspects of the model with the actual implementation found on Cisco devices.

1.1 Structure

- Chapter 2 describes the categorisation of dynamic routing protocols and their types.
- Chapter 3 provides a basic overview of EIGRP. It focuses on general principles and also closely describes message types, the neighbour discovery process, metric calculation, and the used Diffusing Update Algorithm (DUAL).
- Chapter 4 contains a brief overview of BGP. It mostly focuses on general usage, use cases, capabilities and principles. The chapter also includes section on autonomous systems.
- Chapter 5 describes the basic premises of the OMNeT++ simulator and its INET and ANSAINET frameworks. The chapter also contains a description of the actual implementation state of existing EIGRP and BGP simulation models.
- Chapter 6 presents the basic EIGRP and BGP configuration on Cisco devices. This configuration will later be used for verification.
- Chapter 7 describes the author's solution to the previously established model defects and discusses the integration process. While the EIGRP section is quite short, the BGP section relates a complex process of redesigning a significant portion of the model that notably extends its functionality.
- Chapter 8 contains a set of 'routing tutorials' that provide the reader with a detailed image of how these two protocols operate. An extended version these tutorials is planned for a publication on the INET website.
- Chapter 9 concludes the thesis with a summary of the most important take-away points. The resulting simulation models and their respective states are reviewed, and additional plans for the future are stated.

Chapter 2

Routing Protocols

The Internet can be viewed as a collection of disjoint sets of routers, and there are two main types of dynamic routing protocols. The first type, called Interior Gateway Protocols (IGPs), provides routing within a singular set of routers. The other type, called Exterior Gateway Protocols (EGPs), provides routing among different sets. These sets often take form of autonomous systems (AS). These ASes are relatively small components of the Internet, and each is under the control of a single administrative entity, usually a large organisation such as an Internet Service Provider (ISP). Routing protocols from the IGP family, such as EIGRP, OSPF, IS-IS, or RIP, are then often used within these ASes to dynamically maintain the contents of routing tables. Different ASes can internally use different IGP protocols or their combination, or even use them in conjunction with static routes. However, inter-AS connections between neighbouring ASes are operated with an EGP protocol called the Border Gateway Protocol (BGP). The following Section 2.1 further expands on this topic. Different usages are demonstrated in Figure 2.1 and different types of routing protocols are shown in Figure 2.2. Autonomous systems and their relation to BGP is described in Section 4.3. This section itself is based on [10].

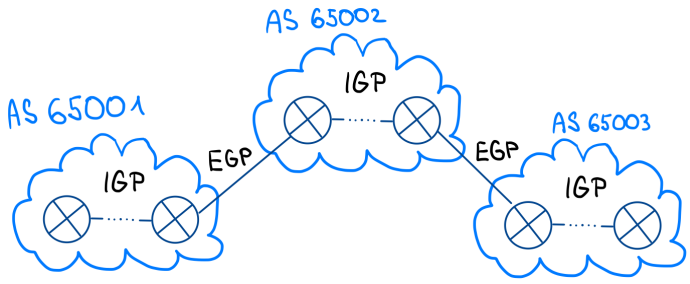


Figure 2.1: Diagram showing different use cases for IGP and EGP protocols.

2.1 Interior Gateway Protocols

The term 'Interior Gateway Protocols' refers to a class of dynamic routing protocols that is used to synchronise routing table entries between routers in the same AS. However, the use of these protocols does not make static routes obsolete, as specific cases can only be handled with static routes. An AS can also use multiple IGP protocols if such usage is favourable. These routing protocols have four main functions:

- discovery of remote networks;
- advertisement of known networks;
- calculation of the best paths;
- recalculation of the best routes following a change in topology.

Favourable characteristics of IGP protocols include low resource usage, fast convergence, good scalability, authentication support, and easy extensibility. There are two main types of IGP protocols: *Distance Vector* and *Link-State*.

2.1.1 Distance Vector

Distance Vector is a family of IGPs that advertises routes as a vector of distance and direction. Distance can vary from a simple hop count to a composite metric with multiple different factors. The direction is the next-hop or exit interface towards the destination network. IGRP, EIGRP, and all versions of RIP belong to the IGP family. Routing decisions are based only on a limited and incomplete knowledge of the full network topology.

IGRP is Cisco's proprietary distance vector routing protocol. It shares many concepts with RIP while making some improvements, especially in regards to scalability. It uses a composite metric and is not limited to 15 hops as opposed to its predecessor. IGRP was later evolved and improved to EIGRP.

Although EIGRP is officially classified as a distance vector, the community often uses terms such as *hybrid* or *advanced distance vector*. This is because it employs some of the same features as link-state protocols, such as neighbour discovery and incremental change-triggered forms of updates. It uses a composite metric that is compatible with IGRP. DUAL is used to find the best route to destination and thus provides loop-less routes even while the topology is not fully converged. It supports IPv4 and IPv6 using Protocol Dependent Modules (PDMs), and its messages use the Type-Length-Value format (TLV), which makes EIGRP more future-proof. EIGRP is further described in Section 3.

2.1.2 Link-State

While the various distance vector protocols advertise route distance, members of the 'Link-State' protocol family instead advertise link states. Routers keep track of its neighbours and generate messages with all the necessary information about directly connected links. The information is distributed throughout the desired area. Each router builds the exact same topology graph based on interchanged information and independently calculates the best routes to all destinations. A topology change only causes the routers to exchange a limited set of incremental updates. IS-IS and OSPF both belong to this protocol family and internally use Dijkstra's algorithm to find the shortest path to destination.

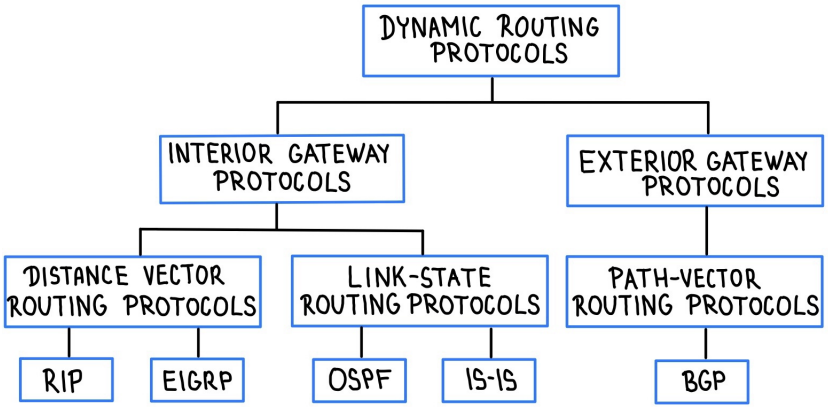


Figure 2.2: Classification of Dynamic Routing Protocols by Cisco [10].

2.2 Exterior Gateway Protocols

The term 'Exterior Gateway Protocols' refers to a class of dynamic routing protocols that is used to exchange routing information between different ASes. A widely used standard is the Border Gateway Protocol (BGP), which is classified as a Path Vector protocol. While numerous attributes influence the best route selection process, the most prominent attribute is the vector of AS numbers on the path to destination (the path vector). Routing decisions are generally based on network policies, which means that the shortest path is not necessarily selected as the best. BGP neighbours are referred to as peers, and there is no dynamic neighbour discovery: IP addresses of the desired peers must be set explicitly and must already be reachable prior to the peering process. The current version is BGP-4 and is described in more detail in Section 4.

Chapter 3

Enhanced Interior Gateway Routing Protocol

EIGRP is an IGP dynamic routing protocol. It is officially classified as distance vector but it is also referred to as being a hybrid between link-state (OSPF, IS-IS) and distance vector (IGRP, RIP) by the community. This is because it uses many features found in link-state protocols, such as non-periodical partial routing updates, and the establishment of neighbourships. Although EIGRP remains a distance vector protocol, these beneficial traits allow it to be viewed as a fairly modern take on distance vector routing. EIGRP was designed and developed by Cisco Systems, Inc. as an improved version of their previous proprietary Interior Gateway Routing Protocol (IGRP), thus 'Enhanced' in the name. EIGRP features classless addressing and support for variable-length network masks (VLSMs) and classless interdomain routing (CIDR). EIGRP calculates the distance using a composite metric that involves up to six different components (more in Section 3.5). It operates directly on top of the network layer from the ISO/OSI model and the protocol's original specification supports IPv4, IPv6, AppleTalk, and IPX by the means of Protocol Dependent Modules (PDMs). To allow for lossless message exchange with no regard for the used PDM, EIGRP employs a unique transport protocol called the Reliable Transport Protocol (RTP). Unlike other distance vector protocols, EIGRP uses stateful neighbourships and incremental routing update messages, reducing convergence time and saving resources. It uses the Diffusing Update Algorithm (DUAL) to calculate loopless shortest routes to destinations. EIGRP supports authentication, equal/unequal load balancing, stub routing to limit unnecessary traffic, and division of topologies into autonomous systems. These ASes are not to be confused with BGP's perspective — from EIGRP's point of view, these ASes represent separate EIGRP routing domains.

EIGRP was submitted as an open standard in 2013 and it is most currently described in informational RFC 7868. This chapter is based on [31, 12, 8, 35, 36, 26].

3.1 Terminology

The following terms are used throughout the chapter:

- Destination - Represents one network known by EIGRP. A single destination can be available via multiple routes.

- Neighbour - Another EIGRP enabled router with an established neighbourship/adjacency.
- Next-hop - A neighbouring router that is one hop closer to the destination.
- Route - A unique combination of destination and next-hop router.
- Reported Distance (RD) - Metric of the path between a next-hop and a destination as advertised by it.
- Computed Distance (CD) - Total metric of the path between the current router and a destination.
- Feasible Distance (FD) - Least known total metric between the current router and the destination.
- Feasibility Condition (FC) - Satisfied if `RD` $<$ `FD`. Ensures loop freedom. All routes that pass `FC` are loop-free; not all routes that fail `FC` contain a loop.
- Feasible Successor - Any neighbouring router that meets the feasibility condition for a given destination.
- Successor - Any feasible successor for a given destination that also provides the least-cost path.
- Active route state - No feasible successor is known, the router is actively attempting to compute the least-cost path. Active routes are not installed into the routing table.
- Passive route state - Feasible successor is available, the route can be installed into the routing table.
- Stuck-In-Active (SIA) - A destination has been in Active state for an excessive amount of time.

3.2 Packets

EIGRP uses unicast and multicast messages. It uses the multicast address `224.0.0.10` for IPv4 and the address `ff02::a` for IPv6. Multicast is used for neighbour discovery and whenever the message is applicable to multiple recipients.

3.2.1 Packet Types

EIGRP defines five different types of packets, however, some of them have more than one use. These are as follows:

HELLO Packet

Hello packets are used to discover neighbours on EIGRP enabled interfaces. They are sent at regular intervals defined by the Hello-Timer. It conveys EIGRP parameters (e.g., K-values, Hold-Timer value) and AS identifier. The K-values determine which components are used to calculate the metric. These have to match between neighbours otherwise neighbourship cannot be established. Hold-timer determines the maximum number of seconds

between consecutive Hello packets. Receiving a Hello packet resets the Hold-Timer. The default value is equal to three times the Hello timer. If the Hold-Timer reaches zero then the neighbour is considered unavailable. Hello packet format is also used for Acknowledgement messages. In any case, Hello packets are not reliably transmitted and do not require acknowledgement.

QUERY Packets

Query packets are used by DUAL to compute the best path to the destination. Routers use this message to advertise a transition of a route to Active state due to it missing a feasible successor. One Query contains one or multiple destinations. After receiving the same number of Reply packets as the number of sent Query packets, the route is transitioned back to Passive state. If the period between the Query and its Reply packet exceeds one half of the SIA time interval, a SIA-Query message is sent. Recipients process SIA-Query messages in the same way as standard Query messages by ultimately responding with a Reply or SIA-Reply, respectively. If no response is sent within one half of the SIA interval, the sender will deem the route as Stuck-In-Active. If a neighbour is considered SIA, the router deletes all routes from that neighbour from the topology table and subsequently from the routing table. Additionally, all routes queried from that specific neighbour for which no response has yet been received are considered unreachable. The router can also reset the adjacency altogether with the SIA neighbour. Query messages are always transmitted reliably and require an Acknowledgement.

REPLY Packets

Reply packets are used as response to a Query or SIA-Query. It informs the recipient about a specific destination's availability and metric. If a SIA-Query is received, the response takes form of a SIA-Reply. This reply can specify a metric for the destination. Otherwise, a set ACTIVE flag indicates that the responding router is in the Active state for the destination (most likely waiting for a Reply/SIA-Reply from another downstream router). Reply packets are always reliably transmitted and always acknowledged.

UPDATE Packets

Update packets advertise available destinations. They are exchanged when a new neighbour is discovered and when a route changes its state from Active to Passive. These messages are sent as unicast when establishing a neighbourship to determine unicast packet delivery capabilities and as multicast in normal operation. The first pair of Update packets between new neighbours is called NULL UPDATE, is always empty and contains a set INIT flag. Update packets are always transmitted reliably and require an Acknowledgement.

REQUEST Packets

Request packets are used to request specific information from one or more neighbours. Request packets are used in route server applications. They use unicast and multicast addresses and are transmitted unreliably.

3.3 Neighbour Discovery

When EIGRP is enabled on an interface, it starts periodically sending Hello packets on a multicast address. These packets carry the router's settings, and it also announces the router's presence on a given link. Neighbour discovery mechanism is analogous to that of OSPF or IS-IS. If two routers are to be considered neighbours, the following conditions must be met.

- Interfaces have to be EIGRP enabled and not passive.
- IPv4 addresses on the interfaces must be in the same network if IPv4 is used. In an IPv6 network, both routers must use valid link-local addresses.
- Routers have to use the same authentication configuration.
- K-Values have to match.
- Routers have to be in the same AS.

When routers become neighbours, they can start exchanging routes via Update packets. Information about EIGRP neighbours is stored in `Neighbor Table`.

3.3.1 Establishing Neighborship

A neighborship between the routers *R1* and *R2* can be in two states from *R1's* perspective.

1. **Pending** - Router *R1* has received a Hello Packet from router *R2*. Router *R1* sends INIT Update Packet and awaits an acknowledgement. While router *R1* has *R2* as pending, it will not send it any Query Packets or other Update Packets until INIT Update is acknowledged.
2. **Up** - Router *R1* has received an acknowledgement for its INIT Update Packet.

When EIGRP is enabled on an interface, the router starts periodically sending Hello Packets. The length of the period is defined by the Hello-Timer. If it receives a foreign Hello packet, it creates a neighbour table entry with the state `PENDING` and also answers with its own Hello packet. To ensure that multicast and unicast communication is possible, it sends an empty unicast Update packet with the `INIT` flag. If a foreign Update packet with the INIT flag is received, it is acknowledged. If an acknowledgement is received, the neighbour's state changes to `UP`, and the initial route exchange begins. The final Update packet is marked as `EOT`. The information acquired during the initial exchange is propagated to the `Topology Table`, and routes with the lowest metric are installed to the routing table.

The process of establishing a neighbourship is shown in the diagram in Appendix B.

3.4 Diffusing Update Algorithm

DUAL is a mechanism for handling topology changes. This section uses terminology from Section 3.1 and provides a brief overview of DUAL. It is an algorithm responsible for handling **events** such as:

- Expiration of EIGRP neighbour
- Change of directly connected interface metric or state

- Receipt of EIGRP Query, Reply or Update message

Actions of DUAL include the following:

- Changes between Passive and Active route states
- Installation or removal of routes within the Topology Table
- Changes to a route's metric or successors

DUAL manages each route individually. If DUAL receives a local event (not triggered by an incoming message) for a passive route, a local computation is performed. If another feasible successor is known, the route's metric and/or successor may change, but the route remains in the Passive state. If no feasible successor is available, local computation is not possible, and thus the route transfers to the Active state. Changes of a route's feasible distance are always propagated to all neighbours via an Update message.

When a route transfers to the Active state, diffusing computation begins by sending Query messages to EIGRP neighbours, except for the originating source, and ends with the receipt of all expected Reply messages. DUAL uses a Finite State Machine (FSM) to track state of routes' diffusing computation. If a route in the Passive state receives a Query message and a feasible successor for a given route is available, the route remains in the Passive state while a Reply is generated. If there is no feasible successor, the router starts its diffusing computation. Once the computation ends, a Reply message is generated if the route's computation was originally triggered by a Query message.

3.4.1 Models

They are multiple ways of modeling DUAL's behaviour. RFC 7868 [31] itself in Section 3.5 uses a FSM which is easy to implement for a programmer, but could be harder to follow for a reader. For this reason, Figure 3.1 presents a flow diagram as an alternative, which visualizes decision paths taken in regards to a destination network, instead of modeling the underlying state machine.

3.4.2 Stub Routers

Stub routing is a way of limiting the scope of DUAL computation. Routers that are configured as Stub advertise this status with special TLV within their Hello messages. The most restricted mode of operation (Receive-only) can be summarized into the following rules:

1. Stub router neighbours exclude it from DUAL computation by not sending it any Query Messages.
2. The stub router answers all received Query messages as if the queried destination was unavailable.
3. The stub router does not advertise any networks.

However, the behaviour of a stub router and its neighbours can be adjusted by configuring the stub router with a valid combination of the following attributes:

- **Receive-only** - The most restricted mode of operation.

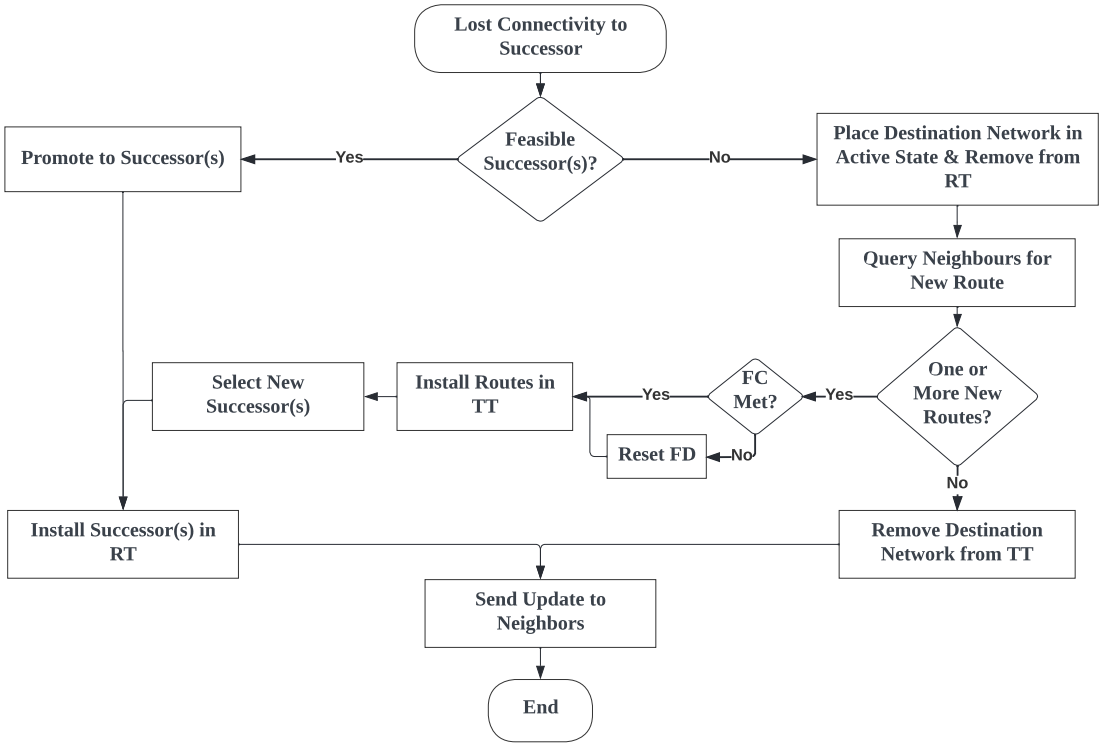


Figure 3.1: Flowchart modeling DUAL.

- **Connected** - The aforementioned rule 3 is not applied when dealing with the stub router's directly connected networks.
- **Static** - The aforementioned rule 3 is not applied when dealing with the stub router's statically configured routes.
- **Leak-map** - The aforementioned rule 3 is not applied when dealing with a specified set of networks/routes.
- **Redistributed** - The aforementioned rule 3 is not applied when dealing with routes redistributed into the stub router's EIGRP process.
- **Summary** - The aforementioned rule 3 is not applied when dealing with summarized routes present in the stub router's EIGRP process.

Depending on the version of IOS, stub routes may or may not generate Query messages.

3.5 Metric Calculation

EIGRP uses a composite metric to evaluate the distance of known routes. It employs up to six components for its metric calculation. These components are called K-Values or coefficients. Each coefficient has a value in the range 0 - 254. The metric calculation can be customized by manipulation of these values. To ensure loopless routes, all EIGRP active devices in the same EIGRP domain have to share the same K-Values. This is ensured by advertising them in Hello Packets. EIGRP supports two modes of metric calculation:

- **Classic Metric** - Metric calculation can include bandwidth, delay, load and reliability. Calculation is based on IGRP metric. To extend IGRP's 24bits to 32 bits, bandwidth and delay are multiplied by 256.
- **Wide Metric** - Metric calculation designed with higher bandwidth networks in mind. Calculation can include the same components as Classic Metric but can also account for jitter and energy. Wide metric uses 64-bit values and only works with `Named Configuration` on Cisco devices.

K-Value	Default Value	Description
K1	1	**Bandwidth** - Minimal interface bandwidth on the path
K2	0	**Load** - Load of directly connected link
K3	1	**Delay** - Sum of link-type based constants on the path
K4, K5	0	**Reliability** - Reliability of directly connected link
K6	0	**Jitter** - Sum of jitter values on the path **Energy** - Sum of milliwatts per kilobit on the path

Table 3.1: EIGRP K-Values.

3.5.1 Classic Metric

Cisco's EIGRP implementation employs classic metric calculation when classic EIGRP configuration is used. Figure 3.1 shows the formula for Classic Metric calculation with non-zero K_5 value.

$$metric = 256 * (K_1 * BW + \frac{K_2 * BW}{256 - LOAD} + (K_3 * DLY) * (\frac{K_5}{REL + K_4})) \tag{3.1}$$

The variables' meaning is as follows:

- BW is scaled **bandwidth**. It is calculated from the lowest bandwidth on any link to the destination. Value of the minimum bandwidth $BwMin$ in $kbps$ is scaled by a factor of 10^7.
$$BW = \frac{(10^7)}{BwMin} \tag{3.2}$$
- DLY is scaled **delay**. It is the sum of link delays along the path to the destination. Delay on each link is defined by the interface type. Table of these constants can be found in RFC 7868 in section 5.6.1.2.
$$DLY = \frac{DlySum}{10} \tag{3.3}$$
- $LOAD$ and REL are values in range 1 - 255. They represent a percentage of **load** and **reliability**. These values are not measured dynamically and are only measured when a route is first learned. A value of 255 would represent a fully loaded link or a 100% reliable link. If K_5 is set to `0`, then the last fraction in equation 3.1 is set to `1`, resulting in the equation 3.4.
$$metric = 256 * (K_1 * BW + \frac{K_2 * BW}{256 - LOAD} + K_3 * DLY) \tag{3.4}$$

3.5.2 Wide Metric

Classic Metric is not properly scaled for today's high-speed interfaces and could cause them to be load balanced in equal-cost manner even when that would not be appropriate. Wide Metric was introduced to address this issue, allowing the metric calculation to scale up to an interface bandwidth of `4.2 Tb/s`. Although Wide Metric calculation uses 64-bit arithmetic, the resultant value may be scaled down to 32 bits if necessary, e.g. to a Classic Metric neighbour. Cisco's EIGRP implementation only supports Wide Metric usage in conjunction with Named Mode Configuration, utilizing the following Formula 3.5 and respective Wide Metric conversion constants:

$$metric = (K_1 * THR + \frac{(K_2 * THR)}{256 - LOAD} + K_3 * LAT + K_6 * EXT) * \frac{K_5}{(K_4 + REL)} \tag{3.5}$$

where:

- THR is the minimum **throughput**. Similarly to the Classic Metric, these values are calculated from the minimum interface bandwidth BW on the path to the destination.

$$THR = \frac{(10^7 * 65536)}{BW} \tag{3.6}$$

- LAT is the total **latency**. On routes which use low-bandwidth interfaces, calculation of this value is based on the sum of interface delays on the path to the destination as in the Classic Metric. LAT value is calculated differently on routes with higher minimum bandwidth interfaces. Equation 3.7 is used if BW is less than `1 Gb/s`, while Equation 3.8 is used otherwise:

$$LAT = \frac{(DLY * 65536)}{10} \tag{3.7}$$

$$LAT = \frac{(10^7 * 65536/10)}{BW} \tag{3.8}$$

- EXT represents **Extended Attributes**. This metric is currently not used, and reserved for future extensions. Although there currently are two attributes — **Jitter** and **Energy** — Cisco's EIGRP does not possess the ability to measure these metric values.

- $LOAD$ and REL retain the same meaning as with the Classic Metric.

Chapter 4

Border Gateway Protocol

Border Gateway Protocol (BGP) is an Exterior Gateway Protocol primarily designed for the exchange of network reachability information between different autonomous systems of the Internet. It is the only widely used EGP protocol at the time of writing.

BGP belongs to the family of Path-Vector protocols as it maintains a vector of autonomous systems that have to be traversed in order to reach a destination. The path-vector also allows for loop detection and hop counting. Unlike IGP, BGP does not use dynamic neighbour discovery. Each neighbour, referred to as *peer* in BGP terminology, must be explicitly configured. The two peering parties typically first agree on what information do they plan on sharing with each other. This could entail a definition of destination networks or a set of expected attributes for exchanged routes: in BGP, a route consists of a destination network prefix and a collection of attributes, most of which are not mandatory. There is no single centralized authority dictating which non-mandatory route attributes have to be specified. Instead, this decision is made according to respective policies of individual peers or peering relationships.

To ensure reliable transport channel for its messages, BGP leverages TCP and has peers listen for connections on well-known port 179.

The most current version is BGP-4 as defined in RFC 4271. BGP can be extended with multiprotocol support (MBGP or MP-BGP). MBGP is also used for services such as MPLS-based virtual private networks (VPNs). MBGP is defined in RFC 4760. The following chapter is based on [25, 14, 13, 29, 26].

4.1 Messages

All BGP messages are sent in unicast mode over TCP connections established between configured BGP peers.

OPEN

Open message type is used to establish a BGP adjacency between two routers. The message contains mandatory fields like BGP Version, Hold-Timer, BGP Identifier (Router-id), and AS number. The advertised AS number allows the routers to determine the type of peering that is being established: either internal (iBGP) or external (eBGP). Peers can choose to include additional Optional parameters to advertise their extended capabilities, such as a 64-bit router-ID, 32-bit AS identifier or support for various address families. These Optional parameters use the Type-Length-Value (TLV) format to allow for better extensibility.

UPDATE

Update messages are used to advertise or withdraw routes. When advertising routes, the message consists of one or more *Network Layer Reachability Information* (NLRI) and of applicable Path Attributes. NLRI's identify destination networks while Path Attributes describe the properties that apply to them. This makes it possible to advertise multiple routes with identical attributes within a single message.

Routes intended for withdrawal are indicated in a special Withdrawn Routes field of the UPDATE message. It can be done in conjunction with route advertisement as long as no prefix is both advertised and withdrawn within a single message.

The format of the message is slightly different when dealing with additional address families other than IPv4 as the message format, lacking TLV support, is not forwards-compatible.

NOTIFICATION

Notification messages inform the peer about fatal errors that result in the termination of both the BGP session and of the underlying TCP connection. The message always specifies the root cause, such as Message Header error, Open or Update message error, FSM error, or Hold Timer Expiration error.

KEEPALIVE

Keep-alive messages are used to ensure that a BGP peer is still available and that the respective TCP connection is still functional. The messages are sent based on an interval. Receipt of a keep-alive message refreshes the Hold Timer, while its expiration causes the BGP session to terminate with the Hold Timer Expiration error. The KEEPALIVE message is also used to acknowledge a received OPEN message.

4.2 Attributes

As already mentioned in the section's introduction, BGP attributes hold information about advertised prefixes. They are classified into 4 categories, each of which imposes some handling and processing requirements:

- **Well-Known Mandatory** - Attributes that must be included with every advertised prefix and recognised by all implementations of BGP.
  - `ORIGIN` - Defines how the originating BGP router learned about a given prefix. Possible values are IGP, EGP, or Incomplete (typically via route redistribution).
  - `AS_PATH` - Defines an ordered or unordered sequence of identifiers of ASes that the update message has traversed.
  - `NEXT_HOP` - Defines the unicast IPv4 address of the router that should be used as a next-hop.
- **Well-Known Discretionary** - Attributes that may optionally be included in a route entry and must be recognised by all implementations of BGP.
  - `LOCAL_PREF` - Defines the preference of the received route to internal peers.

  - `ATOMIC_AGGREGATE` - Defines the usage of a less specific route to the destination.

- **Optional Transitive** - Attributes that do not have to be recognised by all BGP implementations, but must be passed along to other neighbours even if not understood.
  - `AGGREGATOR` - Defines the last AS number that formed the aggregate route together with the IP address of the specific router.

- **Optional Non-Transitive** - Attributes do not have to be recognised by all implementations of BGP. The attribute may only be passed along to other neighbours when understood.
  - `MED` - Defines the priority of the entry point for external peers.
  - `MP_REACH_NLRI` - Advertises feasible routes of to a peer. Its compatible with multiple network layers and address families. However, IPv4 are often carried in separate field.
  - `MP_UNREACH_NLRI` - Withdraws multiple unfeasible routes from service. Its compatible with multiple network layers and address families and again, it not generally used for IPv4.

Different implementations of BGP often support some additional proprietary attributes, such as `WEIGHT` in Cisco devices. The behaviour of attributes `AS_PATH, NEXT_HOP, LOCAL_PREF` and `MED` is demonstrated in Section 8.3.4 of the Testing chapter.

4.2.1 Decision process

While some attributes provide key information that makes the route usable, all attributes serve as means to compare routes to the same destination between each other. Although BGP implementations can adhere to the decision process, as stated in the RFC, the most popular implementation makes slight adjustments to accommodate for a wider variety of scenarios. The most important steps of the decision process are shown in Figure 4.1. A very detailed description of the BGP decision process on Cisco devices can be found on their website [11] and the same goes for Juniper Networks [23].

4.3 Autonomous Systems

The Internet is divided into smaller, more easily manageable networks called autonomous systems. An AS is essentially a collection or a group of IP prefixes owned by a single organisation. AS creates a form of abstraction, as from the other AS's view, an AS provides a consistent picture of what prefixes are available through it, no matter how precisely it is accomplished inside the AS itself. These smaller parts are often managed by different administrative entities and adhere to a custom set of rules and policies. Routing inside an AS is usually managed by IGP or static routing, while connectivity between ASes is typically achieved with BGP. Every public AS is identified by a unique public AS number, however, one public AS can internally consist of multiple private ASes. These are only used for internal subdivisions and are not advertised via BGP. Public AS numbers are assigned by the respective Regional Internet Registries, e.g., RIPE NCC for Europe.

There are three main approaches of how an autonomous system may be connected to other(s):

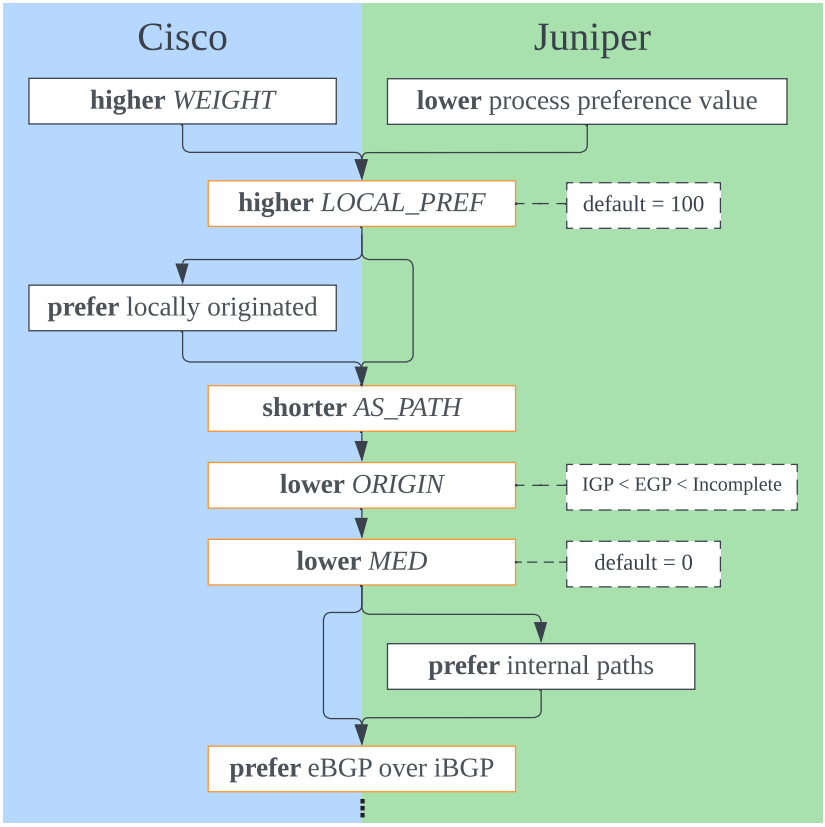


Figure 4.1: Diagram depicting the primary criteria for BGP decision process as implemented by two prominent networking products vendors. Routes are compared from top downwards. Steps standardised in RFC 4271 are outlined in orange. AIGP attribute is disregarded. Information about are sourced from their respective websites [11, 23].

- **Single-Homed AS** (a.k.a. Stub) - The AS is connected to a single external AS that provides it connectivity to the Internet. Single-Homed ASes can use both private and public AS numbers. This particular scenario does not necessarily warrant the use of BGP: the ISP AS typically knows what prefixes are available in the single-homed AS, the use of any dynamic routing protocol provides some benefits but is not necessary.

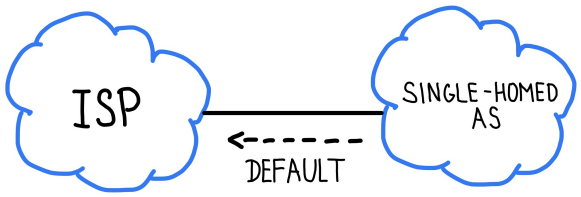


Figure 4.2: Diagram of a single-homed AS connection to the ISP. All incoming and outgoing traffic is going through the ISP.

- **Multi-Homed Non-Transit AS** - AS has multiple Internet gateways. The connection is provided by multiple ISPs. However, this type of AS does not allow *transit*

traffic: traffic that has a source and a destination outside the AS. Only prefixes local to the AS are advertised. BGP is also not required in this case.

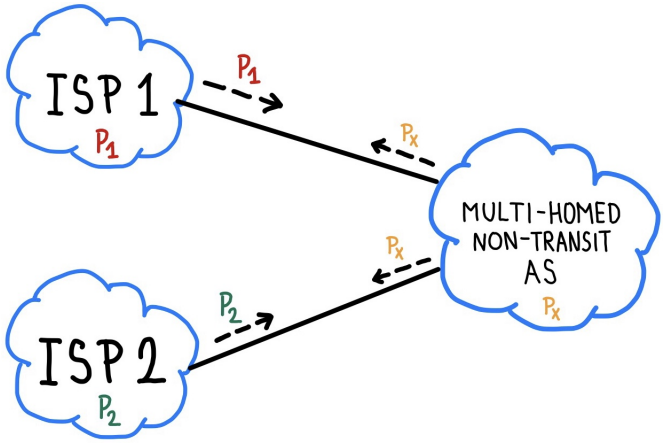


Figure 4.3: Diagram of a multi-homed non-transit AS connection to ISPs. ISPs advertise prefixes P_1 and P_2. Non-Transit AS may advertise its own prefixes P_x if it chooses to.

- **Multi-Homed Transit AS** - AS has multiple gateways to the Internet, while allowing for transit traffic from/to different ASes. BGP is required in this case. **eBGP** is used to advertise known prefixes to other ASes, while **iBGP** is used between routers inside the AS by the transit routers.

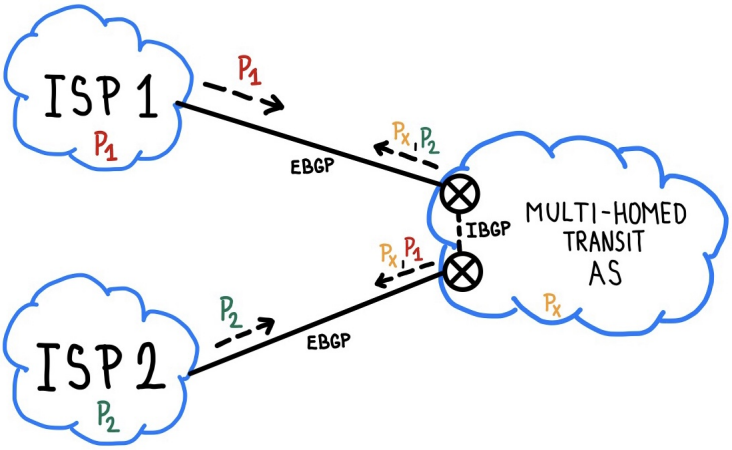


Figure 4.4: Diagram of a multi-homed transit AS connection to ISPs. ISPs advertise prefixes P_1 and P_2, while non-transit AS advertises its own prefixes P_x and also prefixes P_1 learnt from `ISP1` to `ISP2` and vice-versa.

In 2006, it was estimated that around 20-30% of all ASes are transit [37]. The total number of ASes worldwide is currently 109,966 [24]. Tools like DB-IP[1] can be used to access publicly available pieces of information about ASes, their registered organizations, and IP prefixes.

[1] https://db-ip.com/

4.4 BGP Peering

BGP recognises two different types of peering (i.e., prefix exchanging): External BGP (eBGP) and Internal BGP (iBGP). These peering types are each used for a slightly different purpose and come with their own specifics and restrictions. Although this thesis adheres to the naming conventions of iBGP and eBGP, the same as Cisco or RFC 8654 [9], Juniper documentation and other RFCs often use the naming convention of EBGP and IBGP.

4.4.1 External BGP

External BGP peering is formed between neighbours that belong to a different AS. The main purpose of eBGP is to advertise to the neighbouring AS, what prefixes are available in the local AS or available through the local AS (if the local AS allows transit traffic).

eBGP peering may often be established between directly connected (L3) BGP-enabled routers. A situation where peering routers are more than a single hop away from each other is referred to as *multihop eBGP* and it may require raising the BGP session's TTL value in the IP header on certain implementations. Additionally, *multihop eBGP* is unavoidable in certain scenarios, such as if eBGP session is sourced from a loopback interface. Bacause this behaviour is more commonly associated with iBGP, it is described in the following iBGP section.

As for attribute handling, eBGP typically overwrites the `NEXT_HOP` attribute to the local peering source. This enables the other peer to direct traffic to/through the local AS. Similarly, the `AS_PATH` value is prepended with the local AS identifier, to reflect with other BGP peers the fact that the traffic would flow through the local AS. Depending on the local configuration, the local AS identifier may be even prepended multiple times, resulting in a larger overall length of the `AS_PATH` value. Such technique is used to influence BGP decision process of the neighbour as the overall length of `AS_PATH` is one of its factors and shorter paths are preferable. eBGP does not advertise the `LOCAL_PREF` attribute as it does not have any significance to foreign autonomous systems. However, suggestions regarding locally preferred ingress point(s), with respect to advertised prefixes, may be conveyed using the `MED` attribute — which, by contrast, has no local significance.

4.4.2 Internal BGP

Especially in transit ASes that often run BGP on more than one border router, it becomes necessary to synchronise their routing tables to ensure proper function: the one border router that receives transit traffic must know where to forward it through the AS in order to really honour the property of transitivity. In multi-homed scenarios, internal routers need to choose an egress point for outgoing external traffic, ideally the most optimal one. While both aforementioned cases can be achieved using IGPs with various degrees of difficulty and suboptimality, iBGP comes with many benefits that regular IGPs cannot match, such as scalability or policy-based routing.

iBGP peering is not concerned with neighbours' direct connectivity, any kind of L3 connectivity suffices, regardless of the underlying type of routing. However, iBGP peers require full-mesh connectivity, where every iBGP speaker must be peering with each other. This would entail a total of $\frac{N(N-1)}{2}$ peering relationships for N running BGP speakers, including the need to maintain peering configurations on N routers, each specifying $N-1$ neighbour addresses. Such a connection model is required since iBGP peers do not propagate updates

received through iBGP to other iBGP peers. As such, a lack of full-mesh connectivity could result in inconsistent routing tables across the AS.

In practise, various techniques exist to circumvent the need for full-mesh, such as *route reflectors* or *confederations*. For example, BGP route reflectors work as centralised BGP speakers that all others peer with, reducing the number of specified neighbour addresses on each of the N routers to a constant. To make this work, route reflectors slightly adjust the BGP standard when it comes to iBGP behaviour, e.g., disregarding the ban on iBGP-iBGP propagation of iBGP updates mentioned earlier. BGP Route Reflection is standardised in RFC 4456 [7].

When configuring iBGP peering relationships, it is common practise to use addresses associated with loopback interfaces of the two routers, rather than addresses associated with their network interfaces. In the event of one or more links failing or flapping, this provides for additional reliability, maintaining the BGP session as long as the peers are still reachable through some other path.

Generally, iBGP does not alter route attributes such as `NEXT_HOP` or `AS_PATH`. This means that iBGP routes implicitly specify `NEXT_HOP` address of the foreign eBGP border router, although any router that the update has passed through might have overwritten the attribute with its own address. In comparison to eBGP, iBGP propagates the `MED` attribute (received via eBGP) and may also add the `LOCAL_PREF` attribute to influence the selection of egress point.

4.4.3 BGP Peer Finite State Machine

The purpose of BGP Peer finite state machine (FSM) is to track the state of the peering session for peers. The exact definition of this FSM differs in the popular literature, as even the RFC 4217 [29], Section 8 states the following: 'The data structures and FSM described in this document are conceptual and do not have to be implemented precisely as described here...'. Most often, the complexity of the FSM in various literatures is reduced compared to the version in the RFC. This complexity may be caused by usage of TCP in combination with peer-to-peer model. Because of the nature of TCP, one entity has to perform a 'connect' operation while the other performs a 'listen' operation in order to establish a TCP connection from one side. However, thanks to the usage of peer-to-peer model, these entities are coequal, and thus both entities have to perform both of these TCP operations. And as that results in two separate TCP sockets, one needs to be always closed after the connection is established. These sockets are reffered to as *passive* and *active*. Each TCP connection has only one *active* end and one *passive* end. The *passive* socket always listens on port 179, while the *active* socket uses ephemeral ports. Having two individual sockets for each peer also means two instances of BGP peer FSM, and paired with the its large scope as defined in the RFC, it becomes quite a complex problem to determine the state of peering as a whole. The simplification usually ommits the existance of two BGP peer FSM instances, avoids the existance of the two sockets and generally streamlines the transitions. As a result, the whole process can be unnecessarily confusing to anyone unaware of these facts, if one uses any of the simplified versions to implemented or perfectly understand this whole process.

On Cisco devices, unless explicitly configured previously, it is not predetermined if peer will end up with *passive* or *active* socket from author's observations. However, it still uses two instances of BGP peer FSM.

One of the simplified version of the BGP peer FSM is shown in Figure 4.5.

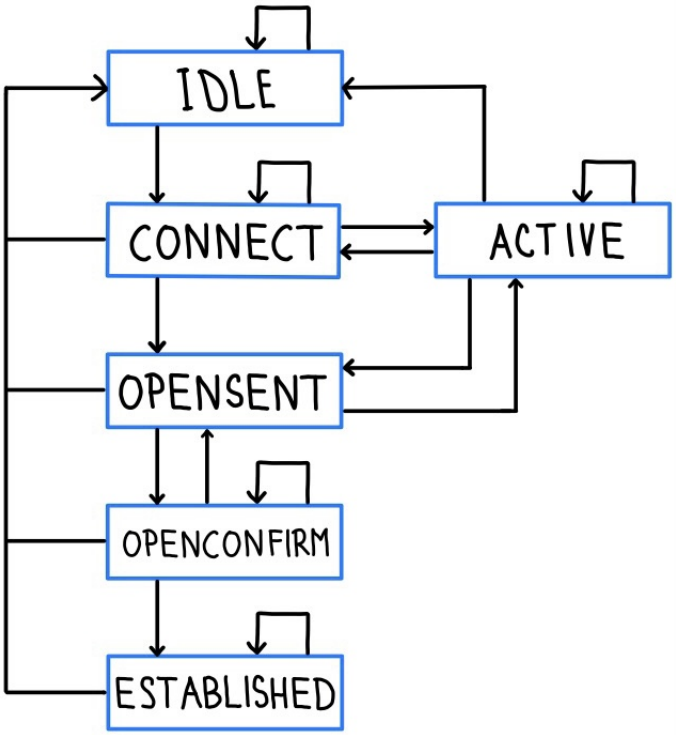


Figure 4.5: BGP neighbor state FSM.

- **IDLE** - Initial state. When BGP is enabled, it initiates a TCP connection and also starts listening for incoming connections from the remote peer. If any error occurs, the BGP returns to the *IDLE* stage. It waits *ConnectRetry Timer* before trying again.
- **CONNECT** - BGP process waiting for the TCP connection to be established. If it is successful, it sends an `OPEN` message, refreshes *ConnectRetry Timer*, and changes state to the *OPENSENT* state. If it is not successful until *ConnectRetry Timer* expires, it changes to the *ACTIVE* state. In other cases, it returns to the *IDLE* state.
- **ACTIVE** - BGP is trying to establish a TCP connection with a neighbour. If successful, it sends an `OPEN` message, refreshes *ConnectRetry Timer*, and changes state to *OPENSENT* state. If it is not successful, it returns to the state *CONNECT* and refreshes *ConnectRetry Timer*. The oscillation between the states *CONNECT* and *ACTIVE* indicates an error in the TCP connection.
- **OPENSENT** - BGP is waiting for an `OPEN` message from its peer. If the TCP connection is closed before that happens, the BGP will return to the *ACTIVE* state. When an `OPEN` message is received, its content is checked. If this check fails, the BGP sends a `NOTIFICATION` message and returns to the state *IDLE*. If the check passes, the BGP sends a `KEEPALIVE` message and transfers to the *OPENCONFIRM* state.
- **OPENCONFIRM** - BGP is waiting for the peer's `NOTIFICATION` or `KEEPALIVE` message. If `KEEPALIVE` is received, the BGP is transferred to the state *ESTABLISHED*. If the *HOLD TIMER* expires or the `NOTIFICATION` is received, the BGP returns to the *IDLE* state.

- **ESTABLISHED** - BGP starts to exchange `UPDATE` messages with its peer. *HOLD TIMER* is refreshed with each reception of a `UPDATE` or `KEEPALIVE` message. If a faulty `UPDATE` or `NOTIFICATION` message is received or the *HOLD TIMER* expires, the BGP is transferred back to the *IDLE* state.

Chapter 5

OMNeT++

OMNeT++ is a discrete C++ simulation library and framework, mostly used for network simulations [28]. OMNeT++ has a generic modular architecture that allows for a wide spectrum of simulation types. Each simulation model consists of modules that communicate by messages, and multiple modules can be combined into a single composite module. Module nesting is not limited, and each module is technically reusable. The behaviour of each module can be easily customised as they are implemented in C++. The module hierarchy is defined in the language of OMNeT++ called *NED* (file extension `.ned`). The format for new messages can be defined with message definition files (file extension `.msg`) which are translated into fully-fledged C++ classes during compilation.

However, messages are not the only way to achieve communication between simulation modules. There are also signals that, among other things, can be used as a publish-subscribe pattern of communication. This is especially useful when the producer and the consumer do not know about each other.

OMNeT++ is highly portable, as it runs on all common operating systems. It also provides an Eclipse-based simulation IDE with simulation visualisation. OMNeT++ is free for non-commercial purposes and is commonly used in education.

Information about OMNeT++ is sourced from OMNeT++ Simulation Manual [27].

5.1 Framework INET

The most popular OMNeT++ framework is INET [21]. It provides a set of implemented simulation models for the link layer (`Ethernet`, `802.11`), network layer (`IPv4`, `IPv6`), transport layer (`TCP`, `UDP`) and application layer (`RIP`, `OSPF`, `HTTP`). These simulation models can be used to create and visualise network simulations, and even implement custom simulation models of network protocols by utilising the existing ones as building blocks.

INET also provides a module `ScenarioManager` that can be used to change the parameters of the simulation while it is executing. Support for `ScenarioManager` has to be added explicitly using a specific base class.

The most current release version is 4.3.7 and it is regularly updated.

5.2 Framework ANSAINET

ANSAINET [5] is an extension of INET. It is being developed at Brno University of Technology and implements several network protocols, such as `HSRP`, `IS-IS`, `BABEL`, `OSPFv3`,

and many others. It utilises INET's protocol models and greatly expands the number of available simulation protocols. Some of these popular models are later merged into INET itself, such as `OSPFv3`. Even though ANSAINET's protocol models often use INET modules to some degree, some of them are based on old versions or rely heavily on ANSAINET modules, complicating the merging process.

5.3 State of EIGRP Simulation Model

EIGRP simulation model was created as part of the Master's thesis by Ing. Jan Bloudíček in 2014. It was implemented in the ANSAINET framework and, at that point, only supported IPv4. Thanks to its good design, it was later extended in 2016 for IPv6 by Ing. Vít Rek. EIGRP routing process is enabled within the module `ANSA_EIGRPRouter.ned`, which extends the `ANSA_Router.ned` node. The most current version is implemented in ANSAINET 3.4.0, which is based on INET 3.4.0.

The objective of this project was to take the implementation from ANSAINET 3.4.0 as is, integrate it with the most current INET API, verify its functionality, and to fix any issues. Ultimately, the updated version was accepted and merged into INET itself. As a result, the model is available to all INET users. It is now considered complete with no plans for further extensions.

This version of the model is available on ANSAINET GitHub repository [4].

5.3.1 Structure

The original EIGRP implementation was divided into several modules. The single-responsibility principle (SRP) was met as each module had narrowly defined responsibilities, which were also summarised by each module's name. The source files were structured into folders according to a single convention. The core of the implementation were the PDMs implemented in files `EigrpIpv4Pdm` and `EigrpIpv6Pdm`. These modules handled all the input events and interacted with other modules, especially the data structure modules. Because IPv4 and IPv6 address families in EIGRP's and INET's perspective are very similar, the modules could usually accommodate both with the usage of `C++ templates`. The module versions for IPv4 and IPv6 were split only if larger code divergences were needed. The configuration of the EIGRP process and the IPv6 interfaces was specified in the `.xml` configuration file. These were the notable modules:

- `pdms/EigrpIpv{4|6}Pdm.{h|cc|}` - Modules that implemented EIGRP protocol-dependent behaviour. It handled all input events, such as incoming messages and the following signals: `INTERFACE_STATE_CHANGED`, `INTERFACE_CONFIG_CHANGED`, `ROUTE-_DELETED`. It implemented the interface `IEigrpPdm` for communication with DUAL and the interface `IEigrpModule` for communication with `DeviceConfigurator`.
- `pdms/EigrpMetricHelper.{h|cc}` - Module that implemented the calculation of the metric for both the *Classic* and *Wide* metrics.
- `EigrpDeviceConfigurator.{h|cc}` - Module responsible for loading `XML` configuration files for IPv6 and IPv4 PDMs. It communicated with the PDMs using the `IEigrpModule` interface.

- `EigrpDual.{h|cc}` - Module that implemented EIGRP DUAL FSM in accordance to the FSM specification in RFC 7868 in Section 3.5. It communicated with PDM through the interface `IEigrpPdm`.
- `EigrpRtp.{h|cc}` - Module that implemented the RTP transport layer.
- `messages/EigrpMessage.msg` - File with the definitions of EIGRP messages. `.msg` files were translated into `C++` classes as part of the compilation process.
- `tables/` - The folder contained a table data structure to keep track of `EigrpDisabledInterfaces` and to implement the following tables:
  - `EigrpInterfaceTable` - tracking of enabled interfaces
  - `EigrpTopologyTable` - tracking of routes
  - `EigrpNeigborTable` - tracking of neighbours
  - `EigrpNetworkTable` - tracking of destinations

  It also contained a module that implemented `EigrpRoute`.

5.3.2 Issues

Initial experiments with the protocol model revealed several shortcomings. The following list includes issues caused by faulty implementation, which are present in the ANSAINET's implementation and that had to be resolved prior to merging into INET:

- **i1** - Flawed metric calculation. This issue occurred when a route spanned interfaces of different speeds.
- **i2** - One of the vectors in `TopologyTable` had a reference counter that did not work. It was not subtracted when a route was deleted, and thus, appropriate entries from the topology table were not deleted in some cases.
- **i2a** - Receipt of a message causing a certain route to be deleted, while the same route was at the time being processed by the RTP module, could cause a simulation crash.
- **i3** - The input signals required a specific order; otherwise, the protocol model could loop indefinitely.

Additionally, dependencies on ANSAINET or INET 3.4.0 had to be removed.

- **i4** - Interface IP configuration was loaded via ANSA-specific class. This method of IP configuration is not consistent with other simulation models which are implemented in INET.
- **i5** - Classification of an address family was fully handled by ANSA. The module should instead have direct access to the network layer.
- **i6** - ANSA uses class `ANSA_InterfaceEntry` to represent an interface and it was used throughout most of the modules. This class provides some functionality on top of INET's `InterfaceEntry` class. The functionality had to be preserved despite switching over to the INET variant.

- **i7** - Message creation and dispatching procedures were not compatible with INET API current at the time.

The aforementioned issues will be further described in the EIGRP Implementation in Section 7.1.

5.4 State of BGP Simulation Model

BGP4 protocol model has been included in INET for a long time and was originally developed by the OMNeT++ developers. At that time, the BGP model had many shortcomings, especially related to `BGP UPDATE` messages that did not contain all necessary routes, but also inconsistent FSM states, unsupported message types, and poor recovery capabilities. All of these shortcomings are described in the Master's thesis by Ing. Adrián Novák [25]. He started his work in 2018. At the same time, contributor Mani Amoozadeh worked on his improvements to the BGP protocol model, which were merged into INET later in 2018 [1]. However, the changes made by Ing. Novák were not merged, as their implementation required another rewrite to be compatible with the new INET and also required synchronisation with the changes and features that were made by the Mani Amoozadeh and OMNeT++ developers in the meantime [2].

This version of the model is available on INET GitHub repository [20].

The objective of this work was to take the two available versions (Novák's version and the INET version), analyse and test their features and capabilities, and merge them into a single improved version. Thorough testing was required as major changes were expected to happen.

5.4.1 Structure

The two protocol model versions were fairly similar in structure. The structure of their respective modules was consistent with that of the OSPF and RIP protocol models. Notable parts included the following:

- `/bgpmessages/` - Folder containing the `BgpHeader.msg` file describing the BGP messages. It also contained the module `BgpUpdate.{h|cc}` that helped calculate size of attributes.
- `Bgp.{h|cc}` - Module responsible for the handling of timers and initialisation of `BgpRouter`. In Novák's version, modules `Bgp` and `BgpRouter` were merged into one.
- `BgpConfigReader.{h|cc}` - Module responsible for the processing of `.XML` configuration files.
- `BgpFsm.{h|cc}` - Module that implemented the BGP neighbour Finite State Machine (FSM). This implementation followed the definition of FSM in RFC 4271 [29], Section 8.
- `BgpRouter.{h|cc}` - Module that implemented the core of the BGP process. It implemented the main operation of BGP and communicated with other modules to manage TCP, eBGP, and iBGP sessions. It was also responsible for the handling of received messages.

Features	INET Version	Novák's version
Cisco-like configuration	✗	✓
IPv4	✓	✓
IPv6	✗	✓
Independent of OSPF	✓	✗
Initial Prefix Exchange	✓	✓
UPDATE Message with Multiple NLRIs	✗	✗
UPDATE Message with Withdrawn Routes	✗	✓
NOTIFICATION Message	✗	✗
BGP Table	✗	✗
Start/Stop/Crash Handlers	✗	✗
TCP Closed Handler	✗	✓
Interface State Change Handler	✗	✗
Routing Table Change Handler	✗	✗
Local Pref Attribute	✗	✗
MED Attribute	✗	✗

Table 5.1: Summary of basic features implemented in the INET Version and Novák's Version.

- `BgpSession.{h|cc}` - Module that implemented session-related methods like sending messages.
- `BgpRoutingTableEntry` - The module represented a single route to a destination. It stored the destination prefix and the respective attributes.

5.4.2 Issues

Cross-referencing fixed bugs and added features of INET 4.2 to Novák's analysis of the 4.0 version revealed that some features implemented in his version were still missing from then current INET implementation. Although Novák's version had some notable improvements, the readability and structure of code suffered quite a bit. And many of the problems with the INET version remained. IPv6 support was added by excessive and unnecessary copying of IPv4 methods without leveraging any of the powers of object-oriented programming. Advertisements of prefixes of an address family different from the address family of the underlying architecture was also not supported. The interactivity of the model, although improved, was still hugely limited as only very specific cases were taken into account. What was essentially created was a very hard-to-maintain code with still only very limited interactivity and features. Table 5.1 provides a summary of features between the two versions of the model described above.

Except for the implementation of these features, the following issues had to be solved:

- **i1** - solve configuration divergences from Cisco;
- **i2** - rewrite explicit IPv6 methods with generics;
- **i3** - add missing support for IPv4 over IPv6 and IPv6 over IPv4;
- **i4** - add support for multiple NLRIs in UPDATE message;

- **i5** - fix message format deviations from the standard;
- **i6** - add missing OPEN message AS check;
- **i7** - fix inconsistent TCP sockets;

The aforementioned issues and features are discussed later on in the implementation Chapter 7 in BGP Section 7.2.

Chapter 6

Cisco Configuration

In 2020, Cisco Systems, Inc. had the largest market share in the service provider and enterprise routers market and was doing even better in the Ethernet switch segment [16]. In combination with the fact that EIGRP is basically still proprietary to Cisco, both the EIGRP and BGP models are compared to their Cisco implementation. This chapter presents a basic configuration for both IPv4 and IPv6. All used Cisco routers ran IOS version `15.4` with the *Advance Enterprise Services* set of features. The chapter focusses specifically on the EIGRP and BGP routing processes and requires a proper interface configuration that is not included. The configuration is based on the information provided by Cisco [12, 13].

6.1 EIGRP

Configuration of EIGRP on Cisco devices is available in two modes. **Classic mode** is the old way, having the configuration scattered throughout the router mode. It only supports using the Classic Metric calculation and as such scales badly on modern high-speed interfaces. Configuration of IPv4 and IPv6 is separated. **Named mode** on the other hand is the new way of configuration. Commands are entered in a more hierarchical manner, the mode supports both IPv4 and IPv6 and scales properly on high-speed interfaces with the usage of Wide Metric.

6.1.1 Classic mode

IPv4

In `Global Configuration Mode`, the following command enables the EIGRP routing process for a given autonomous system.

```
R(config)# router eigrp <autonomous_system_number>
```

The `network` command associates a network with an EIGRP routing process. EIGRP process is enabled on all interfaces that much such command and even multiple commands can be entered.

```
R(config-router)# network <ip_address> [<wildcard_mask>]
```

To suppress the generation of Hello messages on specific interfaces, the `passive-interface` command can be used.

```
R(config-router)# passive-interface <interface_id>
```

The K-Values can be adjusted with the command `metric weights`. The K-Values have to match between routers in order to establish an adjacency.

```
R(config-router)# metric weights 0 <K1> <K2> <K3> <K4> <K5>
```

IPv6

IPv6 unicast routing must be enabled manually.

```
R(config)# ipv6 unicast-routing
```

Similarly to IPv4, the EIGRP routing process is enabled with the following command.

```
R(config)# ipv6 router eigrp <autonomous_system_number>
```

`32-bit router-id` has to be specified.

```
R(config-rtr)# eigrp router-id <A.B.C.D>
```

The K-values can be adjusted the same way as for IPv4.

```
R(config-rtr)# metric weights 0 <K1> <K2> <K3> <K4> <K5>
```

Unlike the IPv4 configuration, where the `network` command is used, the IPv6 EIGRP process must be enabled in the interface configuration mode of each interface.

```
R(config-if)# ipv6 eigrp <autonomous_system_number>
```

Similarly to IPv4, Hello messages can be suppressed on specific interfaces back in the EIGRP process configuration mode.

```
R(config-rtr)# passive-interface <interface_id>
```

Upgrade to Named mode

Both IPv4 and IPv6 `Classic` configurations can be upgraded to `Named` with the following command. This performs a proper conversion, which preserves the current configuration and allows utilisation of new features.

```
R(config-router)# eigrp upgrade-cli <instance_name>
```

6.1.2 Named mode

In `Global Configuration Mode`, following command enables the EIGRP routing process with the provided `name`.

```
R(router)# router eigrp <virtual_instance_name>
```

Command `address-family` enters configuration for the specified address family.

```
R(router-router)# address-family <ipv4|ipv6> autonomous-system
<autonomous_system_number>
```

`Router-id` can be specified with following command.

```
R(router-router-af)# eigrp router-id <A.B.C.D>
```

The K-Values can be customised in the same way as with the Classic mode. K_6 can also be included.

```
R(config-router-af)# metric weights 0 <K1> <K2> <K3> <K4> <K5> <K6>
```

With the IPv6 address family, there is no `network` command. All IPv6 interfaces are enabled for IPv6 EIGRP by default. For the IPv4 address family, the command `network` must be used in the same way as with `Classic mode`.

```
R(config-router-af)# network <ip_address> [<wildcard_mask>]
```

The command `af-interface` enters the interface-specific mode. A unique keyword `default` can be used to specify all interfaces.

```
R(config-router-af)# af-interface <interface_id>|default
```

The command `passive-interface` suppresses the generation of Hello messages on a given interface, while the command `shutdown` completely removes the interface from the EIGRP process.

```
R(config-router-af-interface)# passive-interface
R(config-router-af-interface)# shutdown
```

6.2 BGP

The BGP configuration process is similar to the `Named` mode of EIGRP configuration. Similarly, a single BGP process can accommodate multiple address families.

In `Global Configuration Mode`, following command enables the BGP routing process for a given local autonomous system.

```
R(config)# router bgp <local_AS_number>
```

`32-bit Router-id` can be specified with following command.

```
R(config-router)# bgp router-id <A.B.C.D>
```

Command `neighbor` specifies a BGP peer. IPv4 and IPv6 peers and their autonomous system number can be added with the following command. Setting `remote_AS_number` the same as `local_AS_number` identifies iBGP, while different AS numbers identify eBGP. Multiple neighbours can be specified.

```
R(config-router)# neighbor <ip_address> remote-as <remote_AS_number>
```

The use of a specific interface as a source of BGP traffic to a specific peer can be chosen with the following command.

```
R(config-router)# neighbor <ip_address> update-source <interface_id>
```

The configuration for a specific address family can be entered with the following command.

```
R(config-router)# address-family <ipv4|ipv6> unicast
```

An advertised network can be added with the following command. Multiple `network` commands can be entered. If such network is to be advertised to BGP peers, an exact match with the routing table must be found.

```
R(config-router-af)# network <ip_address> mask <net_mask>
```

Neighbour peering can be enabled with the following command. If multiple address families are used, it is a good practise to explicitly disable unwanted peering by the `no` version of this command.

```
R(config-router-af)# neighbor <ip_address> activate
```

Chapter 7

Implementation

The following chapter contains a description of the implementation portion of the work. Both simulation models, as described in Section 5, are merged into the latest version of the INET framework.

EIGRP portion is much shorter as the model already included most of the basic features that make the model interactive during the simulation. The focus is solely on fixing found issues and migrating to the new INET API.

BGP portion, on the other hand, goes into more detail. It is primarily caused by the lack of numerous basic features that are commonly found in other models, like RIP or OSPF. Thereby, the original versions of the model only allow for limited simulation interactivity. On top of that, the features implemented by Novák, while functional, are poorly designed.

This chapter requires at least basic knowledge from Chapters 3 and 4. Classes, message definition files, and even individual files, which are mentioned in Chapter 5, are referenced here.

Messages, names of methods, classes, files or modules are set in `monospace` while states of EIGRP routes (*Passive* or *Active*) and FSM states in general are set in *italic*.

The chapter text can hopefully be used by other developers or OMNeT++ enthusiasts to broaden their knowledge about the functionality of this particular model and INET/OMNeT++ in general. The specific meaning of the C++ terminology used in the text can be found for example in C++ Glossary by Bjarne Stroustrup [32]. The main sources of information about the internal workings of INET and OMNeT++ are the INET Developer's Guide [18] and OMNeT++ Manual [27].

7.1 EIGRP Implementation Details

The following sections describe the results of my efforts to handle the EIGRP simulation model's flaws listed in 5.3.2. The issues are addressed in the order in which they were listed earlier. Additionally, a description of the merging process is included in Section 7.3.1.

The overall quality of the original simulation model is very high, in the author's opinion. Half of the issues found, namely **i4, i5, i6, i7**, are caused by the transition to the new INET API. Issues **i1, i2(a), i3** were not difficult to fix. Overall, most of the work related to this model was related to the migration process to the new INET API and to the removal of ANSAINET dependencies. The target INET version was `4.2`, the most recent at the time. The high-level structure of the model was left almost untouched.

7.1.1 Metric Calculation (i1)

The metric calculation of a route spanning multiple links with different bandwidths is flawed. This bug is caused by the usage of the `getMax()` function instead of `getMin()` inside the module `EigrpMetricHelper::adjustParam`. This function is used to pick a single bandwidth value from either the value received with the route or the parameters of the interface the route was received on. As mentioned in RFC 7868 [31] in Section 5.6.1.1, the bandwidth is the **lowest** interface bandwidth along the path and not the highest. The validity of this change was later verified by comparing it with the Cisco implementation.

7.1.2 Topology Table (i2)

The `TopologyTable` is internally composed of two instances of `std::vector`: `RouteVector` and `RouteInfoVector`.

- `RouteVector` is a `std::vector` of `EigrpRouteSource` instances. It stores all route sources to all destinations. Each entry is essentially identified by a combination of a `routeInfo`, which specifies a particular destination prefix, and a successor. Each `EigrpRouteSource` additionally stores route parameters, e.g., metric or next-hop.
- `RouteInfoVector` is a `std::vector` of `EigrpRoute` instances. It contains all unique destinations. Each entry in this vector tracks the state of the DUAL FSM, the number of successors, the feasible distance, etc.

The relationship model of the entities `EigrpRouteSource` and `EigrpRoute` is shown in Figure 7.1. Each `EigrpRouteSource` is always associated with a single destination (`EigrpRoute`) while each destination (`EigrpRoute`) requires at least one `EigrpRouteSource`, otherwise it can be deleted. For this purpose, `EigrpRoute` tracks the number of associated `EigrpRouteSource` instances.

However, this number is not properly decremented and the entry `RouteInfo` persists even if there is no `Route` with the same destination (issue **i2**). This could even be observed during the simulation itself, since `RouteInfoVector` is one of the WATCHED (i.e., readable during the simulation) vectors. Unfortunately, resolution of this bug introduced a much more critical one.

7.1.3 Query/Reply Crash (i2a)

The receipt of a `QUERY` signifies that a neighbour is in *Active* state for a specific route or routes advertised in the message. The message is processed by the RTP and DUAL modules. Eventually, a `REPLY` message is generated and sent to the RTP module. The message is enriched with RTP fields and sent back to the PDM and consequentially to a neighbour. In the following specific case, the simulation may crash due to the solution of the issue **2**.

1. A `QUERY` containing route `A.B.C.D` is received.
2. A `REPLY` is generated and sent to the RTP module.
3. A new event is raised, removing the last `Route` for destination `A.B.C.D` from the topology table and thus also removing the associated `RouteInfo`.
4. A `REPLY` containing route `A.B.C.D` is received from the RTP module.

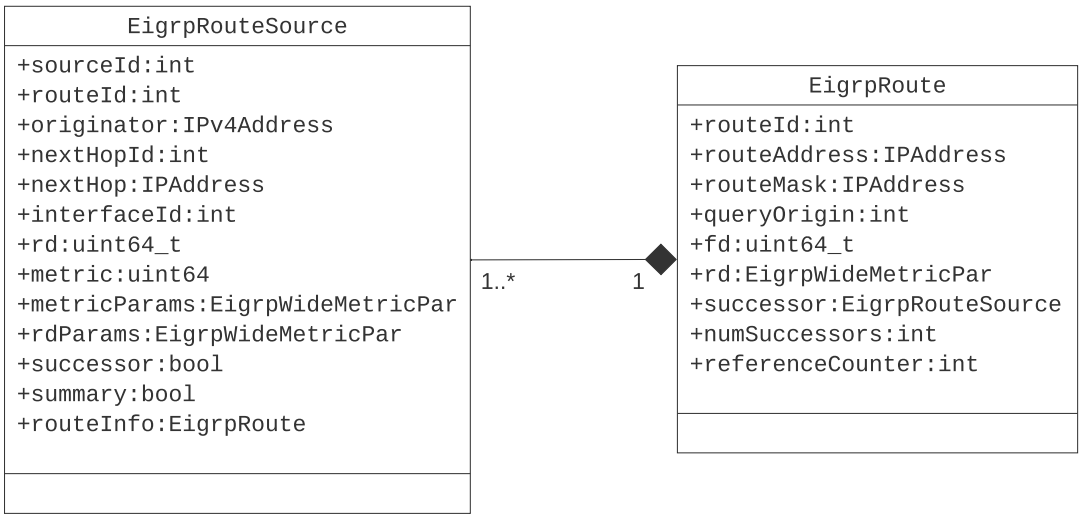


Figure 7.1: Class diagram describing the relation between `EigrpRouteSource`, used by `RouteVector`, and `EigrpRoute` which is used by `RouteInfoVector`. The most important class attributes are included. Depicted classes are used as data structures and as their methods and not important, they are not included in the diagram.

5. During the processing, destination `A.B.C.D` is not found in the `RouteInfoVector` and the simulation crashes (issue **i2a**).

To fix the aforementioned bug, the reference counter is increased for all destinations contained in a `REPLY` that is being sent to the RTP module. After the PDM processes the `REPLY`, the counter is decreased back and the appropriate `RouteInfo` is deleted if its reference counter reached 0.

7.1.4 The Order of Signals (i3)

The EIGRP simulation model subscribes to three types of signals:

- `interfaceStateChangedSignal`;
- `interfaceConfigChangedSignal`; and
- `routeDeletedSignal` (from RT).

Signals are usually received in batches, which means that a single event can cause multiple signals to be emitted. For example, an interface shutdown event causes `interfaceStateChangedSignal` to be emitted. At the same time, `routeDeletedSignal` can be emitted if the shutdown interface was configured with an IP address, as the formerly directly connected network may no longer be reachable.

The problem is caused by the fact that the reception of these signals is expected to happen in a specific order, where `routeDeletedSignal` is always the last (the EIGRP process expects to be informed of an interface shutdown before the removal of a RT route takes place). If this order is not followed, as is the case with the new INET API, EIGRP first receives `routeDeletedSignal`, finds that the interface associated with the deleted RT

entry is still active, and attempts to place the route back to the RT. Unknown implementation details cause the re-added entry to be immediately deleted again, emitting another `routeDeletedSignal` and causing an indefinite loop.

To counteract this problem, the model is not allowed to process the deletion of a single entry multiple times in a sequence. The solution is not ideal but prevents the issue from manifesting. Signal handling could be rewritten in the future.

7.1.5 Interface Configuration (i4)

EIGRP model, as it exists in ANSAINET version 3.4.0, is configured through an ANSA specific class `MultiNetworkNodeConfigurator` that configures the interfaces with IPv4 or IPv6 address specified in the configuration file. As this class is not available in the INET framework, the configuration of the simulation model interfaces had to be redone, taking inspiration from the means of configuration of other similar simulation models, such as OSPFv2 or RIP.

The solution to IPv4 configuration is the `Ipv4NetworkConfigurator` module. It is already used in many examples for other protocols and should be intuitive for INET users. It assigns per-interface IP addresses, strives to take subnets into account, and can also optimise the generated routing tables by merging routing entries. It is an ideal substitute and does not require any code changes.

The IPv6 solution is a more complicated matter. Since the IPv6 configuration has not yet been standardised in INET, authors of the simulation models employ their own solutions. This means that the configuration of interfaces (i.e., the settings of individual IP addresses) is handled by the models themselves during initialisation of the simulation. For this reason, the `EigrpDeviceConfigurator` class was extended to allow such an action.

7.1.6 Classification of Address Family (i5)

To allow the EIGRP model to handle both IPv4 and IPv6 traffic, it requires a mechanism of determining the destination EIGRP PDM. In the case of ANSAINET, the models do not handle this themselves. The categorisation of traffic is performed by `ANSA_MultiNetwork-Layer` multiplexers. From the models' perspective, no extra work is necessary. However, such solution does not have an INET counterpart and an explicit splitter module was needed. The created module registers directly with the network layer on port 88, receives packets of both address families, and forwards traffic to the appropriate PDM. A new structure of the EIGRP simulation model is shown in Figure 7.2.

7.1.7 ANSAINET Interface Dependency (i6)

The EIGRP model is built on top of the ANSAINET router module. As such, it employs special interface modules called `ANSA_InterfaceEntry`, derived from INET `Interface-Entry` (nowadays `NetworkInterface`). These objects are used to represent a single interface with its own attributes, such as IP address, MAC address, or bandwidth. The class `EigrpInterface` creates an abstraction of the interface for the EIGRP process. Since EIGRP employs multiple parameters (chosen by the K-values), and INET `InterfaceEntry` only provides for the bandwidth attribute (as `datarate` - K1), the remaining attributes (i.e., delay, reliability, and load) are stored in `ANSA_InterfaceEntry`. This is problematic as the INET interface does not have access to any additional attributes except for the bandwidth.

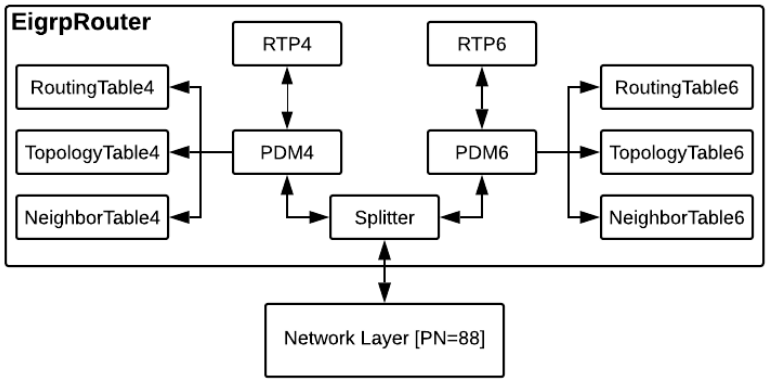


Figure 7.2: Structure of the EIGRP simulation model in INET 4.2 with new splitter. PN stands for Protocol Number, 88 in case of EIGRP.

There are two solutions to this issue. The aforementioned attributes could be moved either to INET's `InterfaceEntry`, which would make them accessible to other models, or to `EIGRPInterface`. Since these attributes are not used by other models, they were moved to the latter. Every usage of the `ANSA_InterfaceEntry` class in the code had to be replaced by `InterfaceEntry` as well.

7.1.8 Integration of Packets (i7)

INET version 4.0 introduced a new and more standardised way of sending messages to outside networks, although composite modules may still use the old way internally. The new method uses the `Packet` data structure, which is built on top of `chunks`. Users define their own chunk types in the `.msg` message definition files and describe the desired message format. Chunks must be constructed with a call to the `makeShared` method instead of using the `new` operator, as was the case in previous versions of INET. The packets themselves can contain several chunks inserted and removed by different protocols as they are passed through the protocol layers. Chunks can be inserted into a packet with functions `insertAtBack` or `insertAtFront` as well as removed with `popAtFront` and `popAtBack`. Packets can be supplemented with metadata via tags, which specify the outermost protocol of the packet, the outgoing interface, the destination address, the TTL, etc. These metadata were previously carried in some form of a `ControlInfo` structure. `Message Dispatchers` connect modules and allow automatic dispatch of packets or messages. The `DispatchProtocolTag` must be attached to the packet as it allows the dispatcher to direct the message to the intended recipient.

To adjust the model to this new mechanism, modifications were made to methods `processMsgFromRtp` and `processMsgFromNetwork` and to their helper methods in both PDMs:

- `processMsgFromRtp` - Helper methods responsible for creating messages (e.g., `createHelloMsg`, `createUpdateMsg` etc.) were updated to create chunks instead, compute their size, insert them into newly created `Packet`s and return those. To reflect this change, `Msg` in the name signature of the helper methods was replaced by `Packet`. The method `processMsgFromRtp` is responsible for attaching the appropriate tags, such as the protocol, the destination interface, the TTL, and sending the message to the outbound gate.

- `processMsgFromNetwork` - Similarly, the helper methods, which are responsible for processing received messages, were changed to work with packets and tags instead.

`ControlInfo` was completely removed from the code as it had no more use.

7.2 BGP Implementation Details

This section describes my efforts to address the issues found in the BGP simulation model mentioned in Section 5.4.2. These efforts were then followed by implementation of the missing features previously shown in Table 5.1. The various approaches taken to deliver these features are then described as well.

Unlike in the EIGRP section, these issues and missing features were not resolved in any particular order and as such, the following text is structured differently. The described issues are sometimes mentioned within the text to point out their relevance to a particular paragraph.

Furthermore, since there are two other simulation models of BGP for INET, as mentioned in Section 5.4, and this work talks about their successes and failures, they are often mentioned. The version currently in INET 4.3 is called the INET version, while the version of the model made by Ing. Novák is called Novák's version.

7.2.1 Configuration of the Simulation Model

One of the first aspects facing the user is the configuration of individual routers. The INET version uses a more 'declarative' approach rather than the 'imperative' approach found on many real devices. Rather than explicitly configuring peers for each router one by one, the INET version of the BGP model is configured via an XML file whose most significant elements and attributes are described below. Some default aspects of the BGP model (such as *connectedCheck*, *ebgpMultihop*, and *externalPeerStartDelayOffset*) can only be specified via module parameters, which unfortunately decentralises configuration.

1. An array of `<AS>` elements with required attribute *id*: defines existence of an autonomous system whose number is specified by the *id* attribute. Child elements with tag `<Router>` and the *internalAddr* attribute are used to specify IP addresses of interfaces of routers that are participating in the iBGP process in a given AS. Redistribution is enabled automatically. `<Router>` elements can contain two types of child elements:
   - `<Network>` with attribute *address* which inserts resolvable prefixes directly to the `BgpRoutingTable`;
   - `<Neighbor>` with attributes *nextHopSelf* and *localPreference* which are used for specific iBGP neighbour configuration.
2. An array of `<Session>` elements with required attribute *id*: defines an eBGP peering session between routers specified with exactly two `<Router>` child elements. These elements have required attribute *externalAddr* which specifies an interface of a router participating in this eBGP peering. Optional child element `<Neighbor>` with required attribute *address* and two optional attributes *connectedCheck* and *ebgpMultihop* can be used to modify eBGP behaviour towards a specified neighbour.

Until recently, there also existed a TCP FSM bug [15] that required the configuration to be loaded from a single file. Originally, BGP nodes would create a global time order of peer establishment to avoid the possibility of two simulation entities performing two TCP connect operations at the same simulation time (which the simulation did not properly support).

With the above style of configuration, there usually only exists a single configuration file, and all BGP-enabled nodes parse this file to check if any of the *internalAddr* or *externalAddr* attributes match any of their configured IP addresses. Admittedly, users who are not familiar with configuring BGP on real devices can set up their desired simulation fairly quickly. However, there are several drawbacks to the aforementioned configuration style of the BGP model of INET:

- single internal address for iBGP;
- missing support for MP-BGP and IPv6 configuration and static route insertion;
- unfriendliness caused by the combination of module parameters and `.xml` file;
- difficult conversion between model configuration and real-life device configuration;
- unfamiliarity to people with a networking background, who are more likely to use such a model in the first place.

New Configuration File

The following section presents a new style of Cisco-like configuration that supports IPv4 and IPv6, partially removes the dependency on the OSPF module, and extends the set of supported BGP route attributes with `Local_Pref` and `MED`. This new configuration structure is loosely based on Novák's version, which itself is based on MP-BGP configuration on Cisco devices. The following subsection ultimately describes the resulting configuration experience of this work's implemented model.

In addition to normal IPv4 configuration, the new format allows for IPv6 configuration, combination of the two, and even advertisement of IPv4 prefixes over IPv6 infrastructure or vice versa (issue **i3**). This behaviour is explained in more detail in the MP-BGP section later on. Furthermore, the new configuration allows for interface IP address assignments and for insertion of static routes. Any code related to the calculation of TCP connect delays was cleaned up, as the previously mentioned TCP bug was fixed after discussion with the INET's developers. The structure of the new configuration is shown in Appendix D. Separation between the global BGP configuration and configuration for a specific address family closely follows configuration on Cisco devices (issue **i1**), unlike the solution found in Novák's version, which merges everything under specific address family. An example of the new configuration file is shown in Appendix E. A non-exhaustive list of elements' connotations is the following:

- `<TimerParams>`: Sets global BGP timers.
- `<Router>`: Represents a single simulation node. The attribute *name* must match the desired node name specified in the `.NED` file. Optionally, the *id* attribute can be used to set global (stored in RT) router-id identifier.
- `<Interface>`: Represents the interface identified by the *id* attribute. This attribute can also be used to identify a loopback interface.

- `<Ipv{4|6}>`: Sets a specific AF IP address.
- `<Bgp>`: Contains BGP settings. The attribute *as* assigns the BGP process with a specific AS number.
- `<Bgp-router-id>`: Sets BGP specific 32-bit router-id identifier.
- `<Neighbor>`: Represents a specific peer identified by the attribute *address*. The following additional attributes are supported if address family is not specified:
  - *remote-as*: Non-optional attribute that specifies remote peer's AS number. This value is used to differentiate between iBGP and eBGP sessions.
  - *disable-connected-check*: disables check of the peer's IP against RT entries tagged as directly connected. TTL is not modified.
  - *ebgp-multihop*: Sets the messages' TTL for the specified neighbour.
  - *update-source*: Sets the peering source to a specific loopback interface.
  - *local-pref*: Sets the local preference attribute for NLRIs which are advertised by the specified neighbour.
  - *med*: Sets the Multi-Exit-Discriminator attribute for NLRIs which are advertised to the specified neighbour.

  If the address family is specified, a different set of attributes is available:
  - *activate*: Activates or deactivates the peer for a specific address family.
  - *next-hop-self*: Forces change of the next-hop attribute to the local peering source interface address.
- `<Address-Family>`: Groups configuration for the specific address family specified by the required attribute *id*.
- `<Network>`: Enables a prefix, identified by the attribute *address*, for BGP advertisement. In order to be installed in the BGP table and thus advertised to BGP peers, the prefix must have an exact match in the RT.
- `<Route{4|6}>`: Adds a static route of the specified prefix, output interface, and next-hop address to the RT of the given AF.

During the international OMNeT++ Summit in September 2021[1], the INET development team requested that the new configuration format coexists alongside the old format. This did not include extending the old format with new features; just for it to behave in the same way as before the introduction of these features. However, the old way of configuration has not been implemented yet.

MP-BGP Configuration

As mentioned above, the new configuration supports both IPv4 and IPv6 address families with the same set of features. The advertisement of IPv4 prefixes over IPv4 infrastructure is the most common and therefore does not require the attribute *activate* to be specified. IPv6 always requires explicit *activate* and so does the advertisement of IPv4 over IPv6

[1] https://summit.omnetpp.org/2021

infrastructure (issue **i3**). All possible scenarios of the AF configuration are shown in Appendix F. Since the next-hop attribute value cannot be used as-is if the AF of the advertised prefixes and of the infrastructure do not match, Cisco devices require a mechanism (e.g., `route-map`), which changes the next-hop to a usable address explicitly. For simplicity, the implemented BGP model automatically changes the next-hop value to the appropriate AF address found on the same source interface. If such an address is not found, the next hop is mapped in the same way as on Cisco devices, as shown in Figure 7.3.

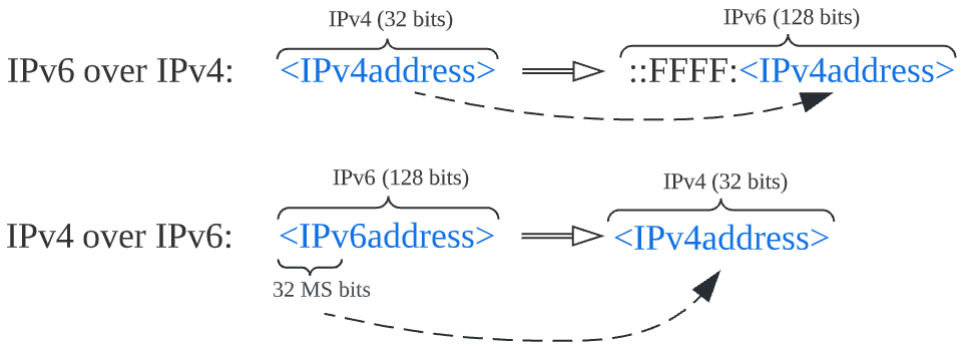


Figure 7.3: Generation of the next-hop address for when advertised AF does not match the AF of the infrastructure. MS stands for *most significant*.

7.2.2 Support for Multi-Address Family

This section describes the process of extending the simulation model for IPv6 support. The configuration was presented in the previous section.

Because the old model was not exactly designed and coded with IPv6 in mind, many changes were needed. While the code from Adrián Novák's BGP repository is no doubt functional, it contains a lot of duplicated code, which, inadvertently and unnecessarily, bloats the codebase. The goal was to have as few address-family-specific methods as possible. All IPv6 addresses mentioned are considered global unicast addresses, since that seems to be the most common scenario. In the future, missing support for link local addresses could be implemented by changing a couple of methods that specifically look for global addresses on interfaces. The INET version uses the following IPv4-specific data structures:

- `BgpRoutingTableEntry`: Represents a single route to a destination. It is derived from `Ipv4Route` and adds BGP specific class attributes like path attributes (ORIGIN, AS_PATH, NEXT_HOP, LOCAL_PREF etc.) while leveraging `Ipv4Route`'s attributes for storing destination prefix, its mask, and next-hop. This makes it easy to install such a route in the RT.
- `BgpRoutingTable`: A `std::vector` of `BgpRoutingTableEntry` instances representing a BGP routing table.
- `AdvertiseList`: A `std::vector` of `Ipv4Address` instances created based on the contents of the `Network` XML elements in the configuration. During node initialisation, these addresses are installed in `BgpTable` if they can be matched with an existing RT entry.

- `PrefixList`: A `std::vector` of `BgpRoutingTableEntry` instances used to filter routes based on their AS numbers and prefixes.

To add support for IPv6, the model had to be extended with a few minor data structures to store different types of routes.

A simple solution would be to create a `BgpRoutingTableEntry6` derived from `Ipv6Route` to accommodate IPv6 prefixes and to add separate methods for this new type. However, such a solution would result in code redundancy, as the only difference between IPv4 and IPv6 `BGPRTEntry` is in the destination prefix and the next-hop address types. The internal logic for installing routes to internal structures remains the same.

A valid solution would be to use C++ templates [34], which would allow the code to be written with *generic types*, whose functionality can be adapted to IPv4 and IPv6 without the need for much redundant code. The code written in these templates is used by the compiler to generate object code for template instances [33], however, it can be difficult to debug.

Instead of diving into templates, the occurrences of `Ipv4Address`, which represented the destination prefix, the next hop address, or the peer's address, were replaced with a generic interface for the network address: `L3Address` [17]. The constructor for `L3Address` exists for both IPv4 and IPv6 addresses, and the method `getType()` returns the type of address that is used to branch out the code if necessary. The methods `toIPv4()` and `toIPv6()`, returning derived types, are used to encode the address in `UPDATE` messages and for the installation in the RT.

The derivative of the `Ipv4Route` class was removed from `BgpRoutingTableEntry` and instead 3 properties were added: `destination` and `nexthop` of generic type `L3Address` to accommodate missing `Ipv4Route` attributes, and an unsigned number to store information on the length of the prefix. With these changes, it is now possible to store either IPv4 or IPv6 BGP route. All methods that work with IPv4 specifically had to be rewritten to work with `L3Address`. Fortunately, TCP methods already fully supported the `L3Address` interface.

The IPv4 and IPv6 routes could technically be stored in the same data structures, but keeping them separate makes it cleaner from the router's perspective. A non-exhaustive list of added structures includes:

- `BgpRoutingTable6`: A `std::vector` of `BgpRoutingTableEntry` to store the best path for each known IPv6 destination.
- `AdvertiseList6`: A `std::vector` of `Ipv6Route` instances to store prefixes and their prefix lengths configured via `Network` XML elements.

The type of `AdvertiseList` was also changed from `Ipv4Address` to `Ipv4Route` to allow for storage of prefix length. Helper methods were rewritten with a generic address type as well. This includes methods such as `isResolvable`, `isInTable`, `isDestinationInTable` and others.

To allow routers to exchange information on supported address families, the set of supported `Optional Parameters` was extended with *capability*. Both routers announce their supported address families through these fields within `OPEN` messages, and the established state is only reached if and only if at least one address family is common between the two peers. Otherwise, a `NOTIFICATION` message is generated and both the TCP and BGP sessions are closed. A peer that is not supporting a given address family does not send any `UPDATE` messages of that kind. Currently, no other optional parameters are supported.

Lastly, the `BgpRouter.ned` module, which extends the generic `router`, had to be enabled for IPv6 and extended by `routingTableModule6` adding IPv6 RT. The OSPF module was disabled as it is does not have any uses. Future work could include implementing proper support for OSPF, RIP and EIGRP redistribution. In the author's opinion, these would bring a lot of value to the simulation model.

MP Path Attributes

Since the `UPDATE` message format differs for IPv6 prefixes, a method was added to specifically send IPv6 NLRIs. The following message definitions, which contain the `UPDATE` message attributes `MpReachNlri` and `MpUnreachNlri`, were added to the message definition file `BgpHeader.msg`, such a snippet is shown in Listing 7.1.

```
class BgpUpdatePathAttributesMpReachNlri extends BgpUpdatePathAttributes
{
  typeCode = MP_REACH_NLRI;
  optionalBit = true;
  length = 0;

  unsigned short AFI @enum(AFITypes) = Ipv6; //2 octets
  unsigned char SAFI @enum(SAFITypes) = Unicast; //1 octet
  unsigned char NextHopLength = 16;
  BgpUpdatePathAttributesNextHop6 nextHop; //16 octet nexthop
  unsigned char reserved = 0; //1 octet reserved
  BgpUpdateNlri6 NLRI[];
}

class BgpUpdatePathAttributesMpUnreachNlri extends BgpUpdatePathAttributes
{
  typeCode = MP_REACH_NLRI;
  optionalBit = true;
  length = 0;

  unsigned short AFI @enum(AFITypes) = Ipv6; //2 octets
  unsigned char SAFI @enum(SAFITypes) = Unicast; //1 octet
  BgpUpdateNlri6 NLRI[];
}
```

Listing 7.1: Definitions of MP attributes to carry IPv6 prefixes.

Along with the additions mentioned above to the message definition file, a very minor deviation in message format (BGP header) was fixed while the file was being reviewed (issue **i5**).

7.2.3 Redesign of Node's Operation

The following section presents a new operation loop that was redesigned to provide a clear chain of actions for the routes to pass through. This change addresses multiple issues and missing features, such as various event handlers or missing BGP table.

Many shortcomings of the model became apparent during the process of adding IPv6 support and its initial testing. Although some aspects, such as the initial route exchange, had just a few minor issues, other aspects, especially behaviour after topology changes, were either only implemented partially or absent altogether. The following issues are addressed in this section.

- `OPEN` messages were not validated by the receiver.

- The `BGPTable` was missing. Its purpose is to store all received routes and especially those that were not installed in `BgpRoutingTable`. If the best route for a given destination became unavailable, another loop-less route from `BGPTable` could be used as a fallback.
- No signals are subscribed. This makes the operating node completely oblivious to any changes to:
  a) the state or configuration of an interface;
  b) RT additions and deletions by other processes (i.e. routing protocols).
- Handler associated with a closure of an active TCP socket is missing (INET version only).
- `AdvertiseList`, which stores prefixes configured by the `network` command, only allows holding routes with an exact match in the RT.
- Handling of routes individually instead of in bulk, resulting, for example, in each `Update` message containing only a single NLRI prefix. The `UPDATE` message's NLRI is an array, and multiple NLRI prefixes can be grouped, if they share the same attributes. Withdrawn routes can always be grouped.

`OPEN` messages are newly checked for the expected AS identifier. If unexpected value is detected, a `NOTIFICATION` message is generated and the TCP connection with the peer is closed (issue **i6**).

With the introduction of the BGP table structure, the way of managing routes had to be completely changed. The diagram in Figure 7.4 shows how the three main route data structures (i.e., `AdvertiseList`, `BgpTable` and `BgpRoutingTable`) interact with each other. What was essentially created is a chain of consecutive subroutines that takes an incoming event together with its associated routes, recalculates the contents of these internal structures accordingly, updates the AF-appropriate RT, and notifies relevant peers about the changes. The context of the event that triggered the computation is passed through the chain as `Source Session ID`. In case of locally originating events, it is set to `-1`, otherwise it is set to the `Session ID` of the peer that triggered it (i.e., events like `Initialize, Start, Interface/RT changed` are locally originated while `Update Message Received, TCP Closed` and `BGP Established` need a context with information about the peer that triggered it).

The following text takes into account only a single address family, but applies to both.

① Advertise List

First, `advertiseList` is populated with prefixes entered by the `network` command by `configReader` class. This only happens once during node initialisation when the configuration is read. The newly added routes do not have to be exactly matched in the RT. This structure is simply used to store configured prefixes. However, this also implies that new prefixes cannot be easily added to `advertiseList` during simulation. Once the configuration file parsing process is complete, every prefix in `advertiseList` is processed into `BgpRoutingTableEntry` and marked as valid or invalid, depending on whether the prefix can be exactly matched in the RT (*check* operation). A vector of these entries is passed to update `BGPTable`. The *check* operation is executed either after the configuration file has

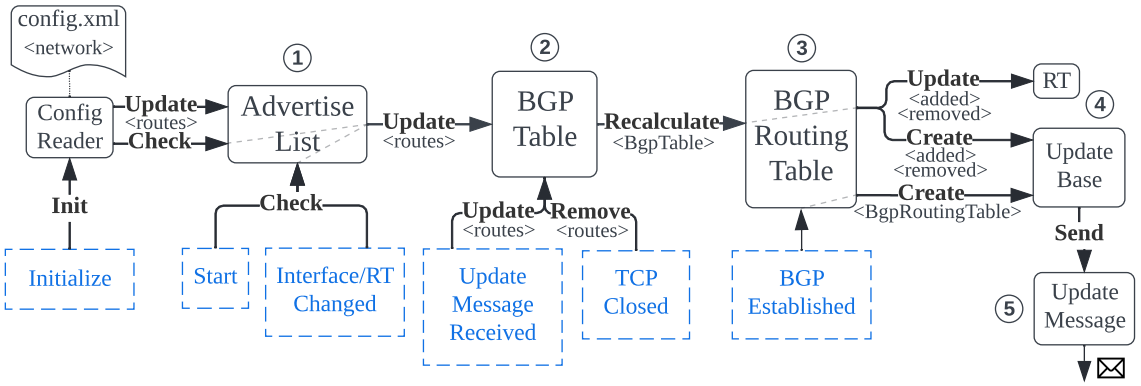


Figure 7.4: Simplified diagram of the interactions between route data structures. Input events are highlighted in blue.

been parsed, when the node has started, or when a signal notification regarding Interfaces or RT has been received.

② Update of the BGP table

This subroutine takes a vector of routes as input. Each invalid entry is removed from `BgpTable` and each valid entry is checked for presence and added if necessary. The routes received in `UPDATE` messages are processed in the same way; both the withdrawn routes and the NLRI are processed into `BgpRoutingTableEntry` instances, with the withdrawn routes also marked as invalid. If an active TCP socket closes, all routes received from that particular peer are invalidated. With this updated `BgpTable`, `BgpRoutingTable` can be recalculated.

③ Calculation of the BGP Routing Table

The algorithm for the calculation of `BgpRoutingTable` is shown in Algorithm 1. It consists of two separate steps:

1. All `BgpRoutingTable` entries are checked for:

   (a) a presence in `BgpTable` as they could have been removed in the previous step;

   (b) next-hop resolvability in the appropriate RT as an RT change could have triggered recalculation;

   If either of these checks fail, the route is:

   (a) added to the `removedRoutes` vector;

   (b) removed from `BgpRoutingTable`.

   At the end of this step, the `BgpRoutingTable` is guaranteed not to contain any routes not present in `BgpTable` and the `removedRoutes` vector contains all these deleted routes.

2. All `BgpTable` entries are checked for next-hop resolvability. The same destination network is then located in the `BgpRoutingTable`. If no such route is found, the

entry is added. Otherwise, if an entry to the same destination is already present, both entries are passed further to the `tieBreakingProcess`. If the new route is better from BGP's perspective, this results in the new route being added to the `BgpRoutingTable` and the old route being removed. If the route was, in fact, added to the `BgpRoutingTable`, it is also added to the `addedRoutes` vector and removed from the `removedRoutes` vector if present.

This procedure leaves `BgpRoutingTable` in sync with `BgpTable` and also returns two vectors: `removedRoutes` and `addedRoutes`, which reflect changes. If the `Established` state is newly reached with a particular peer, `BgpRoutingTable` is only read, and the result of this operation is the whole table.

④ Creation of the Update Base

While `removedRoutes` entries do not require additional processing, since only the prefix and its length are contained in the final message, the advertisement of `addedRoutes` entries requires a calculation of attributes. If the whole `BgpRoutingTable` is given as input, it is taken in the same way as the vector `addedRoutes`. The update base is calculated separately for each peer after several checks are performed:

- Peering is in the `Established` state;
- Source session ID is not the same as the ID of the receiving peer session (*Split Horizon)*;
- Either the source or the destination session must be eBGP or the event must be locally originated.

The following attributes are included in the update message base using the `createUpdateBase` method:

- `Origin` - does not require any processing;
- `AS_path` - generally processed, depending on the type of BGP session with the intended receiver of the message:
  - iBGP - does not require processing as the array is just copied;
  - eBGP - the array is copied, AS number of the BGP process is prepended;
- `Next_hop` - generally processed, depending on the type of BGP session with the intended receiver of the message:
  - iBGP - Replaced with BGP's local peering source IP address if the route is locally originated (i.e., `0.0.0.0` would not make a good next-hop). This can be overruled if the flag `next-hop-self` is configured. In that case, the next-hop is always replaced in the same way.
  - eBGP - Set to BGP's local peering source IP address.

  If the BGP's local peering source IP address and the route are not of the same address family, the appropriate address family IP address of the same interface is included instead. If such an address does not exist, BGP's local peering source IP Address is mapped into different address families as shown in Figure 7.3 in the configuration section.

- `Local_pref` - copied in the case of iBGP sessions, omitted in the case of eBGP.
- `MED` - copied in the case of iBGP sessions, omitted in the case of eBGP unless configured locally.

The output of the `createUpdateBase` method, called `UpdateBase`, is a vector of triplets as shown in Listing 7.2.

```
typedef struct {
    unsigned char length;
    L3Address prefix;
} BgpUpdateNlriBase;

typedef struct {
    std::vector<BgpUpdatePathAttributes *> pathAttributes;
    L3Address nextHop;
    std::vector<BgpUpdateNlriBase *> NLRIs;
} UpdateBaseEntry;

typedef std::vector<UpdateBaseEntry> UpdateBase;
```

Listing 7.2: Type hierarchy of UpdateBase.

`UpdateBase` is a vector, which contains a set of unique combinations of path attributes (`pathAttributes` and `nextHop`) together with prefixes (`NLRIs`) that match such combination (issue **i4**). Attributes contained in the `pathAttributes` vector are AF agnostic (i.e., `origin, AS_path, Local_pref, MED`). The only attribute that is AF specific is `nextHop` and for that reason it is stored separately. This is done in such a way because `BgpUpdatePathAttributes` type is auto-generated from the message definition `.msg` file which does not include any field which could contain both types of addresses. Nevertheless, each item of `UpdateBase` vector represents one update message for a specific peer. If `removedRoutes` are present, they are sent as *withdrawn routes* or *unreach_nlri* only in the first `UPDATE` message.

⑤ Sending of the Update Message

With the calculated peer-specific and AF-specific `UpdateBase`, the appropriate AF-specific `sendUpdateMessage` method of the receiver's `BGPSession` instance is called. It takes `UpdateBase` and `WithdrawnRoutes` (stripped down `removedRoutes vector`) as parameters and constructs and sends an Update Message for each entry in `UpdateBase`. If the vector `WithdrawnRoutes` is present, it is included in the first message only. If `UpdateBase` is not present but `WithdrawnRoutes` is present, then only a single `UPDATE` message with *WithdrawnRoutes* or *unreach_nlri* is sent.

Signal handlers

Neither the INET version nor Novák's version employs signal subscriptions. The former does not even handle an abruptly closed TCP connection, whereas the latter has only a basic, and in the author's opinion rather crude, implementation of such a handler directly in `BgpFsm`. Furthermore, it has a crucial oversight. The state of connected interfaces that do not participate in the BGP peering process is ignored even if their networks are inserted into `AdvertiseList` and advertised to other peers. Improvements to handlers for TCP connections are described in one of the following sections, while this section focusses on signals and their usage for the BGP model.

Algorithm 1: UpdateBgpRoutingTable

```
Data: BgpTable, BgpRoutingTable
Result: removedRoutes, addedRoutes
foreach BgpEntry route ∈ BgpRoutingTable do
    if !isResovable(route.getNextHop()) then
        removedRoutes.push(route);
        BgpRoutingTable.erase(route);
        continue;
    end
    foundMatch ← false;
    foreach BgpEntry entry ∈ BgpTable do
        if entry.isSameAs(route) then
            foundMatch ← true;
            break;
        end
    end
    if !foundMatch then
        removedRoutes.push(route);
        BgpRoutingTable.erase(route);
    end
end
foreach BgpEntry entry ∈ BgpTable do
    if !isResolvable(entry.getNextHop()) then
        continue;
    end
    if decisionProcess(entry, BgpRoutingTable) == ROUTE_ADDED then
        if removedRoutes.containsDestination(entry) then
            removedRoutes.eraseByDestination(entry);
        end
        newRoutes.push(entry);
    end
end
return removedRoutes, addedRoutes
```

The aforementioned issues greatly limit the use cases for both models, as the simulation of link failures could provide crucial information about the stability of the simulated topology. Fortunately, the addition of signals itself was a very simple process and, thanks to the previously explained routine depicted in Figure 7.4, which handles everything from the recalculation of BGP tables to sending `UPDATE` messages, it required just a few lines of additional code.

First, the `Bgp` class must implement the interface `cListener`, which requires the implementation of a single handler method `receiveSignal`. Second, if the new method is to be called, some signals have to be subscribed to first. The following self-explanatory signals were chosen:

- `routeAddedSignal`;
- `routeDeletedignal`;
- `routeChangedSignal`;

for watching for RT changes, while signals

- `interfaceStateChangedSignal`; and
- `interfaceConfigChangedSignal`

are used to look for state or configuration changes on interfaces.

`Bgp` class subscribes to RT changes because a prefix in the `advertiseList` can become exactly matched by the newly added route and thus should be installed in the `BGPTable` and potentially advertised to peers. The same goes for newly removed routes. RT is a node-wide data structure and can be influenced by other routing protocols, for example.

`Bgp` class subscribes to changes on local interfaces because interface, which provides a *CONNECTED* route, thus allowing BGP to advertise such a route, could be shut down, which should trigger a BGP action.

These signals overlap since a shutdown of an interface triggers both `interfaceStateChangedSignal` and multiple `routeAddedSignal` or `routeDeletedSignal` signals. This is because the RT is automatically recalculated if the state of a local interface changes. Thus, subscription to interface changes is not currently impactful, but this could change in the future. The problem of receiving multiple signals had to be resolved even if only RT changes are watched.

Since the entire RT can be wiped and repopulated as a result of a single interface state change, repeated processing of same signals and improper processing of conflicting signals could cause problems during topology changes. Such changes could cause the model to handle each signal separately, resulting in high CPU utilisation and strange behaviour with BGP generating multiple conflicting `UPDATE` messages, seemingly at the same time. To combat this, a stagger mechanism is implemented. Instead of processing each signal individually, only the final state of the RT is processed. A message, which represents a received signal, is scheduled at the current simulation time by the first received signal. On subsequent signals, a check for the presence of such message is performed. After all signals are handled, the node receives the previously scheduled message which triggers a recalculation of `advertiseList` which continues to the recalculation of `BGP Tables` and so on until every appropriate peer is notified of the changes by `UPDATE` message(s).

Signals must be unsubscribed from, which is handled by the destructor of `Bgp`.

With the aforementioned additions, the model is able to receive events regarding additions and deletions to RT, it changes its state accordingly and causes peers to be informed about the changes. The above was already taken into account in the previous section regarding improvements to node's operation 7.2.3 and thus these events are included in Figure 7.4.

Life Cycle Handlers

Implementing the life cycle abstract class `RoutingProtocolBase` allows the node to handle node-specific operations such as `start`, `stop` and `crash` during simulation through `scenarioManager`. If this interface is not implemented by the module, manipulation with the node via `scenarioManager` throws an exception, and thus the simulation fails. Implementing the interface requires the following steps:

(a) `handleMessage` has to be refactored to `handleMessageWhenUp` (meaning the node is UP);

(b) implementation of the following self-explanatory methods:

- `handleStartOperation`;
- `handleStopOperation`;
- `handleCrashOperation`;

All of these methods have to leave the node in a consistent state.

The derivation of the base class `RoutingProtocolBase` replaces the derivation of `cSimpleModule` as the base class. With these changes, the node can now handle above-mentioned operations during the simulation via `scenarioManager`.

7.2.4 TCP Operations

Even though BGP uses TCP to provide a reliable transport layer for its messages, it does not use the typical client-server model; instead, it uses a peer-to-peer model. This means that all peers, two in the case of the BGP session, are coequal and equipotent nodes. In a classic client-server model, the client would perform the TCP *connect* operation while the server would perform the TCP *listen* operation. With BGP, both peers perform both operations: both are actively listening for the incoming `SYN` segment, while both are actively (with reasonable intervals) trying to contact the peer with their own `SYN` segment. This means that each peer needs two TCP sockets for every other configured peer. One socket is used to listen for incoming connections and the other to actively try to connect to the other peer.

In both versions of the studied BGP models, TCP sockets are stored together with peer-specific data in an instance of the `BgpSession` class. All such instances, with each representing one peer, are stored in the `_BGPSessions` map as values, while their `Session ID` is used as key. Both sockets in each peering instance are created with the `create{I|E}bgpSession` methods of the `BgpRouter` class, that is invoked by the `configReader` class after the parsing process of the configuration `.xml` file is completed.

TCP callbacks are implemented by deriving an abstract class `TcpSocket::ReceiveQueueBasedCallback` and by setting such a derived class as a callback for TCP sockets. Both of these sockets use these callbacks. However, there is one problem with the sockets.

If the two TCP connections between BGP peers are opened simultaneously, the simulation cannot detect this, and a socket mismatch happens. In such a case, both routers use `activeSocket` for outgoing messages and `passiveSocket` for receiving messages. When the TCP connection is not initialised simultaneously by both parties, this does not happen and the initialising party uses the `activeSocket` and other uses `passiveSocket` normally.

The following 2 sockets are created for each peer.

- `activeSocket` represent a socket that performs a `connect` operation. It uses port number `50,000` raised by the index of the given peer.
- `passiveSocket` is a socket that performs the `listen` operation. It uses a well-known port `179`.

As mentioned in Section 7.2.3, thanks to the already implemented handler hooked onto the appropriate TCP callback, `BgpTable` recalculation is triggered if a TCP connection to a peer is closed. All routes in `BgpTable` that were received from the no-longer-available peer are invalidated and removed, triggering a recalculation of `BgpRoutingTable` and updating other peers of the changes.

My additional improvement in TCP socket operations is to stop the `activeSocket` from closing if `SYN+ACK` is not received. This socket has to periodically send `SYN` segments until a session is established. Once closed for whatever reason, both sockets must be placed in the initial state (issue **i7**). This was accomplished by the introduction of a couple of self-messages and with more functional callback handlers.

More information about the TCP simulation model can be found in the official INET *Developer's Guide* [18] in Section *TCP Socket*.

7.3 Implementation Conclusion

7.3.1 EIGRP Conclusion

With all known issues addressed and resolved, the EIGRP simulation model was submitted to INET via pull request #570 [3] on October 30, 2020. After a few changes from the INET maintainers, it was merged and subsequently released in INET version 4.3.0. This simulation model is now a part of the INET framework [19].

7.3.2 BGP Conclusion

The resulting model was redesigned and reimplemented with a goal to provide a better simulation experience for its users. Most notably, this new version of the model greatly expands the ways to interact with the simulation by implementing a number of event handlers that can be invoked while the simulation is executing. This enables simulation of new scenarios such as as complete node or interface failures. The model also reacts to the changes of the local routing table. Additionally, the model was extended with complete IPv6 support via the usage of abstract classes, keeping the overall complexity of the code base as low as possible. The BGP decision process was extended and some new route attributes were implemented to allow for more complex routing policies. Moreover, the `NOTIFICATION` message was implemented, and the `UPDATE` message creation process was rewritten, allowing for more realistic message exchanges. Lastly, the configuration of the model was modified to more closely follow the configuration process on Cisco devices.

However, there are currently still some missing features that will have to be addressed before the model is submitted for a merge with the INET code base. INET developers have indicated an interest in these additional features as this model should completely replace the existing one and thus, should be a clear improvement with all features of its predecessor. It, for example, includes the old way of configuration so as not to break the already existing simulations. The serialisation of messages that allows the simulation to output the traffic as `.pcap` files also needs to be extended with the new attributes and message formats.

When these additions to the model are complete, the new model together with an extended version of the 'BGP routing tutorials' from Section 8.3 will be submitted via a pull request to INET. Currently, it is only available on the author's GitHub repository [6].

Chapter 8

Testing

This chapter contains a set of 'routing protocol tutorials' as a means of providing a concrete proof of the validity of the models. This direction was chosen as the resulting text provides additional educational value. The purpose of these tutorials is to have an extended version of them published on the INET tutorial website [22], where they can help networking enthusiasts understand the most important aspects of these specific routing protocols. For this reason, the tutorials may be written a in different style than the rest of this thesis.

8.1 Methodology

The following routing tutorials are divided into smaller sections, where each section focusses on an isolated feature or aspect of the protocol/model. Each of these sections may be composed of multiple scenarios to further demonstrate how the specific feature or aspect behaves under different conditions.

All tutorials are executed on the created simulation models. The models themselves were primarily modelled after their corresponding RFCs, but they were also largely influenced by implementation on Cisco devices. Because of this, each tutorial was also executed with Cisco binary images within the EVE-ng emulator[1]. Both changes to the routers' internal structures and exchanged messages were closely inspected and compared between the simulation and the emulator. Any discrepancies found between the model and the Cisco implementation most often lead to an alteration of the simulation model, because in the author's opinion, it is more practical for the model to more closely reflect the real-life implementation of the protocol rather than its specification.

Each tutorial consists of the following:

- `Reproduction` package, which contains all the necessary simulation files and Cisco configuration files to recreate the featured scenarios in both simulation and Cisco devices. Captured traffic in form of `pcap` files is included.
- Topology diagram, which shows how the different network devices are interconnected, as well as the addressing scheme used. In BGP tutorials, the separation of routers into different autonomous systems is also depicted.
- Accompanying text, which explains the behaviour of the protocol for each included scenario. The text is supported with simulation screenshots and diagrams to assist the reader in understanding the text.

[1] https://www.eve-ng.net/

Router identifiers and names of the messages are set in `monospace` while identifiers of FSM states are set in *italic*.

OMNeT++ in version 6.0 is used to run the simulation, and the `ScenarioManager` module is used to change the simulation during runtime. The EVE-ng emulator with IOS version 15.7(3)M2 is used to perform the reference measurement. Similarly to `ScenarioManager`, embedded event manager can execute applets on IOS routers. All links in the mentioned topologies are 10 Mbps Ethernets.

8.2 EIGRP Tutorials

There are three EIGRP tutorials in this section, each with its own topology:

1. **Network Command** - Explains how the `network` command affects the EIGRP process, what specific actions are triggered and how to deal with some of its aspects.
2. **Establishing Neighbourship and Initial Route Exchange** - Explains how EIGRP routers dynamically discover neighbours and highlights which conditions have to be met in order for two neighbouring EIGRP routers to exchange routing information, and additionally, showcases such exchange.
3. **DUAL Calculation** - Explains how the DUAL recalculates the best path to a destination when changes to the topology occur.

8.2.1 Network Command

The `network` command has two distinctive purposes: enabling dynamic neighbour discovery and injecting networks into the local EIGRP database, the topology table. Let us first explore how this database works and what its purpose is.

EIGRP topology table is data structure created for the purpose of storing EIGRP routes, both locally originated routes and routes received from a neighbour via an `UPDATE`. Multiple routes for a single destination may be known and because of that, they are generally grouped by their destination. The structure contains feasible and unfeasible routes, and these and other route traits are tracked via specific tags. The tags convey information about the feasibility of a route, whether it is a successor route or not, its reported distance, and computed distance. The destination itself tracks information about its feasible distance and the number of successors.

Only the best (successor) routes are advertised to EIGRP peers. Additional rules and constraints regarding the advertisements may be applied (*split-horizon* with *poison-reverse*), which are explained in the DUAL tutorial 8.2.3.

The `network` command is made up of two parts: the `IPAddress` prefix and the `Wildcard` mask, and its main purpose is to match with the addresses set on the router interfaces. The `Wildcard` part of the command defines which bits of the prefix portion must match exactly with the IP address and which can be variable. If any of the addresses of an interface match the `network` command, such addresses and interfaces are processed further, as explained in the following paragraph. In general, the `Wildcard` mask can be viewed as an inverted version of the normal netmask. Every address on any router interface is tested against the prefix and wildcard mask of every `network` command. If the prefix, its wildcard mask and the tested IP address are written bitwise, `0` in the mask means that the bit on the same index in the prefix has to match the bit on the same index in the tested address, while

Network Command	Example Matches	Notes
<Network> <IPAddress>10.0.0.1</IPAddress> <Wildcard>0.0.0.0</Wildcard> </Network>	10.0.0.1	Matches only IP address 10.0.0.1
<Network> <IPAddress>10.0.0.0</IPAddress> <Wildcard>0.0.255.255</Wildcard> </Network>	10.0.0.1 10.0.1.10 10.0.9.100	Matches the whole subnet of 65,025 addresses
<Network> <IPAddress>0.0.0.0</IPAddress> <Wildcard>255.255.255.255</Wildcard> </Network>	10.0.0.1 192.168.1.10 172.24.9.3	Matches any address

Table 8.1: EIGRP Tutorial 1 - Examples of how the different EIGRP `network` commands match IP addresses. The netmasks of the addresses are always ignored.

the bit value of `1` means that the bit on the same index of the address can be variable and does not have to match the prefix. The result of testing an address against a `network` command can be either positive, as the address matches the necessary bits with the prefix, or negative, if the address does not match the necessary bits. Examples of this behaviour are shown in Table 8.1.

As mentioned above, a single `network` command can match multiple addresses on multiple interfaces. When an IP address of an interface matches any of the `network` commands, the EIGRP process executes two important actions:

1. It enables the dynamic neighbour discovery process on that interface. Even if multiple addresses on a single interface are matched, only a single discovery process is started with the first matched IP address used as a source for its messages.

2. IP address of that interface gets added to the EIGRP topology table. If multiple IP addresses on a single interface are matched, EIGRP creates a topology table entry for each unique network.

This enables the EIGRP process to establish an adjacency with other EIGRP enabled routers on such interfaces and it also populates the local EIGRP topology table with locally originated routes, which may be advertised to EIGRP neighbours.

Because the `network` command can be as general or as specific as necessary and multiple commands can be used at once, EIGRP enabled interfaces can be precisely specified. However, there is a downside to the `network` command. As the aforementioned actions are tied together, an insertion of a route into EIGRP topology table for the purpose of advertising it to EIGRP neighbours triggers dynamic neighbour discovery process on its interface automatically. If such an interface is a LAN interface, where no EIGRP neighbour will ever be found, the router will continuously waste resources with its `Hello` messages. Additionally, such behaviour may be even dangerous, as a malicious device may be set up on such connection, and if there is a lack of authentication, fake and malicious routes may be distributed throughout the EIGRP domain. For these two reasons, it is essentially good

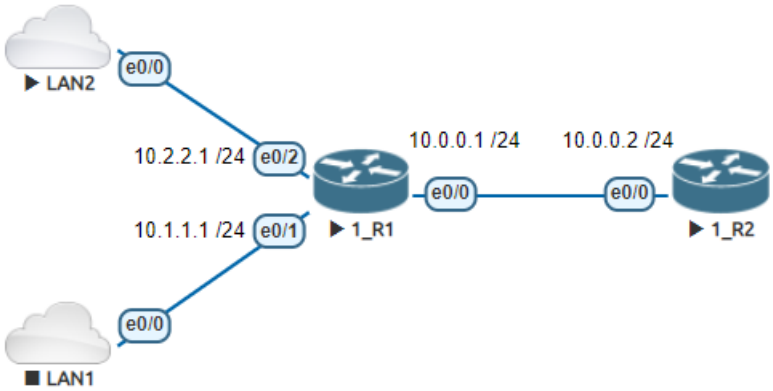


Figure 8.1: EIGRP Topology 1 - `network` command.

practice to disable the dynamic neighbour discovery process on interfaces which are not toward any neighbours. This is accomplished by configuring such interface as a `passive` interface. Its network can still be added to the EIGRP topology table, but no EIGRP neighbourships can be established on the interface.

Topology

To showcase this behaviour, consider the topology with the addressing scheme as depicted in Figure 8.1. The focus is only on router `R1`, as we explore the behaviour of the `network` command.

1 - Populating EIGRP Databases and Dynamic Neighbour Discovery

Let us now explore what `network` commands can be entered to insert both `LAN`s' networks into the EIGRP topology table and to enable the dynamic discovery of neighbours on interface towards router `R2`.

As explained above, the `network` command is very flexible. To enable EIGRP on router `R1`, we can use any of the combination of commands as shown in Listings 8.1, 8.2 and 8.3.

```
<Network>
  <IPAddress>10.2.2.1</IPAddress>
  <Wildcard>0.0.0.0</Wildcard>
</Network>
<Network>
  <IPAddress>10.1.1.1</IPAddress>
  <Wildcard>0.0.0.0</Wildcard>
</Network>
<Network>
  <IPAddress>10.0.0.1</IPAddress>
  <Wildcard>0.0.0.0</Wildcard>
</Network>
```

Listing 8.1: EIGRP Tutorial 1 - the most specific `network` commands.

In Listing 8.1, each interface is matched with a specific command. If any new interfaces are enabled on router `R1`, they are not added to the EIGRP process.

```
<Network>
  <IPAddress>10.0.0.0</IPAddress>
  <Wildcard>0.255.255.255</Wildcard>
</Network>
```

Listing 8.2: EIGRP Tutorial 1 - a `network` command matching IP addresses with the first byte equal to `10`.

In Listing 8.2, all interfaces are matched with a single command. If any new interfaces are enabled on router `R1`, they are added to the EIGRP process only if their IP address matches the configured prefix (i.e., first byte must be equal to `10`).

```
<Network>
  <IPAddress>0.0.0.0</IPAddress>
  <Wildcard>255.255.255.255</Wildcard>
</Network>
```

Listing 8.3: EIGRP Tutorial 1 - most ambiguous `network` command.

In Listing 8.3, all interfaces are matched with a single command. If any new interfaces are enabled on router `R1`, they are always added to the EIGRP process when they are assigned with any IP address.

The resulting topology table is shown in Figure 8.2. It contains all directly connected networks which can be advertised to EIGRP neighbours.

```
[0] P 10.0.0.0/24 is successor FD:281600 via Connected (281600/0), IF:eth0(101)
[1] P 10.2.2.0/24 is successor FD:281600 via Connected (281600/0), IF:eth2(103)
[2] P 10.1.1.0/24 is successor FD:281600 via Connected (281600/0), IF:eth1(102)
```

Figure 8.2: EIGRP Tutorial 1 - Resulting topology table of router `R1`.

However, EIGRP is now actively discovering neighbour on links towards both `LAN`s, which is not a desirable behaviour, as explain above. To this end, a `PassiveInterface` command should be used as shown in Listing 8.4.

```
⋮
<PassiveInterface>eth2</PassiveInterface>
<PassiveInterface>eth1</PassiveInterface>
```

Listing 8.4: EIGRP Tutorial 1 - Usage of `PassiveInterface` to prevent unnecessary `HELLO` messages.

In summary, all connected networks are added to the EIGRP topology table, which enables them to be advertised via EIGRP, but adjacency may be established only on interface `eth0` as the other interfaces are configured as passive.

8.2.2 Establishing Neighbourship and Initial Route Exchange

EIGRP utilizes RTP protocol to reliably transport certain messages. RTP, similar to TCP, uses two fields: `sequence` and `acknowledge` to precisely identify outgoing messages and verify their receipt, respectively. Acknowledgement can be piggy-backed into any unicast message or sent on its own. No prior RTP-specific-handshake is necessary.

EIGRP uses `HELLO` messages to dynamically discover neighbours. At this point, such message serves a double purpose: it annouces a presence of EIGRP enabled router a on

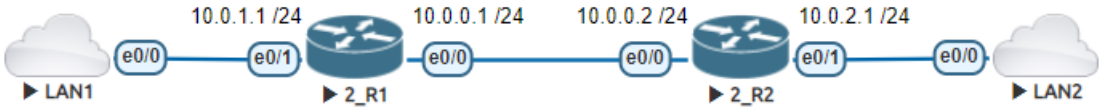


Figure 8.3: EIGRP Topology 2 - Establishing Neighbourship and Initial Route Exchange.

link with a certain subnet and secondly, it conveys sender's EIGRP parameters, such as its *autonomous system identifier*, *K-Values*, *STUB* settings, and *Hold time*.

Some properties and parameters like *subnet*, *autonomous system identifier*, and *K-Values* must match between the routers if they are to exchange any routing information, while others like *STUB settings* and *Hold time* just alter neighbour's behaviour towards the sending router. Routers may exchange routing information only if the neighbourship is established. Both multicast and unicast reachability is verified before the two EIGRP routers become neighbours.

When routers exchange routes, each unique destination has its route with the least metric selected as the successor route and such route may be installed in the routing table. Successor's computed distance is saved as a feasible distance, which separates known routes for each destination into feasible and unfeasible by the feasibility condition. Feasible distance is not increased until DUAL recalculation is necessary. If all feasible routes are for whatever reason unavailable, DUAL tries to find a new feasible successor. This usually entails a reset of feasible distance.

After the neighbourship is established between the EIGRP routers, `HELLO` messages are still sent periodically to let the receiving router know that the sender is still active. The receipt of the `HELLO` message refreshes `Hold Timer` for the specific neighbour. If it were to reach zero, the neighbour would be considered unavailable, and all the routes received from that neighbour would be deleted.

Topology

The topology to showcase this behaviour, as well as the addressing scheme, is shown in Figure 8.3. Each router is configured to dynamically discover neighbours on the link toward the other router, and the network towards the `LAN` is properly matched with a `network` command so that it can be advertised to the other router and paired with `PassiveInterface` command.

1 - Initial Route Exchange

This tutorial focusses on the neighbour discovery process and on the initial synchronisation of routing databases.

When EIGRP starts the dynamic neighbour discovery process, it starts to periodically send `HELLO` messages on a multicast address `224.0.0.10`. The interval between consecutive messages is determined by *Hello Interval*, implicitly set to `5` seconds.

When receiving a foreign `HELLO` message from an unknown neighbour, its parameters are checked, and if they are valid, a new entry in the local `Neighbor` table is created for this specific neighbour. The state of this entry is set to *Pending* and an empty `UPDATE` message with the `INIT` flag is sent to that specific neighbour. This serves to verify the unicast reachability of the neighbour. When the forein `UPDATE` message with the `INIT` flag is received, it is acknowledged. When an acknowledgement of sent `INIT UPDATE` message is

Event#	Time	Relevant Hops	Name	TxUpdate? / Source	Length / Destination	Info / Type	Length
#7	0.417'021'998'437	R1 ⟶ R2	EIGRP_HELLO_MSG	10.0.0.1	224.0.0.10	EIGRP	78 B
#25	0.417'084'448'437	R2 ⟶ R1	EIGRP_HELLO_MSG	10.0.0.2	224.0.0.10	EIGRP	78 B
#45	0.417'146'898'437	R1 ⟶ R2	EIGRP_HELLO_MSG	10.0.0.1	224.0.0.10	EIGRP	78 B
#47	0.417'156'448'437	R2 ⟶ R1	arpREQ	0A-AA-00-00-00-03	FF-FF-FF-FF-FF-FF	ARP REQUEST	72 B
#63	0.417'218'898'437	R1 ⟶ R2	arpREQ	0A-AA-00-00-00-01	FF-FF-FF-FF-FF-FF	ARP REQUEST	72 B
#71	0.417'276'548'437	R2 ⟶ R1	EIGRP_UPDATE_MSG	10.0.0.2	10.0.0.1	EIGRP	72 B
#73	0.417'286'098'437	R1 ⟶ R2	EIGRP_UPDATE_MSG	10.0.0.1	10.0.0.2	EIGRP	72 B
#88	0.417'343'748'437	R2 ⟶ R1	arpREPLY	0A-AA-00-00-00-03	0A-AA-00-00-00-01	ARP REPLY	72 B
#99	0.417'353'298'437	R1 ⟶ R2	arpREPLY	0A-AA-00-00-00-01	0A-AA-00-00-00-03	ARP REPLY	72 B
#106	0.417'410'948'437	R2 ⟶ R1	EIGRP_ACK_MSG	10.0.0.2	10.0.0.1	EIGRP	72 B
#109	0.417'420'498'437	R1 ⟶ R2	EIGRP_ACK_MSG	10.0.0.1	10.0.0.2	EIGRP	72 B
#136	0.417'478'148'437	R2 ⟶ R1	EIGRP_UPDATE_MSG	10.0.0.2	10.0.0.1	EIGRP	110 B
#137	0.417'487'698'437	R1 ⟶ R2	EIGRP_UPDATE_MSG	10.0.0.1	10.0.0.2	EIGRP	110 B
#164	0.417'575'748'437	R2 ⟶ R1	EIGRP_ACK_MSG	10.0.0.2	10.0.0.1	EIGRP	72 B
#165	0.417'585'298'437	R1 ⟶ R2	EIGRP_ACK_MSG	10.0.0.1	10.0.0.2	EIGRP	72 B
#190	0.417'642'948'437	R2 ⟶ R1	EIGRP_UPDATE_MSG	10.0.0.2	224.0.0.10	EIGRP	110 B
#191	0.417'652'498'437	R1 ⟶ R2	EIGRP_UPDATE_MSG	10.0.0.1	224.0.0.10	EIGRP	110 B
#216	0.417'740'548'437	R2 ⟶ R1	EIGRP_ACK_MSG	10.0.0.2	10.0.0.1	EIGRP	72 B
#217	0.417'750'098'437	R1 ⟶ R2	EIGRP_ACK_MSG	10.0.0.1	10.0.0.2	EIGRP	72 B

Figure 8.4: EIGRP Tutorial 2 - Exchanged messages between routers `R1` and `R2`.

```
[0] P 10.0.0.0/24 is successor FD:281600 via Connected (281600/0), IF:eth0(101)
[1] P 10.0.2.0/24 is successor FD:307200 via 10.0.0.2 (307200/281600), IF:eth0(101)
[2] P 10.0.1.0/24 is successor FD:281600 via Connected (281600/0), IF:eth1(102)

[0] P 10.0.0.0/24 is successor FD:281600 via Connected (281600/0), IF:eth0(101)
[1] P 10.0.1.0/24 is successor FD:307200 via 10.0.0.1 (307200/281600), IF:eth0(101)
[2] P 10.0.2.0/24 is successor FD:281600 via Connected (281600/0), IF:eth1(102)
```

Figure 8.5: EIGRP Tutorial 2 - Final states of EIGRP topology tables of routers `R1` and `R2`.

received, the state of the sending neighbour is locally set to *Up* state. When this happens, the neighbourship is considered established, and the initial exchange of routing information can commence. It takes the form of even multiple `UPDATE` messages, each of which may carry multiple routes. The last sent `UPDATE` message contains a `EOT` flag to signify that all routes have been sent. In this specific case, routers exchange empty `UPDATE` message with the `INIT` flag, then they exchange another set of `UPDATE` messages with `EOT` flag routing information about the directly connected `LAN` and finally, as they both apply the `split-horizon` with `poison-reverse` rule, also mentioned later on in the DUAL tutorial 8.2.3. As a result of this mechanism, each router sent `UPDATE` multicast message to advertise destinations that have a successor on that interface as unavailable.

The general sequence diagram of this exchange is shown in Appendix B. The traffic between the router is shown in Figure 8.4 and the final state of the EIGRP topology tables of both routers is shown in Figure 8.5. Router `R1` learnt to reach destination `10.0.2.0/24`, while router `R2` learnt the same about destination `10.0.1.0/24`.

8.2.3 DUAL Calculation

DUAL is an algorithm used by EIGRP to recalculate the best loopless route to a destination. DUAL is generally very conservative, if a route is deemed as loopless by DUAL, it is always loopless, however, a loopless route may be rejected under certain circumstances as it could potentially contain a loop from DUAL perspective.

As mentioned in the previous tutorial, DUAL recalculation is triggered whenever no feasible successor for a route is available. The goal of DUAL is to find the new successor

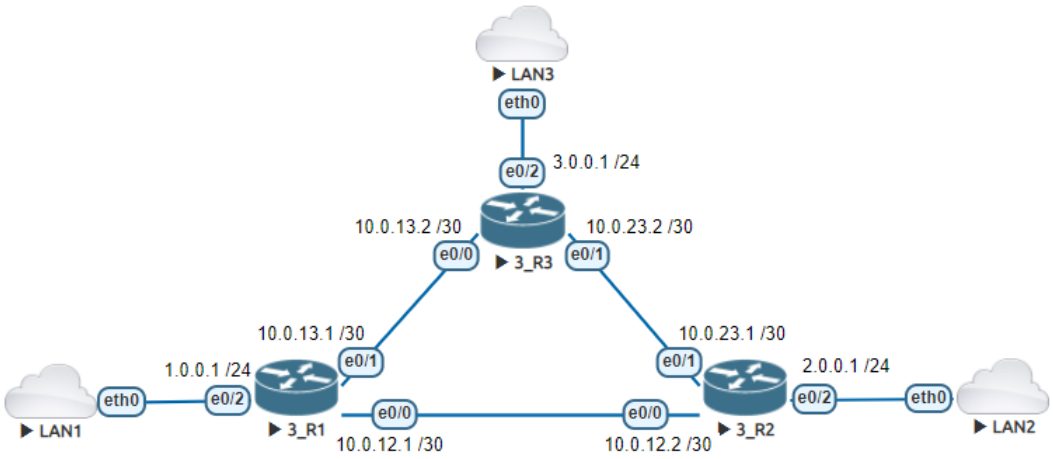


Figure 8.6: EIGRP Topology 3 - DUAL Calculation.

route with the least distance, this usually requires a reset of feasible distance to a new value.

DUAL FSM in instantiated for each destination individually, states of this FSM for different destinations do not affect each other. A destination can be either in the *passive* state, when the feasible successor is known, or in the *active* state, while the DUAL is actively trying to find a new successor.

DUAL uses the `QUERY` message to convey a transition of a specific destination to an *Active* state to its neighbours. When a `QUERY` message is received from a neighbouring router for a destination, the recipient removes route through the sender to such destination from the EIGRP topology table if present, and if it still has a feasible successor, a `REPLY` carrying the local metric for this destination is sent. Otherwise, if no feasible successor for this destination is known, the destination transitions to *Active* state and `QUERY` messages are sent to neighbours. When a router has no suitable neighbours to send `QUERY` to, or all `REPLY` messages are received, a `REPLY` is sent back to the sender of `QUERY`, if the calculation was triggered by receiving such `QUERY`. Destination is either newly available through some neighbour or, if not, removed from the EIGRP topology table completely. `UPDATE` messages are sent to all neighbours whenever the distance to a destination changes.

As an additional means of routing loop prevention, EIGRP uses the rule *split-horizon with poison-reverse*. Routes sourced from a neighbour are never advertised back to the same neighbour, and destinations are advertised as unreachable towards their current successors.

Topology

To demonstrate the aforementioned behaviour, the topology consists of three routers and three `LAN`s. All routers are configured to advertise all connected networks. The topology, as well as the addressing scheme are depicted in Figure 8.6.

1 - Recalculation of Routes on R1

The topology is initially converged. To observe how DUAL recalculates the new best path to the destination, let us disconnect the link between router `R1` and `R2` and observe, what messages are sent by router `R1` to `R3` and what changes are made to the `R1` EIGRP topology table. The initial EIGRP topology table of router `R1` is shown in Figure 8.7.

```
[0] P 10.0.12.0/30 is successor FD:281600 via Connected (281600/0), IF:eth0(101)
[1] P 2.0.0.0/24 FD:307200 via 10.0.13.2 (332800/307200), IF:eth1(102)
[2] P 2.0.0.0/24 is successor FD:307200 via 10.0.12.2 (307200/281600), IF:eth0(101)
[3] P 3.0.0.0/24 is successor FD:307200 via 10.0.13.2 (307200/281600), IF:eth1(102)
[4] P 3.0.0.0/24 FD:307200 via 10.0.12.2 (332800/307200), IF:eth0(101)
[5] P 10.0.23.0/30 is successor FD:307200 via 10.0.13.2 (307200/281600), IF:eth1(102)
[6] P 10.0.23.0/30 is successor FD:307200 via 10.0.12.2 (307200/281600), IF:eth0(101)
[7] P 1.0.0.0/24 is successor FD:281600 via Connected (281600/0), IF:eth2(103)
[8] P 10.0.13.0/30 is successor FD:281600 via Connected (281600/0), IF:eth1(102)
```

Figure 8.7: EIGRP Tutorial 3 - The initial state of EIGRP topology tables of router `R1` when the link between router `R1` and `R2` is still connected.

#	Time	Relevant Hops	Name	ID / Source	Destination	Type	Length
1	50.000	R1 --> R3	EIGRP_QUERY_MSG	10.0.13.1	224.0.0.10	EIGRP	198 B
2	50.000 158 450	R3 --> R1	EIGRP_ACK_MSG	10.0.13.2	10.0.13.1	EIGRP	72 B
3	50.000 225 650	R3 --> R1	EIGRP_REPLY_MSG	10.0.13.2	10.0.13.1	EIGRP	198 B
4	50.000 384 100	R1 --> R3	EIGRP_ACK_MSG	10.0.13.1	10.0.13.2	EIGRP	72 B
5	50.000 441 750	R3 --> R1	EIGRP_UPDATE_MSG	10.0.13.2	224.0.0.10	EIGRP	110 B
6	50.000 451 300	R1 --> R3	EIGRP_UPDATE_MSG	10.0.13.1	224.0.0.10	EIGRP	154 B
7	50.000 574 550	R3 --> R1	EIGRP_ACK_MSG	10.0.13.2	10.0.13.1	EIGRP	72 B
8	50.000 584 100	R1 --> R3	EIGRP_ACK_MSG	10.0.13.1	10.0.13.2	EIGRP	72 B
9	50.000 641 750	R3 --> R1	EIGRP_QUERY_MSG	10.0.13.2	224.0.0.10	EIGRP	110 B
10	50.000 729 800	R1 --> R3	EIGRP_ACK_MSG	10.0.13.1	10.0.13.2	EIGRP	72 B
11	50.000 797	R1 --> R3	EIGRP_REPLY_MSG	10.0.13.1	10.0.13.2	EIGRP	110 B
12	50.000 885 050	R3 --> R1	EIGRP_ACK_MSG	10.0.13.2	10.0.13.1	EIGRP	72 B

Figure 8.8: EIGRP Tutorial 3 - Exchanged traffic between routers `R1` and `R3` after the link between routers `R1` and `R2` was disconnected.

When the link is disconnected, router `R1` originates a `QUERY` message advertising *Active* states for routes `10.0.12.0/30, 2.0.0.0/24`. Router `R3` responds with a `REPLY` that contains `R3`'s information about given routes. This message contains a metric for route `2.0.0.0/24`. It also advertises `10.0.12.0/30` as unreachable because it has already received `QUERY` from router `R2`. In addition, router `R3` advertises route `10.0.12.0/30` as unreachable to `R1`, as its distance on router `R3` changed to infinity. As router `R3` became the successor for routes `2.0.0.0/24` and `10.0.23.0/30`, `R1` applies the *poison-reverse* rule and advertises these destinations as not available to `R3`. Finally, router `R3` is advertising route `10.0.12.0/30` in the *Active* state via a `QUERY`. This message is caused by the arrival of a `QUERY` from router `R2`. Router `R1` responds with a `REPLY` with infinite metric as no such destination is present in the EIGRP topology table and no additional neighbours exits. This captured traffic is shown in Figure 8.8.

The resulting EIGRP topology table of router `R1` is shown in Figure 8.9. Destination `10.0.12.0/30` was deleted, a route to destination `10.0.2.0/24` was recalculated.

```
[0] P 2.0.0.0/24 is successor FD:332800 via 10.0.13.2 (332800/307200), IF:eth1(102)
[1] P 3.0.0.0/24 is successor FD:307200 via 10.0.13.2 (307200/281600), IF:eth1(102)
[2] P 10.0.23.0/30 is successor FD:307200 via 10.0.13.2 (307200/281600), IF:eth1(102)
[3] P 1.0.0.0/24 is successor FD:281600 via Connected (281600/0), IF:eth2(103)
[4] P 10.0.13.0/30 is successor FD:281600 via Connected (281600/0), IF:eth1(102)
```

Figure 8.9: EIGRP Tutorial 3 - The initial state of EIGRP topology tables of router `R1` when the link between router `R1` and `R2` is still connected.

In the current state of the model, DUAL shows a slight deviation from the expected behaviour, as the destination `10.0.23.0/30` on route `R1` also transitioned into the `Active` state even when a feasible successor was present when the link was disconnected. This only happens when there are multiple successors for a route and only resulted in an unnecessary inclusion of destination `10.0.23.0/30` in the initial `QUERY` message.

8.3 BGP Tutorials

There are four BGP tutorials in this section, each with its own topology:

1. **Establishing Peering** - Explains how BGP leverages TCP, how the BGP peer FSM transitions between different states and even explores scenarios where the BGP process is misconfigured.
2. **Network Command** - Explains how the `network` command injects routes into BGP databases and showcases conditions that have to be met in order for the route to be advertised to BGP peers.
3. **Exchanging Updates** - Explains how prefixes are advertised and why they can be eventually withdrawn. This tutorial also showcases how BGP can use multiple different address families at once.
4. **Attributes** - Explains the purpose of the most important BGP attributes and showcases how they are advertised or changed.

8.3.1 Establishing Peering

The main goal of this tutorial is to showcase the process of establishing peering between BGP-enabled routers and to also demonstrate how BGP and the underlying TCP deal with configuration errors.

Topology

To showcase how BGP is establishing peering with its neighbours, we need just two BGP-enabled routers and a single link connecting them, on which we can observe exchanged messages. This topology, together with the address scheme, AS, and interface identifiers, is shown in Figure 8.10.

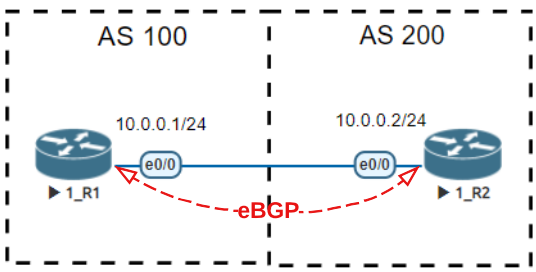


Figure 8.10: BGP Topology 1 - Establishing Peering.

1 - Valid Configuration

In this first case, the both BGP routers have a valid configuration, in regards to information shown in Figure 8.10. If both routers are started at the same time in the simulation, the resulting traffic is exactly mirrored by both nodes. Even though communication in such case is still technically valid, its irreproducible on real devices and thus the start of router `R1` is delayed by `0.5` seconds. A sequence diagram of the traffic between the routers is shown in Appendix C and exact captured traffic is shown in Figure 8.11.

The first thing we can observe is the ARP/NDP traffic as both routers are trying to find MAC address of the neighbour. Information about peer's specific MAC address is necessary, as all messages are sent as unicast. This is in contrast to some IGP protocols like EIGRP or OSPF, which use multicast for neighbour discovery and thus do not require information about each other's specific MAC addresses. TCP handshake follows a successful matching of IP address to MAC address. It takes form of three consecutive segments: `SYN`, `SYN+ACK`, `ACK`. The initiating party, `R1` in this case, sends a TCP segment containing its *Sequence Number* together with a `SYN` flag to initiate the handshake. `R1`'s BGP FSM for router `R2` transitions from *IDLE* state to *CONNECT* state.

The receiving party, `R2`, responds with a segments containing its own *Sequence Number* together with the received *Sequence Number* increased by one in the *Acknowledgement Number* field. This message is accompanied with two flags: `SYN` and `ACK`. `R2`'s BGP FSM for router `R1` stays in the *ACTIVE* state until the handshake is completed.

As a last step, the receiving party of this segment, `R1`, acknowledges this message with its own message. It contains the *Sequence Number* as received in the previous message in the *Acknowledgement Number* field. To reflect the fact that the previous message was received, the generated message also contains the received *Sequence Number* increased by one in the *Acknowledgement Number* field. This message contains flag `ACK`. `R1`'s BGP FSM for router `R2` can now transition from *CONNECT* state to *OPENSENT* state.

When this last message is sent by `R1`, the reliable TCP transport channel is regarded as open and functional by any application layer protocol, BGP in this case. `R1`'s BGP FSM for router `R2` transitions from *CONNECT* state to *OPENSENT* state. On the other hand, `R2` needs to wait for the last `ACK` message. When this happens, `R2`'s BGP FSM for router `R1` can now also transition from *ACTIVE* state to *OPENSENT* state.

What essentially happened was that each TCP participating party informed the other party of the randomly generated *Sequence Number*, that will be used as the initial value of counting transmitted application layer bytes. Each reliably transmitted message with new application layer data will require a receipt of the TCP segment containing *Acknowledgement Number* increased by the number of transmitted application layer bytes to be considered delivered.

As for the BGP communication, routers exchange `OPEN` messages. Each checks the received `OPEN` message for expected values regarding AS identifier. Since they are correct, each router generates `KEEPALIVE` message as acknowledgement. When this message is received, the BGP FSM for the peer transitions to the *ESTABLISHED* state. BGP routers are now able to advertise routes though the means of `UPDATE` messages.

2 - Invalid IP Configuration

One common way of BGP misconfiguration is to assign a BGP process with neighbour with an incorrect IP address. In this case, BGP process of router `R2` was configured with a non-

Event#	Time	Relevant Hops	Name	TxUpdate? / Source	Length / Destination	Info /	Type	Length
#6	0.000	R2 ⟶ R1	arpREQ	0A-AA-00-00-00-02	FF-FF-FF-FF-FF-FF	ARP	REQUEST	72 B
#12	0.000'005'810	R1 ⟶ R2	arpREPLY	0A-AA-00-00-00-01	0A-AA-00-00-00-02	ARP	REPLY	72 B
#19	0.000'011'620	R2 ⟶ R1	SYN	10.0.0.2:50000	10.0.0.1:179	TCP		72 B
#28	0.000'017'430	R1 ⟶ R2	RST+ACK	10.0.0.1:179	10.0.0.2:50000	TCP		72 B
#41	0.254'175'521'177	R2 ⟶ R1	NSpacket	<unspec>	ff02::1:ff00:2	ICMPv6		90 B
#50	0.257'383'685'955	R1 ⟶ R2	NSpacket	<unspec>	ff02::1:ff00:1	ICMPv6		90 B
#62	0.500	R1 ⟶ R2	SYN	10.0.0.1:50000	10.0.0.2:179	TCP		72 B
#70	0.500'005'810	R2 ⟶ R1	SYN+ACK	10.0.0.2:179	10.0.0.1:50000	TCP		72 B
#82	0.500'011'620	R1 ⟶ R2	TcpAck	10.0.0.1:50000	10.0.0.2:179	TCP		72 B
#99	0.500'017'430	R2 ⟶ R1	tcpseg(l=37)	10.0.0.2:179	10.0.0.1:50000	TCP		103 B
#100	0.500'018'340	R1 ⟶ R2	tcpseg(l=37)	10.0.0.1:50000	10.0.0.2:179	TCP		103 B
#121	0.500'026'630	R2 ⟶ R1	TcpAck	10.0.0.2:179	10.0.0.1:50000	TCP		72 B
#122	0.500'027'540	R1 ⟶ R2	TcpAck	10.0.0.1:50000	10.0.0.2:179	TCP		72 B
#139	0.500'033'350	R2 ⟶ R1	tcpseg(l=19)	10.0.0.2:179	10.0.0.1:50000	TCP		85 B
#140	0.500'034'260	R1 ⟶ R2	tcpseg(l=19)	10.0.0.1:50000	10.0.0.2:179	TCP		85 B
#161	0.500'041'110	R2 ⟶ R1	TcpAck	10.0.0.2:179	10.0.0.1:50000	TCP		72 B
#162	0.500'042'020	R1 ⟶ R2	TcpAck	10.0.0.1:50000	10.0.0.2:179	TCP		72 B
#179	0.500'047'830	R2 ⟶ R1	tcpseg(l=23)	10.0.0.2:179	10.0.0.1:50000	TCP		89 B
#180	0.500'048'740	R1 ⟶ R2	tcpseg(l=23)	10.0.0.1:50000	10.0.0.2:179	TCP		89 B
#199	0.500'055'910	R2 ⟶ R1	TcpAck	10.0.0.2:179	10.0.0.1:50000	TCP		72 B
#200	0.500'056'820	R1 ⟶ R2	TcpAck	10.0.0.1:50000	10.0.0.2:179	TCP		72 B

Figure 8.11: BGP Tutorial 1 - Exchanged messages between routers `R1` and `R2`.

existing BGP neighbour `10.0.0.10` while the BGP process of `R1` was left in an unchanged configuration with neighbour `10.0.0.2`. All other aspects were left unchanged.

In the case of Cisco implementation, the communication in this case fails on the three-way handshake during the initialisation of reliable transport channel. While router `R1` initiates the three-way handshake with its `SYN` segment, router `R2` responds with TCP segment containing `RST` and `ACK`. It signalizes that it received the message with `ACK` flag and `Acknowledgement Number`, but since it is expecting a different source address (i.e., `10.0.0.1`), it is resetting the connection with `RST` flag. These two messages are repeating every roughly 10 seconds. The BGP FSM state for neighbour `R2` oscillates between *ACTIVE* state, as it tries to connect, and *IDLE* state as `RST` segments are received. `R1`, on the other hand, is not generating `SYN` segments to establish connection with neighbour `10.0.0.10` is it lacking information about L2 MAC-address of the destination. Even if ARP table entry with `R1`'s MAC address is provided as a `10.0.0.10`'s MAC address, `R1` never processes such a message and routes it back onto the same link towards `R2` instead. TTL of such message is quickly exceeded.

However, the communication in OMNeT++ is slightly different. Because of the limits of INET TCP model, source IP address of the incoming connection (i.e., received `SYN`) cannot be verified for matching the locally configured peer and thus the connection is always accepted. After context of the non-initiating node switches to the BGP process, it verifies if the IP address on the other side of the the TCP connection corresponds with the locally configures one and if it does not match, then it resets the listening socket. Since the implementation of the BGP peer FSM in the simulation is slightly different then the one implemented on Cisco devices, the state of the peering oscillates between *IDLE*, *CONNECT* and *ACTIVE* states. This discrepancy of the implementation is explicitly allowed by the RFC 4271 [29] (Section 8, BGP Finite State Machine).

In either scenario, BGP peer FSM does not reach the *ESTABLISHED* state and thus no routing information is exchanged.

3 - Invalid AS Configuration

Another common instance of BGP misconfiguration is assigning a neighbour with incorrect AS identifier. What could essentially happen is that one neighbour could view the peering

as iBGP while the counterparty would view the peering as eBGP. In such case, the BGP would not function properly as the two types of peering are far from being equivalent. To this end, `OPEN` message contains a field to advertise the local AS number, which the recipient can verify for the expected locally configured AS value for that specific peer. If the received AS identifier is unexpected, `NOTIFICATION` message carrying *Bad Peer AS* error code is generated. The peering, as well as the TCP transport channel, are subsequently reset. As TCP closes, it generates `FIN, ACK` segments to indicate that no additional application data will be sent from their side (`FIN`). As the connection is reset, the routers will eventually try to establish the peering again. BGP peer FSM on both sides goes from the initial *IDLE* state though most of the state to eventually reach *OPENSENT* or *OPENCONFIRM* state before going back to the *IDLE* state.

Since *ESTABLISHED* state is not reached by either party, no routing information is exchanged.

8.3.2 Network Command

The `network` command is currently the only implemented way to directly inject routes into the local BGP process. Such injected routes may be advertised to configured BGP peers. Network command, unlike other techniques mentioned in the following paragraph, provides precise control over the advertised routes as each network command represents exactly one BGP route.

Another theoretical and not yet implemented way of locally injecting routes into the BGP process may be accomplished with some flavour or form of route redistribution. In contrast to the `network` command, redistribution usually involves a vector of routes that is created from the routing table and only includes routes from a specified source. The source may specify only directly connected routes, statically configured routes, or even routes learnt through a specific IGP protocol. This vector of routes is inserted into the local BGP route database (i.e., BGP table or its equivalent), and individual routes may be advertised to BGP peers.

In order for any prefix entered via the network command to be advertised to locally configured BGP peers, it has to match the following criteria:

1. prefix and its mask must exactly match any entry in the local routing table;
2. next-hop of the previously mentioned entry must be resolvable (the ability to be resolved of individual routing table entries is not enforced in the simulation).

If the next-hop found in the routing table through the exact match implies that the route is directly connected (i.e., is equal to `0.0.0.0`), it is replaced by a local BGP peering source (i.e., IP address of the interface which the BGP uses a source for the connection for each peer). Otherwise, the next-hop is copied and used as route's default next-hop attribute.

The network command is composed of two parts: `address` and `mask`, as shown in Listing 8.5. Together, they represent a specific destination network, and from the context of BGP, such specified network should be advertised to configured peers.

```
<Network address="1.1.1.1" mask="255.255.255.255"/>
```

Listing 8.5: BGP Tutorial 2 - Example of `network` command.

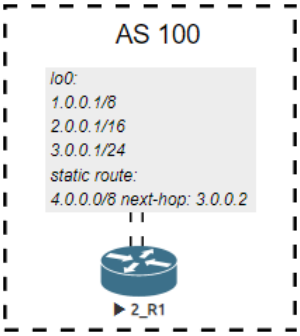


Figure 8.12: BGP Topology 2 - `network` command.

Topology

To showcase the aforementioned behaviour, we need just a single router `R1`. This router is configured according to the Figure 8.12: it has one loopback (logical/virtual) interface `lo0` that is assigned with the following IPv4 addresses:

- `1.0.0.1/8`
- `2.0.0.1/16`
- `3.0.0.1/24`

In addition, a static route for the prefix `4.0.0.0/8` is configured on the router as well.

1 - Populating BGP Databases

The BGP process of router `R1` is configured with the four network statements as shown in Listing 8.6. Let us observe, how these statements changed the BGP route databases.

```
⋮
<Address-family id="Ipv4">
  <Network address="1.0.0.0" mask="255.0.0.0"/>
  <Network address="2.0.0.0" mask="255.255.0.0"/>
  <Network address="3.0.0.0" mask="255.255.255.0"/>
  <Network address="4.0.0.0" mask="255.255.0.0"/>
  ⋮
```

Listing 8.6: BGP Tutorial 2 - Used `network` command.

Routing Table

Routing table stores information about reachable networks, their prefixes, next-hop addresses, output interfaces, and their metric. It initially contains all destinations available through any local interface and can be extended with static (locally configured) or dynamic (learned via some dynamic routing protocol) routes. Routing table is not a BGP-specific structure and is accessable router-wide.

When the simulation is initialised, the networks on the directly connected links are installed in the routing table. They can be identified by the `C` (as for Connected) code at the beginning of the entries. The static route is also installed by the BGP module in the routing table, and is identified by the `S` code. The general state of the routing table is shown in Figure 8.13.

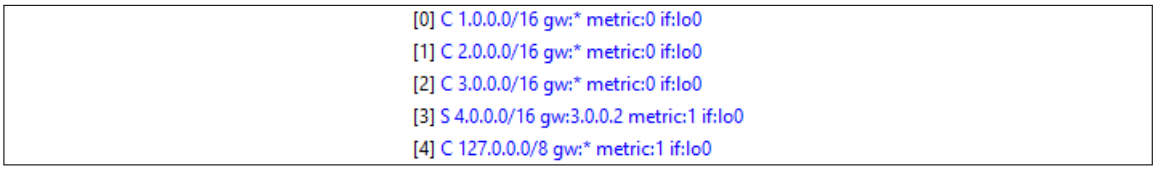


Figure 8.13: BGP Tutorial 2 - state of routing table.

Advertise List

Advertise list stores all prefixes configured by the `network` command. There are only two rules that each route must obey in order to be placed in this list:

a) entry must be a valid IPv4/IPv6 address; and

b) entry has to be unique.

The prefixes contained in this list do not necessarily have to be advertised to BGP peers. This structure only reflects the content of the configuration `.xml` file, so that when the BGP process for whatever reason needs to access the configured prefixes, it does not have to parse the input file and it uses this list instead.

Since all prefixes entered by the network command obey these rules, they are all placed in this structure. Advertise list is shown in Figure 8.14.

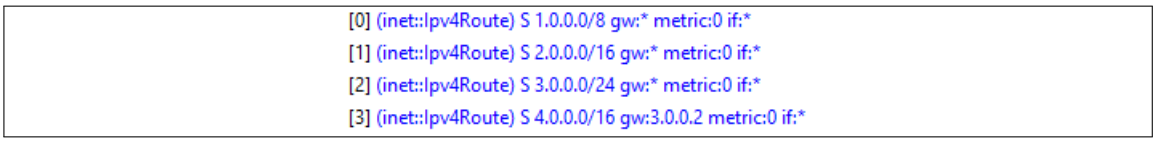


Figure 8.14: BGP Tutorial 2 - state of BGP advertise list.

BGP Table

BGP table stores all routes received through BGP and all routes from `Advertise List` that exactly match any entry in the routing table. A presence of a route in the BGP table does not imply that it is advertised to BGP peers, however, every route advertised through BGP must be present in this table.

Since only routes `2.0.0.0/16` and `4.0.0.0/8` could exactly match a routing table entry, they are the only routes installed in the BGP table, as shown in Figure 8.15.

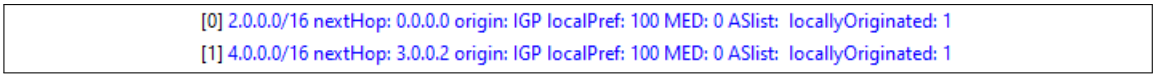


Figure 8.15: BGP Tutorial 2 - state of BGP table.

BGP Routing Table

BGP routing table stores only the best route to each destination, as a single destination can be available with multiple routes. Routes are evaluated according to local policies, and only

the routes in this table may be advertised to BGP peers. Locally configured routes (i.e., configured by the `network` command under this specific router) are always advertised to all BGP peers. However, there are some constraints placed by the BGP standard that prohibit the advertisement of certain routes learnt through BGP from other peers depending on the type of BGP session it was learned from. BGP routing table is shown in Figure 8.16.

```
[0] 2.0.0.0/16 nextHop: 0.0.0.0 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 1
[1] 4.0.0.0/16 nextHop: 3.0.0.2 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 1
```

Figure 8.16: BGP Tutorial 2 - state of BGP routing table.

Since BGP routing table is composed of the best routes from BGP table to each destination, it contains both routes `2.0.0.0/16`, `4.0.0.0/8`. What is a important detail is that the initial `next-hop` attribute for these routes is different. Because network command for route `2.0.0.0/16` matched against a routing table entry flagged as directly connected, it copied next-hop of `0.0.0.0` which is meaningless as a next-hop. If this route were to be advertised to other BGP peers, its `next-hop` attribute would always be set to the source of peering process for each particular neighbour. Route `4.0.0.0/8` in contrast to route `2.0.0.0/16` matched against a routing table entry with a remote next-hop and as such, it would use its address as a default value for the `next-hop` attribute. Depending on the type of peering, this attribute could be, again, changed to the BGP peering source.

8.3.3 Exchanging Updates

BGP uses `UPDATE` messages as a vessel for route advertisements and withdrawals. The two peers always synchronise their databases when they reach the *ESTABLISHED* state with each other and after the fact, they only exchange incremental updates: introducing new prefixes or changing or completely removing previously advertised prefixes. Routes are always advertised together with their attributes, which may be changed by the sender or even the receiver. A single `UPDATE` message can advertise multiple prefixes as long as their attributes are the same. On the other hand, withdrawal of routes is announced only by prefixes and generally speaking, a single `UPDATE` message can both advertise prefixes and withdraw different ones at the same time.

The routes received in the `UPDATE` message are installed in the BGP table and only the best route for each destination in the BGP table is installed in the BGP routing table. Any change to the BGP table requires a recalculation of the BGP routing table. If an event occurs that causes the BGP routing table to completely remove some destination, an `UPDATE` message that withdraws this destination prefix is sent to the appropriate peers.

Topology

To properly demonstrate the BGP route advertisements and withdrawals, we need at least a few routers with BGP peering. The topology, as well as the addressing scheme is shown in Figure 8.17. Routers `R0` and `R3` both participate in internal peering with `R1` and `R2`, respectively, while routers `R1` and `R2` participate in eBGP peering with each other. Each router is enabled to advertise all connected links through BGP. For example, router `R1` is configured with three `network` commands for prefixes: `10.1.1.0/24`, `10.0.0.0/24` and `2001:1111::/64`. To showcase the capabilities and quirks of Multiprotocol BGP, IPv4 and IPv6 prefixes are advertised over IPv4 peering connection. Since the focus is on the content

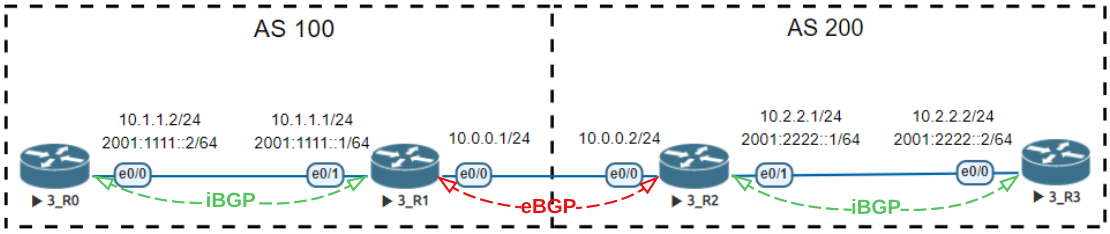


Figure 8.17: BGP Topology 3 - Exchanging Updates.

Event#	Time	Relevant Hops	Name	TxUpdate? / Source	Length / Destination	Info	Length	Info
#144	0.500	R1 ⟶ R0	SYN	10.1.1.1:50001	10.1.1.2:179	TCP	72 B	50001→179 [Syn] Seq=125000 Win=7504 | IPv4
#158	0.500'057'650	R0 ⟶ R1	SYN+ACK	10.1.1.2:179	10.1.1.1:50001	TCP	72 B	179→50001 [Syn Ack=125001] Seq=125014 Win=7
#176	0.500'115'300	R1 ⟶ R0	TcpAck	10.1.1.1:50001	10.1.1.2:179	TCP	72 B	50001→179 [Ack=125015] Seq=125001 Win=7504
#199	0.500'172'950	R0 ⟶ R1	tcpseg(l=45)	10.1.1.2:179	10.1.1.1:50001	TCP	111 B	(UNKNOWN) (inet::bgp::BgpOpenMessage) versio
#201	0.500'182'500	R1 ⟶ R0	tcpseg(l=45)	10.1.1.1:50001	10.1.1.2:179	TCP	111 B	(UNKNOWN) (inet::bgp::BgpOpenMessage) versio
#230	0.500'271'350	R0 ⟶ R1	TcpAck	10.1.1.2:179	10.1.1.1:50001	TCP	72 B	179→50001 [Ack=125046] Seq=125060 Win=7504
#231	0.500'280'900	R1 ⟶ R0	TcpAck	10.1.1.1:50001	10.1.1.2:179	TCP	72 B	50001→179 [Ack=125060] Seq=125046 Win=7504
#248	0.500'338'550	R0 ⟶ R1	tcpseg(l=19)	10.1.1.2:179	10.1.1.1:50001	TCP	85 B	(UNKNOWN) (inet::bgp::BgpKeepAliveMessage)
#249	0.500'348'100	R1 ⟶ R0	tcpseg(l=19)	10.1.1.1:50001	10.1.1.2:179	TCP	85 B	(UNKNOWN) (inet::bgp::BgpKeepAliveMessage)
#276	0.500'416'150	R0 ⟶ R1	TcpAck	10.1.1.2:179	10.1.1.1:50001	TCP	72 B	179→50001 [Ack=125065] Seq=125079 Win=7504
#277	0.500'425'700	R1 ⟶ R0	TcpAck	10.1.1.1:50001	10.1.1.2:179	TCP	72 B	50001→179 [Ack=125079] Seq=125065 Win=7504
#294	0.500'483'350	R0 ⟶ R1	tcpseg(l=160)	10.1.1.2:179	10.1.1.1:50001	TCP	226 B	(UNKNOWN) (inet::SequenceChunk) length = 160
#295	0.500'492'900	R1 ⟶ R0	tcpseg(l=165)	10.1.1.1:50001	10.1.1.2:179	TCP	231 B	(UNKNOWN) (inet::SequenceChunk) length = 165
#314	0.500'677'750	R0 ⟶ R1	TcpAck	10.1.1.2:179	10.1.1.1:50001	TCP	72 B	179→50001 [Ack=125230] Seq=125239 Win=7504
#315	0.500'687'300	R1 ⟶ R0	TcpAck	10.1.1.1:50001	10.1.1.2:179	TCP	72 B	50001→179 [Ack=125239] Seq=125230 Win=7504

Figure 8.18: BGP Tutorial 3 - Initial synchronisation of BGP databases between internal peers `R0` and `R1`.

of the BGP messages, the TCP segments that do not include any new application data are ignored. General state of the topology is mentioned at the start of both following segments.

1 - Advertising Prefixes

All BGP peering is initially shut down. Before enabling eBGP peering between routers `R1` and `R2`, let us first enable iBGP peering and observe which routes are exchanged between internal peers `R0` and `R1`. Since IPv4 and IPv6 are enabled on both devices, they announce this as two separate capabilities to each other in the `OPEN` message. Since they clearly have IPv4 and IPv6 address families in common, peering eventually reaches the *ESTABLISHED* state on both sides and both routers can exchange prefixes from their respective BGP routing tables via `UPDATE` messages. Both routers send four `UPDATE` messages: one `UPDATE` message advertising IPv4 prefix `10.1.1.0/24` (router `R1` also advertises prefix `10.0.0.0/24`), one `UPDATE` message advertising IPv6 prefix `2001:1111::/64` and two End-of-RIB messages (EoR) to indicate to the peer the completion of the initial routing update after the session is established for each address family. An analogous exchange could be observed between internal peers `R2` and `R3`. This initial exchange is shown in Figure 8.18.

The content of the BGP databases of router `R1` is shown in Figure 8.19. Routes from the aforementioned `UPDATE` message exchange were installed in the BGP table for both address families. Because of this, router `R1` has two entries for each prefix. One is directly connected, and one is learnt from `R0`. Since the locally originated route is preferred, it is installed in the BGP routing table. This is true for both IPv4 and IPv6. This is again analogous on all other routers.

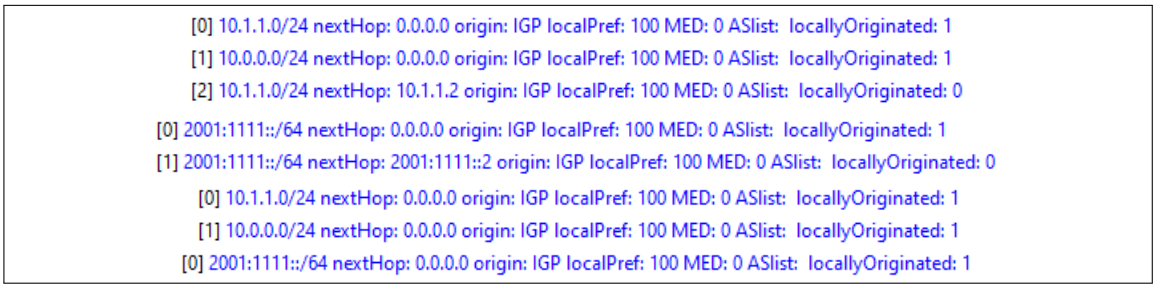


Figure 8.19: BGP Tutorial 3 - states of BGP tables and BGP routing tables respectively on router `R1` after the initial exchange with router `R0`.

Note that on router `R1`, the IPv6 route received to destination `2001:1111::/64` from router `R0` has a valid next-hop address `2001:1111::2`. Although IPv4 peering was used to exchange IPv6 reachability information, the IPv6 route is usable without additional manipulation. However, this does not have to be true for all BGP real-life implementations. IPv6 route always needs an IPv6 next-hop address. Since IPv4 was used to establish peering, BGP can set its IPv4 address as the next-hop attribute for the originated route `10.1.1.0/24`. Some implementations of BGP (e.g., Cisco) always use the same address family for the `next-hop` attribute as the address family the peering is established on if the `next-hop` attribute has to be changed. It would encode the IPv4 address in IPv6 format to create a next-hop attribute, which, most likely, would not be a valid address. It serves as an indication that the next hop should be changed explicitly, via a route-map, for example. However, in the this model, the need to always explicitly change the next-hop attribute for all routes transmitted over a different address family is removed. During the calculation of the `next-hop` attribute, the router locates the interface used as the next-hop and searches its configuration for the same address family next-hop address as is the address family of the advertised prefix. If such an address exists, it is used as next-hop. If multiple suitable addresses are present on the interface, the behaviour is not defined. If the address of the correct address family is not found on the interface, the next-hop address is encoded into the different address family. However, currently there are no means to change such a next-hop.

With iBGP sessions established in both ASes, let us connect them and observe the initial IPv4 and IPv6 route advertisement over the IPv4 connection between routers `R1` and `R2`. Because the address family capabilities are the same, the *ESTABLISHED* state is reached on both sides. Each peer is now going to advertise the content of its BGP routing table via `UPDATE` messages. There are again four `UPDATE` messages, one for route advertisement and one empty to signal that all routes have been sent for each address family. Since there are no valid IPv6 addresses for eBGP peering sources between routers `R1` and `R2`, the `next-hop` attribute for IPv6 routes contains unusable addresses as they are just encoded versions of IPv4 addresses used as peering sources.

As all IPv4 routes contained in the `UPDATE` message were installed in the BGP table and subsequently in the BGP routing table on both `R1` and `R2`, these routes can be advertised to the iBGP peers on both sides. However, IPv6 routes were not installed in the BGP routing table, as their `next-hop` attributes were not resolvable in the local routing table for reasons explained earlier. The contents of all BGP databases on router `R1` is shown in Figure 8.20. Note that IPv6 route to destination `2001:2222::/64` is in the BGP table but not in the BGP routing table.

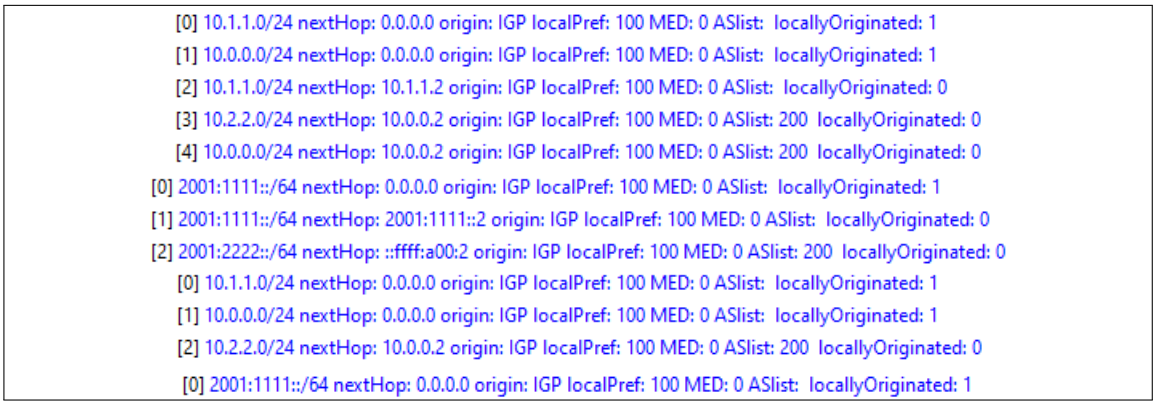


Figure 8.20: BGP Tutorial 3 - states of BGP tables and BGP routing tables respectively on router `R1` after the initial exchange with router `R2` is complete.

2 - Withdrawing Prefixes

Route withdrawal is a mechanism utilised by BGP that enables one router to inform its peer that a previously advertised prefix is no longer available through BGP and such route should be removed from internal BGP databases. Unlike prefix advertising, withdrawn routes are announced only as prefixes and do not carry route attributes. They are also announced via the `UPDATE` message in a special field.

A router can withdraw a route for the following reasons:

- receipt of `UPDATE` message with withdrawn route from a peer that previously advertised the such a route when no usable alternative route to a given destination is available;
- TCP connection to a peer is closed, causing all received routes installed in the BGP routing table from this peer to be withdrawn if no usable alternative route exists;
- a change of routing table that causes:
  a) locally originated route to no longer exactly match any entry in routing table; or
  b) next-hop for a route to be no longer resolvable;

  while no usable alternative route is present in the BGP table.

A receipt of a withdrawn route signals that the sender removed the destination from its BGP routing table. However, a route to such destination can still be present in the sender's BGP table, as BGP table stores even unusable routes. Nevertheless, the recipient of a route withdraw message always removes the particular route from both the BGP table and the BGP routing table. This withdrawal is propagated to other BGP peers via the withdrawn routes field if and only if such a route was installed in the BGP routing table and no other usable route for such destination exists in the BGP table. If there is a viable fallback route in the BGP table, the route for that specific destination is merely replaced in the BGP routing table and instead of withdrawal, peers are notified of this change by a regular `UPDATE` message, re-advertising the route with new attributes.

Figure 8.21: BGP Tutorial 3 - states of BGP tables and BGP routing tables respectively on router `R1` after the peering with router `R2` is closed.

To showcase route withdrawal, the previously converged topology is changed by removing a link between routers `R1` and `R2`. The initial state of the router `R1` is the same as in Figure 8.20.

When the link is removed, router `R1` detects that the interface towards its peer `R2` is not usable and considers the TCP connection closed. This mirrors the behaviour of Cisco devices. Router `R1` then updates its BGP table by removing routes it received from router `R2`, recalculates the BGP routing table and notifies its only peer, router `R0`.

The `UPDATE` messages it sends are withdrawing two IPv4 routes: `10.0.0.0/24` and `10.2.2.0/24`. Since router `R0` has route `10.2.2.0/24` in its BGP routing table and no alternative route is available in the BGP table, the route is removed, same goes for route `10.0.0.0/24`. Additionally, router `R1` removes IPv6 route to destination `2001:2222::/64` from its BGP table but since no changes to IPv6 BGP routing table occurred on router `R1`, no additional withdrawals via `UPDATE` messages are performed. The final state of BGP databases on router `R1` is shown in Figure 8.21.

3 - Incompatible Address Families

Since the advertised address family is somewhat (see the `next-hop` attribute above) independent of the address family used for the peering itself, let us observe what transpires if each peer wants to exchange reachability information for a different address family. Router `R1` is configured to only advertise the IPv4 address family and router `R2` is configured to only advertise the IPv6 address family. The routers exchange their supported address families with `OPEN` messages and each checks the capabilities advertised by the counterparty. Since there is no address family in common, the peering has no purpose, and thus a `NOTIFICATION` message is generated by both devices and the TCP connection is closed. No prefixes were exchanged, and routers will periodically try to establish peering again, but unless the configuration changes, that will never happen.

8.3.4 Attributes

Each BGP route is made up of two parts: prefix and attributes. While the prefix identifies the destination network, attributes convey additional information about the route. Attributes are generally variable, and their presence or absence depends on the policy of individual autonomous systems and their managing entities. However, a trio of attributes must be conveyed for each route. These are called **Well-Known Mandatory** attributes and are the following: `ORIGIN`, defining how the route was originally learnt by BGP, `AS_PATH` specifying a sequence of AS numbers that the route has traversed, and `NEXT_HOP` defining an address which should be used as a next-hop to reach the destination specified by the

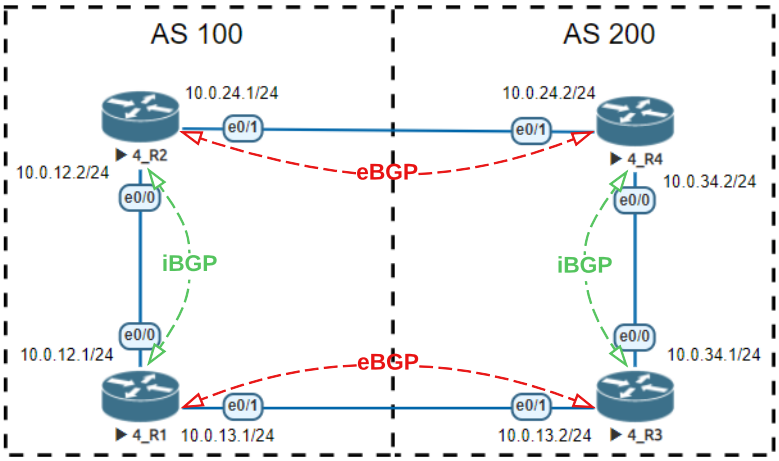


Figure 8.22: BGP Topology 4 - BGP Attributes.

prefix of the BGP route. An additional attribute called `LOCAL_PREF` is required similarly to **Well-Known Mandatory** and must be present for each route, but only in scenarios where routes are exchanged between internal BGP peers (iBGP). All other attributes are regarded as **Optional**.

Topology

The following sections focus on the general meaning of each attribute, why, when, and how it can change, and what to generally look out for when debugging undesirable behaviour. For this purpose, the topology, depicted in Figure 8.22, consists of four routers separated into two autonomous systems.

1 - Next-hop

As stated above, `NEXT_HOP` is a **Well-Known Mandatory** BGP route attribute that must be advertised for each route. Its purpose is to inform the recipient in which direction to send traffic to reach the destination advertised in the route. Generally speaking, this address does not have to be directly connected, but must be resolvable by the recipient if such a route is to be used and installed in the routing table. BGP routes whose next-hop addresses are not resolvable are not considered viable and are not advertised to other BGP peers.

`NEXT_HOP` attribute is always set when the route is locally originated, i.e., created locally by `network` command or redistribution. BGP route is always created from an entry in the routing table. Type of the entry that the BGP route is created from defines the initial value for its `NEXT_HOP` attribute. The next-hop address from the routing table entry which the route is created from is always copied. At this point, it contains either some remote next-hop address of another router or it is equal to `0.0.0.0`, meaning that the destination is directly connected on some local interface.

When a route is received via the `UPDATE` message, it always contains a remote address. This route is always placed in the BGP table. However, if it is to be installed in the BGP routing table and advertised to other BGP peers, its `NEXT_HOP` attribute must be resolvable.

Figure 8.23: BGP Tutorial 4 - states of the BGP table and the BGP routing table respectively on router R2.

When a route is advertised to an iBGP peer and the NEXT_HOP attribute is a remote address, it is just copied and sent to the neighbour. However, if the NEXT_HOP attribute is equal to 0.0.0.0, it must be changed, since 0.0.0.0 translates to *directly-connected*, which could not be true for the receiving neighbour, rendering the route unusable. For this reason, it is always changed to the local BGP peering source address, so either to the address of the 'physical' *(eth)* interface used to communicate with said peer or to the address of a local loopback *(lo)* interface, which is configured to act as a source for such connection. The change in the peering source can be achieved via the usage of the `local-source` attribute and a loopback identifier. The change of the NEXT_HOP attribute can also be enforced for a specific neighbour by specifying the attribute `next-hop-self`.

In contrast to iBGP, a route advertised through eBGP always changes the NEXT_HOP attribute to the local peering source of the sender.

So, generally speaking, when an AS learns a BGP route through eBGP, the NEXT_HOP attribute contains an address of the remote eBGP router from a remote AS. Such route is redistributed through the local AS without any changes to the NEXT_HOP attribute. And finally, this can be changed if the local eBGP router in the local AS, that receives such a route, uses a `next-hop-self` configuration attribute for its iBGP neighbours. In that case, the NEXT_HOP attribute changes to the peering source for each iBGP connection.

Because of the aforementioned behaviour, setting a `next-hop-self` attribute for a specific neighbour is only used in iBGP scenarios where the neighbour that is receiving the route through iBGP would not know where the original next-hop is.

Let us consider the topology shown in Figure 8.22 without the link between R1 and R3. In this specific case, each router has only a single `network` command configured: routers R1, R2 are configured to advertise network 10.0.12.0/24 while routers R3, R4 are configured to advertise network 10.0.34.0/24. Since the link between R1 and R3 is not present, the two autonomous systems are connected only by the link between routers R2 and R4.

When the topology is converged, let us inspect the BGP table and BGP routing table of routers R2 and R1. BGP databases of R2 are shown in Figure 8.23. Firstly, destination 10.0.12.0/24 is available both locally, with next-hop 0.0.0.0, and through R1, with next-hop 10.0.12.1, which is the peering source address of R1. Secondly, destination 10.0.34.0/24 is available via next-hop 10.0.24.2 which is the peering source address of router R4, that is directly connected. Both networks are reachable and destination network 10.0.34.0/24 is installed in the local routing table.

Let us now inspect the same BGP structures on router R1, which has established iBGP peering with router R2. These structures are shown in Figure 8.24. Firstly, destination 10.0.12.0/24 is available again both locally and via R2. On the other hand, destination 10.0.34.0/24 is present in the BGP table but it is not installed in the BGP routing table. Hence, the destination is not available and any traffic going to that destination would

Figure 8.24: BGP Tutorial 4 - states of the BGP table and the BGP routing table respectively on router R1.

not be routed. This is because router R2 did not alter the NEXT_HOP attribute of route to destination 10.0.34.0/24 and advertised address 10.0.24.2 as a next-hop. However, since router R1 does not know where such next-hop is, the route could not be installed in the BGP routing table. Such route would never be used or even advertised to any other router.

If the route to destination 10.0.34.0/24 is to be made usable, either router R2 alters the NEXT_HOP attribute to some address that is known, or router R1 must learn how to reach the network 10.0.24.0/24. Both of these solutions are achievable purely by BGP.

Solution 1 - *next-hop-self* One solution is to make the router R2 change the NEXT_HOP attribute for the neighbour R1 to its peering source, which is known by router R1. This is accomplished by adding the command shown in Listing 8.7 to the address-family-specific configuration of R2.

```
⋮
<Address-family id="Ipv4">
  ⋮
  <Neighbor address="10.0.12.1" next-hop-self="true"/>
</Address-family>
```

Listing 8.7: BGP Tutorial 4 - configuration of *next-hop-self* on router R2.

As a result, all IPv4 routes advertised to neighbour R1 will have their NEXT_HOP attribute changed to address 10.0.12.2. The resulting BGP databases of router R1 are shown in Figure 8.25. Destination 10.0.34.0/24 is now installed in both routing tables and is reachable through next-hop 10.0.12.2/24.

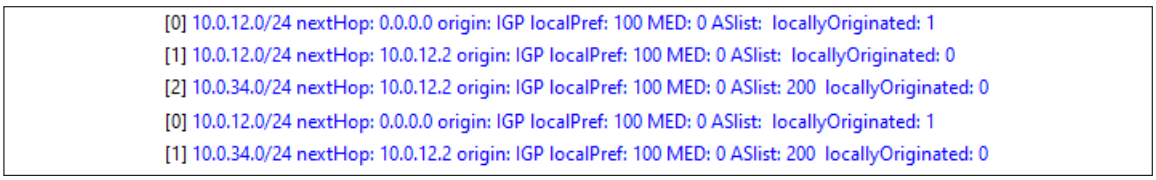


Figure 8.25: BGP Tutorial 4 - states of the BGP table and the BGP routing table respectively on router R1 after the configuration of *next-hop-self* on router R2.

Solution 2 - *Advertise Network* Another solution is to not change the NEXT_HOP attribute itself, rather let router R1 know how to reach the network 10.0.24.0/24. This can be accomplished via multiple ways: any IGP protocol running on both R1 and R2 with said network being advertised, via BGP or even with static routing. To achieve this

via BGP, network `10.0.24.0/24` has to be inserted into BGP through redistribution or `network` command on router `R2`. This cannot be accomplished on `R4` as this would result in the exact same situation. This command is shown in Listing 8.8. It inserts network `10.0.24.0/24` into `R2`'s BGP databases thus allowing it to advertise the prefix to router `R1` via iBGP.

```
⋮
<Address-family id="Ipv4">
  ⋮
  <Network address="10.0.24.0" mask="255.255.255.0"/>
</Address-family>
```

Listing 8.8: BGP Tutorial 4 - configuration of additional network on router `R2`.

This expands the BGP databases of all devices in the topology by route `10.0.24.0/24`, and it makes route `10.0.34.0/24` reachable by `R1` as shown in Figure 8.26.

```
[0] 10.0.12.0/24 nextHop: 0.0.0.0 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 1
[1] 10.0.24.0/24 nextHop: 10.0.12.2 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 0
[2] 10.0.12.0/24 nextHop: 10.0.12.2 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 0
[3] 10.0.34.0/24 nextHop: 10.0.24.2 origin: IGP localPref: 100 MED: 0 ASlist: 200  locallyOriginated: 0
[0] 10.0.12.0/24 nextHop: 0.0.0.0 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 1
[1] 10.0.24.0/24 nextHop: 10.0.12.2 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 0
[2] 10.0.34.0/24 nextHop: 10.0.24.2 origin: IGP localPref: 100 MED: 0 ASlist: 200  locallyOriginated: 0
```

Figure 8.26: BGP Tutorial 4 - states of the BGP table and the BGP routing table respectively on router `R1` after the addition of `network` command on router `R2`.

2 - AS Path

`AS_PATH` is another **Well-Known Mandatory** BGP route attribute that has two main purposes: loop detection and route selection. To understand how this is accomplished, let us first take a look at what the `AS_PATH` attribute looks like.

`AS_PATH` takes the form of a vector of AS identifiers and it can be composed out of ordered (i.e., sequence) vector and/or unordered (i.e., set) vector. Generally, by means of these vectors, this attribute informs the receiver which ASes the route has traversed and these same ASes will generally have to be traversed in order to reach said destination. The attribute usually includes the ordered vector as each AS prepends its own AS identifier into the `AS_PATH` which eventually creates the sequence. However, if a BGP router executes route aggregation (i.e., summarizes multiple prefixes with a single, less specific one), it removes all of `AS_PATH` sequences and instead, creates an unordered set which contains all unique identifiers present in any of the aggregated routes. Essentially, it is preserving the information about which ASes any of the aggregated routes have traversed. When such route is advertised to external BGP peers, a new sequence is created. In such scenario, the `AS_PATH` attribute consists of one unordered set, describing which ASes were traversed by any of the aggregated routes prior to the aggregation, and one ordered sequence, which determines which ASes the route has traversed since the aggregation.

One of the two main purposes of the `AS_PATH` attribute is loop detection. When a route is received via the BGP `UPDATE` message, the local router checks if its own AS identifier is not already present in any of the `AS_PATH` attribute vectors. If it is, the route is discarded

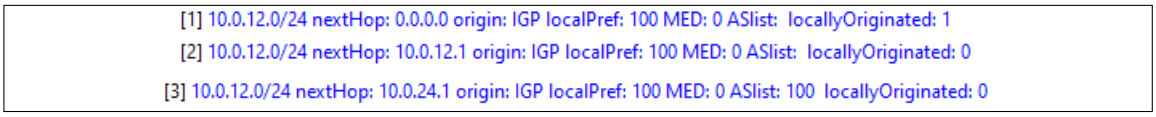


Figure 8.27: BGP Tutorial 4 - routes to destination `10.0.12.0/24` as known by routers `R2` and R4 respectively.

before being installed in any BGP database, as it signifies a routing loop has occurred. Otherwise, the route is processed as usual and is installed in the BGP table.

The second main purpose of the `AS_PATH` attribute is to serve as a route selection criterion. The length of the attribute is one of the most influential aspects of breaking the ties between routes to a same destination. Each AS identifier in the ordered sequence contributes by one towards the overall size, however, the whole unordered set always counts as a one, regardless of how many ASes are in the set. Routes with lower overall length are preferred.

The `AS_PATH` attribute is always prepended with the local AS identifier when the route is being advertised via eBGP. When the route is originated, or even received via either iBGP or eBGP, the `AS_PATH` is not altered. This means that if a route originated in the local AS is being advertised via iBGP, the `AS_PATH` attribute can be completely empty. Finally, when eBGP advertises a BGP route, the local AS identifier can be prepended multiple times. This technique is used to influence the recipient's BGP decision process if there are multiple ingress points into the local AS. This can be accomplished because shorter `AS_PATH` attributes are preferred, as mentioned earlier.

To showcase this behaviour, let us look at the BGP tables, and more specifically, the BGP routes to destination `10.0.12.0/24` on routers `R2` and `R4` on a converged topology shown in Figure 8.22. This route on both routers is shown in Figure 8.27.

Router `R2` knows two routes to destination `10.0.12.0/24`: one is locally originated and the second is learned through `R1`. Both routes originate in the local AS, and thus both have an empty `AS_PATH` attribute. Once the route is, however, advertised over eBGP to router `R4`, the `AS_PATH` attribute is prepended with the AS identifier of `R2`, i.e., 100.

3 - Local Preference

The `LOCAL_PREF` attribute is one of the `Well-Known Discretionary` attributes. It may be included in the route and it must be understood by all BGP implementations. Its general purpose is to make certain routes more locally preferable than others. This may include scenarios when a local AS wants to use a certain egress point whenever possible, as it maybe provides greater speeds or lower latency or cost. As this attribute is never advertised over eBGP, but must be included in any advertised route over iBGP, router that receives routes over eBGP often offsets the `LOCAL_PREF` attribute value before redistributing the route throughout the local AS to advocate for that route's preference. This attribute is only taken into account if, of course, there are multiple routes to a single destination.

The `LOCAL_PREF` attribute is defined as the one with the greatest influence on the BGP decision process by BGP RFC 4271; real implementations, however, most often employ some purely local attribute that supersedes `LOCAL_PREF` attribute (i.e., `WEIGHT` on Cisco devices). The default value is `100`, the higher, the better.

```
[0] 10.0.12.0/24 nextHop: 0.0.0.0 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 1
[1] 10.0.12.0/24 nextHop: 10.0.12.2 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 0
[2] 10.0.34.0/24 nextHop: 10.0.13.2 origin: IGP localPref: 100 MED: 0 ASlist: 200  locallyOriginated: 0
[3] 10.0.34.0/24 nextHop: 10.0.12.2 origin: IGP localPref: 100 MED: 0 ASlist: 200  locallyOriginated: 0

[0] 10.0.12.0/24 nextHop: 0.0.0.0 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 1
[1] 10.0.34.0/24 nextHop: 10.0.13.2 origin: IGP localPref: 100 MED: 0 ASlist: 200  locallyOriginated: 0
```

Figure 8.28: BGP Tutorial 4 - states of the BGP table and the BGP routing table respectively on router `R1`.

```
[0] 10.0.12.0/24 nextHop: 0.0.0.0 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 1
[1] 10.0.12.0/24 nextHop: 10.0.12.1 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 0
[2] 10.0.34.0/24 nextHop: 10.0.24.2 origin: IGP localPref: 100 MED: 0 ASlist: 200  locallyOriginated: 0
[3] 10.0.34.0/24 nextHop: 10.0.12.1 origin: IGP localPref: 100 MED: 0 ASlist: 200  locallyOriginated: 0

[0] 10.0.12.0/24 nextHop: 0.0.0.0 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 1
[1] 10.0.34.0/24 nextHop: 10.0.24.2 origin: IGP localPref: 100 MED: 0 ASlist: 200  locallyOriginated: 0
```

Figure 8.29: BGP Tutorial 4 - states of the BGP table and the BGP routing table respectively on router `R2`.

As a demonstration, let us consider the topology shown in Figure 8.22. All links are connected and *next-hop-self* is configured towards the internal peer on all routers. Without any changes to the `LOCAL_PREF` attribute, let us inspect the BGP databases of routers `R1` and `R2` and examine their preferred route to destination `10.0.34.0/24`. BGP databases of router `R1` are shown in Figure 8.28, while `R2`'s databases are shown in Figure 8.29.

Router `R1` prefers router `R3` as a next-hop to reach destination `10.0.34.0/24` and similarly, router `R2` prefers router `R4` as a next-hop. This is because both routes have the same `LOCAL_PREF` of `100`, `AS_PATH` length of `1`, `ORIGIN` code of `IGP` and a default `MED` of `0`. To break ties for these routes, BGP prefers routes learnt through eBGP over routers learnt through iBGP. For this reason, router `R1` prefers the route from `R3` and router `R2` the route from `R4`.

Now, let us say that we want to use the link between routers `R1` and `R3` even on router `R2` to reach the destination network `10.0.34.0/24` and that we can only alter the configuration of the routers in AS 100. Altering the `LOCAL_PREF` attribute is the perfect option here. As the higher `LOCAL_PREF` attribute of a route makes it more preferable, we even have a choice: lowering `LOCAL_PREF` attribute value on router `R2` for routes received from `R4` or raising the `LOCAL_PREF` attribute value on router `R1` for routes received from `R3`. The latter could be a better option, considering that if we add another connection between the autonomous systems, we would spare ourselves the need to lower the `LOCAL_PREF` attribute value there as well. The additional configuration of `LOCAL_PREF` attribute is shown in Listing 8.9. It configures router `R1` to set the `LOCAL_PREF` attribute of all routes received from neighbour `10.0.13.2` to `110`.

```
⋮
<Neighbor address="10.0.13.2" local-pref="110"/>
```

Listing 8.9: BGP Tutorial 4 - setting of `LOCAL_PREF` attribute to all routes received from neighbor `10.0.13.2`.

With this change, let us again inspect the BGP databases of R2. This is shown in Figure 8.30. Router R2 now newly uses the router R1 as a next-hop to reach destination 10.0.13.2.

Figure 8.30: BGP Tutorial 4 - states ofthe BGP table and the BGP routing table respectively on router R2 when LOCAL_PREF attribute is configured.

In summary, AS uses the LOCAL_PREF attribute to make some specific local AS **egress** point more preferable locally.

4 - Multi Exit Discriminator

Multi Exit Discriminator attribute, MED for short, is a Optional Non-Transitive BGP route attribute. This means that it does not have to be recognised by all BGP implementations and is passed along with the route only if understood. Similarly to the LOCAL_PREF attribute, the purpose of the MED attribute is to influence the BGP decision process by making some certain routes more preferable. However, instead of influencing the decision process in the local AS, it is used to influence the decision process of the remote AS. Unlike the LOCAL_PREF attribute, the MED attribute is added to the route by the sending eBGP peer and not by the recipient. Routers can be configured to alter the MED attribute for routes as they are advertised to a specific neighbour. If configured locally for a specific eBGP peer, the attribute is advertised only through eBGP for that specific peer but never through iBGP. The eBGP recipient of a route with the MED attribute, on the other hand, advertises this attribute to iBGP peers. However, it is never advertised outside of the borders of the receiving AS. *MED* attribute has a smaller priority when it comes to the BGP decision process, so the locally set preference set via LOCAL_PREF attribute supersedes the MED attribute. Essentially, *MED* attribute is used to 'suggest' LOCAL_PREF-like value to a neighbouring AS. Such neighbouring AS can but does not have to adhere to such a 'suggestion' as it can override the *MED* attribute with the higher-priority LOCAL_PREF attribute or ignore the MED completely, as it is purely optional.

Unlike the LOCAL_PREF attribute, the default value is 0, and a lower value makes the route more preferable. MED attribute is only relevant when there are multiple ingress points to the local AS as a means to set a specific one as more preferable than others.

Let us now showcase this behaviour on the topology shown in Figure 8.22. All links are connected and *next-hop-self* is configured towards the internal peer on all routers. Without any changes to the MED attribute, let us inspect the BGP databases of routers R3 and R4 and examine their preferred route to destination 10.0.12.0/24. BGP databases of router R3 are shown in Figure 8.31 and R4's are shown in Figure 8.32.

As was the case initially with the LOCAL_PREF attribute, router R3 prefers R1 as a next-hop while R4 prefers router R2 as a next-hop to reach destination 10.0.12.0/24. Let us now influence the decision process of AS 200 by only changing configuration in AS 100.

```
[0] 10.0.34.0/24 nextHop: 0.0.0.0 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 1
[1] 10.0.34.0/24 nextHop: 10.0.34.2 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 0
[2] 10.0.12.0/24 nextHop: 10.0.13.1 origin: IGP localPref: 100 MED: 0 ASlist: 100  locallyOriginated: 0
[3] 10.0.12.0/24 nextHop: 10.0.34.2 origin: IGP localPref: 100 MED: 0 ASlist: 100  locallyOriginated: 0
[0] 10.0.34.0/24 nextHop: 0.0.0.0 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 1
[1] 10.0.12.0/24 nextHop: 10.0.13.1 origin: IGP localPref: 100 MED: 0 ASlist: 100  locallyOriginated: 0
```

Figure 8.31: BGP Tutorial 4 - states of the BGP table and the BGP routing table respectively on router `R3`.

```
[0] 10.0.34.0/24 nextHop: 0.0.0.0 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 1
[1] 10.0.34.0/24 nextHop: 10.0.34.1 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 0
[2] 10.0.12.0/24 nextHop: 10.0.24.1 origin: IGP localPref: 100 MED: 0 ASlist: 100  locallyOriginated: 0
[3] 10.0.12.0/24 nextHop: 10.0.34.1 origin: IGP localPref: 100 MED: 0 ASlist: 100  locallyOriginated: 0
[0] 10.0.34.0/24 nextHop: 0.0.0.0 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 1
[1] 10.0.12.0/24 nextHop: 10.0.24.1 origin: IGP localPref: 100 MED: 0 ASlist: 100  locallyOriginated: 0
```

Figure 8.32: BGP Tutorial 4 - states of the BGP table and the BGP routing table respectively on router `R4`.

This configuration is shown in Listing 8.10. It configures router `R2` to set the `MED` attribute of all routes advertised to neighbour `10.0.24.2` to `50`.

```
⋮
<Neighbor address="10.0.24.2" MED="50"/>
⋮
```

Listing 8.10: BGP Tutorial 4 - setting of `MED` attribute to all routes advertised to neighbour `10.0.24.2`.

If we now inspect the BGP databases of router `R4`, as shown in Figure 8.33, the next-hop value of route `10.0.12.0/24` changed from `R2` to `R3`, meaning that router `R4` will now route traffic through the link connection between routers `R1` and `R3`.

```
[0] 10.0.34.0/24 nextHop: 0.0.0.0 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 1
[1] 10.0.34.0/24 nextHop: 10.0.34.1 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 0
[2] 10.0.12.0/24 nextHop: 10.0.24.1 origin: IGP localPref: 100 MED: 50 ASlist: 100  locallyOriginated: 0
[3] 10.0.12.0/24 nextHop: 10.0.34.1 origin: IGP localPref: 100 MED: 0 ASlist: 100  locallyOriginated: 0
[0] 10.0.34.0/24 nextHop: 0.0.0.0 origin: IGP localPref: 100 MED: 0 ASlist:  locallyOriginated: 1
[1] 10.0.12.0/24 nextHop: 10.0.34.1 origin: IGP localPref: 100 MED: 0 ASlist: 100  locallyOriginated: 0
```

Figure 8.33: BGP Tutorial 4 - states of the BGP table and the BGP routing table respectively on router `R4` after `MED` attribute is configured.

In summary, AS uses the `MED` attribute to make some specific local AS **ingress** point more preferable for the neighbouring AS.

Chapter 9

Conclusion

This Master's thesis discussed two simulation models of dynamic routing protocols, EIGRP and BGP. To provide the reader with the necessary theoretical background, the first few sections described the protocols themselves, highlighting the most notable features and use cases. Furthermore, subsequent chapters provided the reader with the necessary context regarding the previous state of their simulation models. The structures of these models, as well as their issues, were listed in order. The next chapters described the origin and resolution of the mentioned issues.

EIGRP model proved to be of high quality. The problems were fairly minor, and most of the work focussed on integration with the new INET API. The resulting simulation model was submitted, merged with the INET code base, and subsequently released in INET v4.3.

As for the BGP model, the initial goal was to combine the two already available versions, Novák's version and the INET version. However, their assessments showed that both had some fatal flaws. If these two versions were just merged together, the resulting model would be largely inferior to other simulation models present in INET. Among other things, there would be a lot of redundant code difficult to maintain, and the model's capability to react to topology changes would still be very limited. Due to these findings, a considerable portion of the BGP simulation model was rewritten. Novák's version of the model was used only as a reference, and the INET version served as a new base. The resulting model more closely reflects the behaviour of real BGP implementations and allows for more complex simulation scenarios as well. This model has yet to be submitted for a merge into the INET framework.

A set of 'routing tutorials' was created as a means to verify the validity of both simulation models. An extended version of these tutorials is planned for publication on the INET website, where they can be easily accessed by the OMNeT++/INET community. Additionally, these tutorials provide the reader with a detailed understanding of the protocols' and models' behaviour. Each tutorial is accompanied by a reproduction package, allowing the reader to verify the findings. As an additional value, this work shows how these two protocols can be configured on Cisco devices.

The EIGRP portion of my work was presented by myself and my supervisor at the OMNeT++ Community Summit 2020[1] and was in a form of contribution part of the Student Conference Excel@FIT 2021[2]. Discussion around the BGP model held a special 'hackathon'

[1] https://summit.omnetpp.org/archive/2020
[2] https://excel.fit.vutbr.cz

session during the OMNeT++ Community Summit 2021[3] and it was also in a form of a contribution present at Excel@FIT 2022.

In conclusion, the two created simulation models are superior to their predecessors. One of them has already been merged with the popular framework INET and the other is missing only a few minor features requested by the INET maintainers. When these issues are resolved, this new model, together with the 'routing tutorials' will be submitted as well.

[3] https://summit.omnetpp.org/archive/2021

Bibliography

[1] Amoozadeh, M. *Pull request #381 - Bgp improvements* [online]. 2018 [cit. 2022-05-12]. Available at: https://github.com/inet-framework/inet/pull/381.

[2] ANSAINET. *Pull request #452 - BGPv4 additional implementation* [online]. 2019 [cit. 2022-05-12]. Available at: https://github.com/inet-framework/inet/pull/452.

[3] ANSAINET. *Pull request #570 - ANSAINET EIGRP* [online]. 2020 [cit. 2022-05-12]. Available at: https://github.com/inet-framework/inet/pull/570.

[4] ANSAINET. *ANSAINET EIGRP Simulation Model* [online]. 2022 [cit. 2022-05-12]. Available at: https://github.com/kvetak/ANSA/tree/aebe5bb25777da8fadd99ae62da893ff7bfc0b11/src/ansa/routing/eigrp.

[5] ANSAINET. *ANSAINET Framework* [online]. 2022 [cit. 2022-05-12]. Available at: https://ansa.omnetpp.org/.

[6] ANSAINET. *BGP Simulation Model* [online]. 2022 [cit. 2022-05-12]. Available at: https://github.com/AwziNihilist/inet/tree/topic/ANSA-BGP/src/inet/routing/bgpv4.

[7] Bates, T., Chan, E. and Chandra, R. *BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP)* [Internet Requests for Comments]. RFC 4456. RFC Editor, 2006. 1-12 p. Available at: https://datatracker.ietf.org/doc/html/rfc4456.

[8] Bloudicek, J. *Modelation of Routing Protocol EIGRP*. Brno, CZ, 2014. Master's Thesis. Brno University of Technology, Faculty of Information Technology. Available at: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=119392.

[9] Bush, R., Patel, K. and Ward, D. *Extended Message Support for BGP* [Internet Requests for Comments]. RFC 8654. RFC Editor, 2019. Available at: https://datatracker.ietf.org/doc/html/rfc8654.

[10] Cisco Networking Academy. *Introduction to Routing Dynamically, Chapters 3-13* [online]. Mar 2014 [cit. 2022-05-12]. Available at: https://www.ciscopress.com/articles/article.asp?p=2180210.

[11] Cisco Systems Inc. *BGP Best Path Selection Algorithm* [online]. 2016 [cit. 2022-05-12]. Available at: https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/13753-25.html.

[12] Cisco Systems Inc. *IP Routing: EIGRP Configuration Guide* [online]. Feb 2018 [cit. 2022-05-12]. Available at: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_eigrp/configuration/15-mt/ire-15-mt-book.html.

[13] Cisco Systems Inc. *IP Routing: BGP Configuration Guide* [online]. Sep 2019 [cit. 2020-12-31]. Available at: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_bgp/configuration/xe-16/irg-xe-16-book/configuring-a-basic-bgp-network.html.

[14] Halabi, S. *Internet Routing Architecture*. 2nd ed. Cisco Press, 2000. ISBN 1-57870-233-X.

[15] Hornig, R. *TCP simultaneous open don't work* [online]. 2015 [cit. 2022-05-12]. Available at: https://github.com/inet-framework/inet/issues/92.

[16] IDC. *Global Ethernet Switch and Router Markets Deliver Mixed Results in Q2 2020* [online]. Sep 2020 [cit. 2020-12-31]. Available at: https://www.idc.com/getdoc.jsp?containerId=prUS48943122.

[17] INET. *L3Address Generic Interface* [online]. 2017 [cit. 2022-05-12]. Available at: https://doc.omnetpp.org/inet/api-old/doxy/classinet_1_1_l3_address.html.

[18] INET. *Developer's Guide* [online]. 2022 [cit. 2022-04-18]. Available at: https://inet.omnetpp.org/docs/developers-guide.

[19] INET. *EIGRP Simulation Model* [online]. 2022 [cit. 2022-05-12]. Available at: https://github.com/inet-framework/inet/tree/master/src/inet/routing/eigrp.

[20] INET. *INET BGPv4 Simulation Model* [online]. 2022 [cit. 2022-05-12]. Available at: https://github.com/inet-framework/inet/tree/bd9bfcb90ca1dcdbdbb8d9c252f432d80889251d/src/inet/routing/bgpv4.

[21] INET. *INET Framework* [online]. 2022 [cit. 2022-05-12]. Available at: https://inet.omnetpp.org.

[22] INET. *INET Tutorials* [online]. 2022 [cit. 2022-05-12]. Available at: https://inet.omnetpp.org/docs/tutorials/.

[23] Juniper Networks, Inc. *Understanding BGP Path Selection* [online]. 2021 [cit. 2022-05-12]. Available at: https://www.juniper.net/documentation/us/en/software/junos/vpn-l2/bgp/topics/concept/routing-protocols-address-representation.html.

[24] Maigron, P. *World - Autonomous System Number statistics* [online]. 2022 [cit. 2022-05-12]. Available at: https://www-public.imtbs-tsp.eu/~maigron/RIR_Stats/RIR_Delegations/World/ASN-ByNb.html.

[25] Novak, A. *Modeling and Simulation of BGP*. Brno, CZ, 2019. Master's Thesis. Brno University of Technology, Faculty of Information Technology. Available at: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=197475.

[26] Odom, W. *CCNA Routing and Switching 200-125 Official Cert Guide Library*. 1st ed. 2016 [cit. 2022-05-12]. 1-1600 p. ISBN 978-1-58720-581-1.

[27] OpenSim Ltd. *OMNeT++ Simulation Manual* [online]. 2022 [cit. 2022-05-12]. Available at: https://doc.omnetpp.org/omnetpp/manual/.

[28] OpenSim Ltd. *Simulator OMNeT++* [online]. 2022 [cit. 2022-05-12]. Available at: https://omnetpp.org.

[29] Rekhter, Y., Li, T. and Hares, S. *A Border Gateway Protocol 4 (BGP-4)* [Internet Requests for Comments]. RFC 4271. RFC Editor, 2006. 1-104 p. Available at: https://tools.ietf.org/html/rfc4271.

[30] Roser M., Ritchie H., Ortiz-Ospina E. The Internet's history has just begun. *Our World in Data* [online]. 2015. Available at: https://ourworldindata.org/internet.

[31] Savage, D., Ng, J., Moore, S., Slice, D., Paluch, P. et al. *Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP)* [Internet Requests for Comments]. RFC 7868. RFC Editor, 2016. 1-104 p. Available at: https://tools.ietf.org/html/rfc7868.

[32] Stroustrup, B. *Bjarne Stroustrup's C++ Glossary* [online]. 2012 [cit. 2022-05-12]. Available at: https://www.stroustrup.com/glossary.html.

[33] Sun Microsystems, Inc. *Compiling C++ Templates* [online]. 2000 [cit. 2022-05-12]. Available at: https://docs.oracle.com/cd/E19957-01/806-3572/Templates.html.

[34] The C++ Resources Network. *C++ Templates* [online]. 2020 [cit. 2022-05-12]. Available at: https://www.cplusplus.com/doc/oldtutorial/templates/.

[35] Veselý, V., Bloudíček, J. and Ryšavý, O. Proceedings of the 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2014). In:. Wien, Austria: [b.n.], 2014, p. 50–58 [cit. 2022-05-12]. ISBN 978-989-758-045-1.

[36] Veselý, V., Rek, V. and Ryšavý, O. Enhanced Interior Gateway Routing Protocol with IPv4 and IPv6 Support for OMNeT++. In:. 2015, p. 65–82 [cit. 2022-05-12]. ISSN 2194-5357.

[37] Zhang, B., Kambhampati, V., Massey, D., Oliveira, R., Pei, D. et al. *A secure and scalable Internet routing architecture (SIRA)*. Pisa, Italy: [b.n.], 2006 [cit. 2022-05-12].

Appendix A

Contents of Included Disk Media

File / Folder	Description
thesis.pdf	Copy of this thesis in PDF format.
/src	Folder containing INET 4.3 with both simulation models.
/tutorials/omnet/	Folder containing tutorial simulation files for OMNeT++.
/tutorials/cisco/	Folder containing tutorial Cisco configuration files.
readme.md	Short description of contents.

Table A.1: Contents of the included disk media.

Appendix B

Sequence Diagram of EIGRP Dynamic Neighbour Discovery and Initial Route Exchange

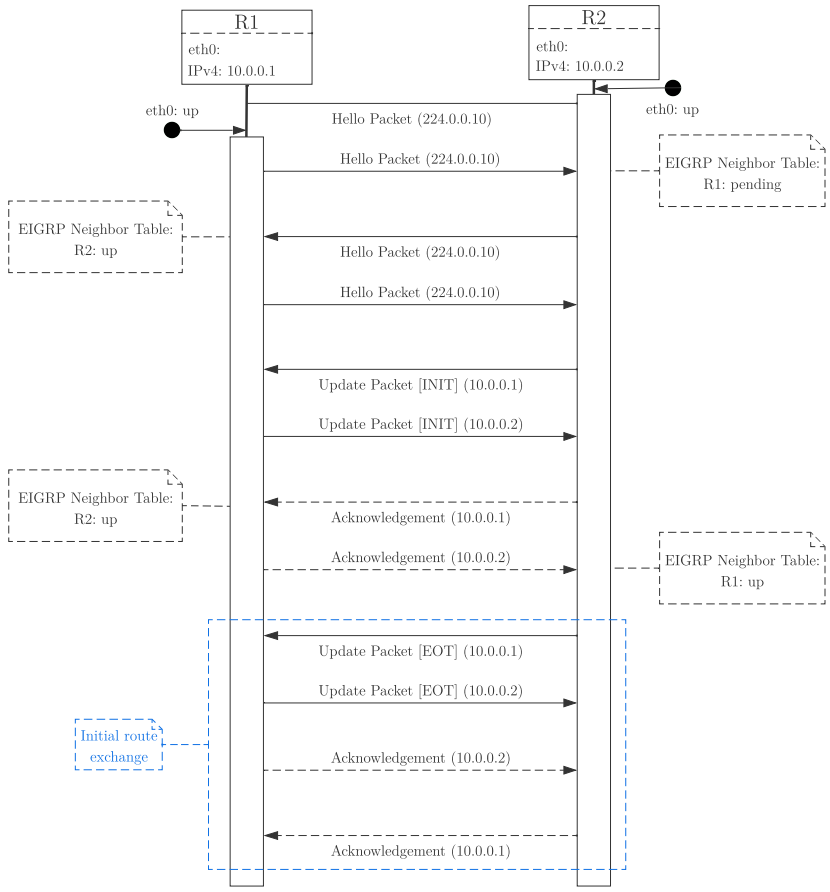


Figure B.1: Sequence diagram of the EIGRP neighbour establishing process. Acknowledgements to received messages can be piggybacked into any unicast message to the given neighbour. However, this diagram shows explicit acknowledgements.

Appendix C

Sequence Diagram of BGP Session Establishment

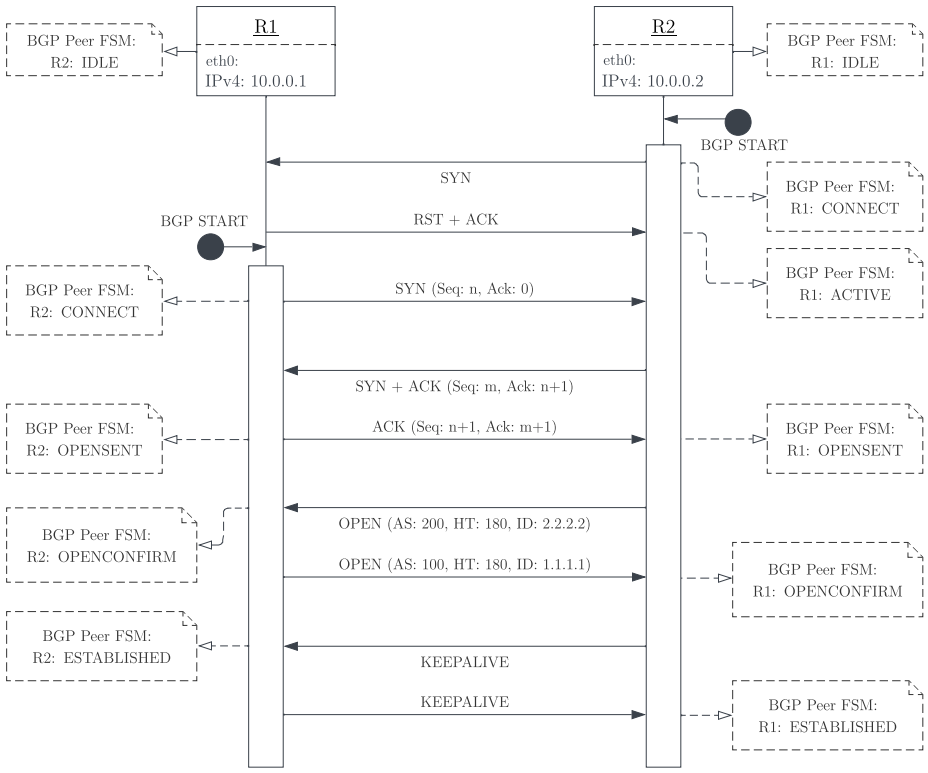
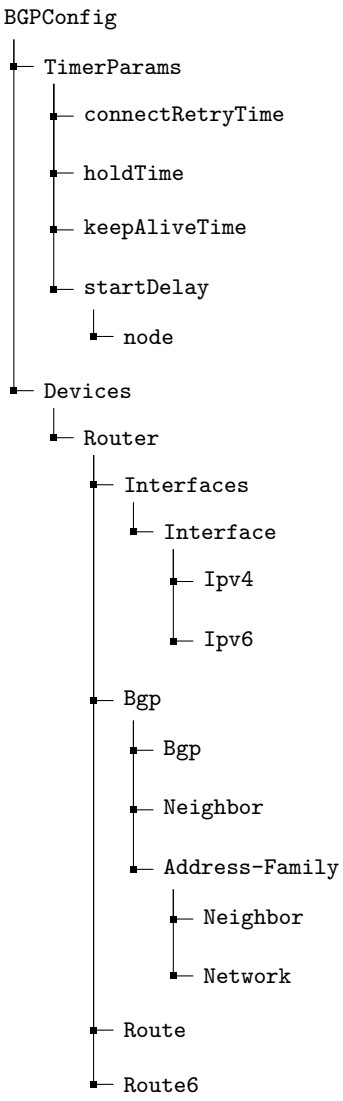


Figure C.1: Sequence diagram modeling a example of TCP and BGP communication between routers `R1` and `R2` as implemented in the BGP model. All TCP segments without any application data (i.e., TCP Acknowledgements) or retransmissions are disregarded after the three-way handshake is complete.

Appendix D

Structure of the BGP Configuration File



Appendix E

Example of BGP configuration file

```
<BGPConfig>

  <!-- optional Global BGP Timers -->
  <TimerParams>
    <connectRetryTime>120</connectRetryTime>
    <holdTime>180</holdTime>
    <keepAliveTime>60</keepAliveTime>
    <startDelay>
        <node id="R2">0.5</node>
    </startDelay>
  </TimerParams>

  <!-- List of routers -->
  <Devices>

   <!-- Single device BGP configuration -->
    <Router name="R1" id="1.1.1.1">
      <Interfaces>
        <Interface id="lo0">
          <!-- IP + mask -->
          <Ipv6 address="2001:db8::1/64"/>
          <Ipv4 address="1.1.1.1" mask="255.255.255.255"/>
        </Interface>
      </Interfaces>
      <!-- BGP specific settings -->
      <Bgp as="100">
        <Bgp router-id="11.11.11.11"/>
        <Neighbor address="10.0.0.2" remote-as="200"/>
        <Neighbor address="10.0.0.2" disable-connected-check="true"/>
        <Neighbor address="10.0.0.2" ebgp-multihop="2"/>
        <Neighbor address="10.0.0.2" update-source="lo0"/>
        <Neighbor address="10.0.0.2" local-pref="150"/>
        <Neighbor address="10.0.0.2" MED="10"/>
        <!-- IPv4 settings -->
        <Address-family id="Ipv4">
          <Neighbor address="10.0.0.2" activate="true"/>
          <Neighbor address="10.0.0.2" next-hop-self="true"/>
          <Network address="1.1.1.1" mask="255.255.255.255"/>
        </Address-family>
```

```
        <!-- IPv6 settings -->
        <Address-family id="Ipv6">
          <Neighbor address="10.0.0.2" activate="true"/>
          <Neighbor address="10.0.0.2" next-hop-self="true"/>
          <Network address="2001:db1::/64"/>
        </Address-family>
      </Bgp>
      <!-- Static routes -->
      <Route prefix="10.0.0.0" mask="255.0.0.0" int="eth0" nexthop="10.0.0.1"/>
      <Route6 prefix="2001:db8::/64" int="eth0" nexthop="2001:db8:11::1"/>
    </Router>
  </Devices>
</BGPConfig>
```

Appendix F

Relation of the Activate Attribute and IP Address Families

```
<!-- IPv4 infrastructure -->
  ⋮
  <Neighbor address="{ipv4_peer}".../>

  <!-- Start advertisement of IPv4 prefixes-->
    <!-- Implicit -->

  <!-- Stop advertisement of IPv4 prefixes-->
  <Address-family id="Ipv4">
    <Neighbor address="{ipv4_peer}" activate="false"/>
    ⋮
  </Address-family>

  <!-- Start advertisement of IPv6 prefixes -->
  <Address-family id="Ipv6">
    <Neighbor address="{ipv4_peer}" activate="true"/>
    ⋮
  </Address-family>

  <!-- Stop advertisement of IPv6 prefixes -->
    <!-- Implicit -->

<!-- IPv6 infrastructure -->
  ⋮
  <Neighbor address="{ipv6_peer}".../>

  <!-- Start advertisement of IPv4 prefixes-->
  <Address-family id="Ipv4">
    <Neighbor address="{ipv6_peer}" activate="true"/>
    ⋮
  </Address-family>

  <!-- Stop advertisement of IPv4 prefixes-->
  <Address-family id="Ipv4">
    <Neighbor address="{ipv6_peer}" activate="false"/>
    ⋮
  </Address-family>
```

```
<!-- Start advertisement of IPv6 prefixes -->
<Address-family id="Ipv6">
  <Neighbor address="{ipv6_peer}" activate="true"/>
  ⋮
</Address-family>

<!-- Stop advertisement of IPv6 prefixes -->
<Address-family id="Ipv6">
  <Neighbor address="{ipv6_peer}" activate="false"/>
  ⋮
</Address-family>
```